KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutus

Jani Koski

AZURE MACHINE LEARNING STUDIO: KUVANTUNNISTUS NEUROVERKKOTOTEUTUKSENA

Opinnäytetyö
Toukokuu 2020

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Toukokuu 2020** **Tietojenkäsittelyn koulutus**

Tekijä
Jani Koski

Nimeke
Azure Machine Learning Studio: Kuvantunnistus neuroverkkototeutuksena.

Toimeksiantaja
Automation in Network

Tiivistelmä

Tämän opinnäytetyön aiheena oli kuvantunnistuksen toteuttaminen Azure Machine Learning Studion avulla. Tavoitteena oli toteuttaa kuvantunnistustehtävään soveltuvia neuroverkkoja, kuvata neuroverkkojen hyödyntämistä web-palveluina sekä esitellä näihin tavoitteisiin liittyviä Azure Machine Learning Studion ominaisuuksia. Lisäksi tavoitteena oli arvioida Azure Machine Learning Studion käytettävyyttä työskentelystä saatujen käyttökokemusten pohjalta.

Tavoitteiden toteuttamiseksi kehitettiin numeroita tunnistava web-sovellus, joka käyttää Azure Machine Learning Studiossa kehitettyjä web-palveluita. Raportin tietoperustallinen osuus käsittelee neuroverkkojen perusrakennetta ja toiminnallisuutta sekä Azure Machine Learning Studiossa rakennettavien kuvantunnistusprojektien työnkulkuja. Toteutuksen osuudessa kuvataan toteutetut web-palvelut ja web-sovelluksen toiminnallisuus. Lisäksi näistä tehtävistä saatujen kokemusten perusteella arvioitiin Azure Machine Learning Studion käytettävyyttä.

Kehitystyön tuloksena web-sovellus kommunikoi web-palveluiden kanssa suunnitellusti. Azure Machine Learning Studion tarjoama laskentakapasiteetti saattaa rajoittaa isompien projektien toteuttamista, mikä on hyvä ottaa huomioon projekteja suunniteltaessa.

Kieli	Sivuja	36
	Liitteet	1
suomi	Liitesivumäärä	2

Asiasanat

neuroverkko, kuvantunnistus, azure machine learning studio

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2020
Degree Programme in Business Information Technology

Author
Jani Koski

Title
Azure Machine Learning Studio: Image Recognition Using Neural Networks.

Commissioned by
Automation in Network

Abstract

The topic of this thesis was image recognition using Azure Machine Learning Studio. The objectives were fourfold. Firstly, to develop neural networks for image recognition task. Secondly, to illustrate the utilization of these projects as web services. Thirdly, to present features and workflows in Azure Machine Learning Studio that relate to the aforementioned objectives. And finally, to assess the usability of Azure Machine Learning Studio based on cumulated experiences from working with the prior objectives.

To carry out these objectives a number recognizing web application was developed that implements web services developed in Azure Machine Learning Studio. The knowledge base of this report is concerned with the structure and functionality of neural networks as well as common workflows for image recognition tasks in Azure Machine Learning Studio. The functional part of the report presents the neural network configurations that were implemented as web services and the functionality of the web application. Finally, a short review is made about the usability of Azure Machine Learning Studio.

As a result, the web application communicates with the web services as planned. It should be taken into account that the computing capacity of Azure Machine Learning Studio might restrict the development of larger projects.

Language	Pages	36
	Appendices	1
Finnish	Pages of Appendices	2

Keywords

neural network, image recognition, azure machine learning studio

Sisältö

Lyhenteet

AMLS	Azure Machine Learning Studio.
NPM	Node Package Manager, Node.js:n paketinhallintajärjestelmä.

1 Johdanto

Kuvantunnistusta hyödyntävät erilaiset sovellukset ovat viime vuosina yleistyneet monilla yhteiskunnan aloilla. Terveydenhuollon kuvantamissovellutukset, robotiikka, lisätyn todellisuuden pelit ja turvallisuusalan sovellutukset ovat joitakin esimerkkejä monista kuvantunnistuksen käyttökohteista. Myös tulevaisuus näyttää lupaavalta. Kuvantunnistus on keskeinen osa ihmisen jokapäiväistä toiminnallisuutta ja sen koneellinen tuottaminen mahdollistaa esimerkiksi lukuisten tätä toimintoa tarvitsevien rutiinien automatisoinnin. Tarvetta kuvantunnistusta hyödyntäviin moninaisiin sovellutuksiin tulee näin ollen todennäköisesti olemaan jatkossakin.

Tämän opinnäytetyön tavoitteena oli toteuttaa AMLS:lla kuvantunnistustehtävään koulutettuja neuroverkkoja, kuvata neuroverkkojen hyödyntämistä web-palveluina sekä esitellä tähän aiheeseen liittyviä AMLS:n ominaisuuksia. Lisäksi tavoitteena oli arvioida AMLS:n käytettävyyttä saatujen käyttökokemusten perusteella. Opinnäytetyössä on käytetty AMLS:n ilmaisversiota.

Työn toimeksiantajana on Automation in Network -hanke, joka pyrkii uudistamaan sekä tehostamaan ammattikorkeakoulujen automaatiotekniikan opetusta ja opiskelua (Seinäjoen ammattikorkeakoulu 2020). Opinnäytetyön on tarkoitus tuottaa sisältöä, jonka avulla voidaan arvioida AMLS:n soveltuvuutta opetuskäyttöön.

Seuraava luku taustoittaa koneoppimisen käsitettä pääpiirteittäin. Luvussa 3 esitellään keinotekoisen neuronin ja neuroverkon rakennetta ja toimintoja. Tarkempi katsaus luodaan konvoluutioneuroverkon rakenteeseen, koska ne ovat kuvantunnistuksen saralla tämän hetken menestyksekkäimpiä neuroverkkorakenteita. Luvussa 4 esitellään AMLS:n työympäristöä, web-palveluita ja aiheeseen liittyviä työnkulkuja. Luvussa 5 esitellään kuvantunnistusprojektit, joista luotiin web-palvelut sekä web-sovellus, joka käyttää näitä palveluita. Lukuun sisältyy myös AMLS:n käytettävyysarvio. Luvussa 6 pohdin missä onnistuin, mitä olisi pitänyt tehdä paremmin sekä mahdollisen jatkotutkimuksen aiheen.

2 Koneoppiminen

Koneoppiminen sijoittuu tietojenkäsittelytieteessä laajemman termin keinoäly alle, joka ideana ulottuu tuhansien vuosien taakse. Varsinaisen nykyaikaisen keinoälyn idean isäksi kutsutaan Alan Turingia (1950), joka pohti artikkelissaan Computing Machinery and Intelligence, miten päätellä, voiko kone ajatella. Artikkelissa hänen lähtökohtansa koneelliseen ajatteluun käsittivät sellaiset koneelliset esitykset, jotka päällisin puolin imitoivat ihmisenkaltaista toimintaa (Turing 1950, 8). Kun esimerkiksi tietokonepeleissä älykäs agentti havaitsee pelaajan ja lähtee hyökkäämään tätä kohti, kyseessä on usein vain yksinkertaisempi tilakone, jonka toiminta ei liity varsinaiseen koneoppimisprosessiin, vaan tieto pelaajasta välitetään muuttujien avulla. Tapahtumasta voidaan kuitenkin käyttää termiä keinoäly sen näennäisen ihmisenkaltaisen toiminnan ansiosta.

Koneoppiminen on keinoälyn alakategoria, joka tutkii koneellisen oppimisen mahdollistavia algoritmejä (Segaran 2007, 3). Koneoppimisen teoreettisessa ytimessä ovat matematiikka, tilastotiede sekä tietojenkäsittelytiede, mutta lisäksi useat eri tieteenalat ovat olennaisesti vaikuttaneet koneoppimisalgoritmien kehitykseen tarjoamalla strategioitaan erityyppisten ongelmien ratkaisemiseen. Esimerkiksi neurotiede, filosofia, logiikka, evoluutiobiologia ja psykologia ovat syvällisesti vaikuttaneet koneoppimisen suuntauksiin ja sitä kautta algoritmien rakenteeseen (Mueller & Massaron 2016, 29).

Koneoppiminen jaotellaan tavallisesti kolmeen tyyppiin: ohjattuun, ohjaamattomaan ja vahvistettuun oppimiseen. Ohjatussa oppimisessa algoritmille syötetään sen koulutusvaiheessa jokaisen datanäytteen mukana oikea haluttu tuloste, jota kohti algoritmi pyrkii sopeuttamaan toimintaansa. Ohjaamattomassa oppimisessa ei tarjota valmiita oikeita vastauksia, vaan algoritmi löytää itsenäisesti niin sanottuja sarjoja datasta. Vahvistetussa oppimisessa algoritmia koulutetaan epäsuoran palautteen avulla.

Tyypillinen kuvantunnistuksen tehtävä on ohjattuun oppimiseen sisältyvä moniluokittelu, jossa kuvanäytteet luokitellaan ennalta määriteltyihin objektiluokkiin

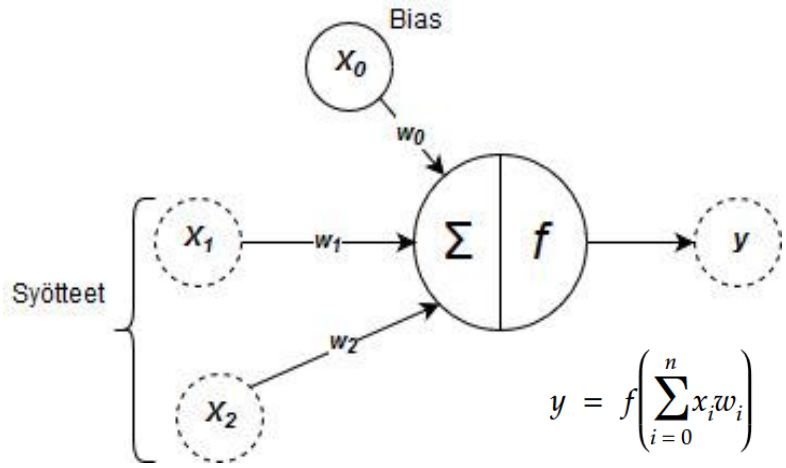
kuten kirjaimiin, esineisiin tai muihin reaalimaailman objekteihin. Luokittelevan algoritmin koulutuksen tavoitteena on generalisaatio, yleistetty **malli**, joka koulutuksen jälkeen kykenee suoriutumaan tehtävästään tarkoituksenmukaisesti uutta dataa saadessaan.

3 Neuroverkot

Neuroverkot koostuvat kerroksittain asetelluista keinotekoisista neuroneista, joita kutsutaan myös perseptroneiksi, yksiköiksi ja solmuiksi. Jatkossa käytän keinotekoisesta neuronista nimitystä ”solmu”. Seuraavaksi esitetään katsaus solmun ja neuroverkkojen toiminnallisuuteen ja rakenteisiin.

3.1 Solmun rakenne

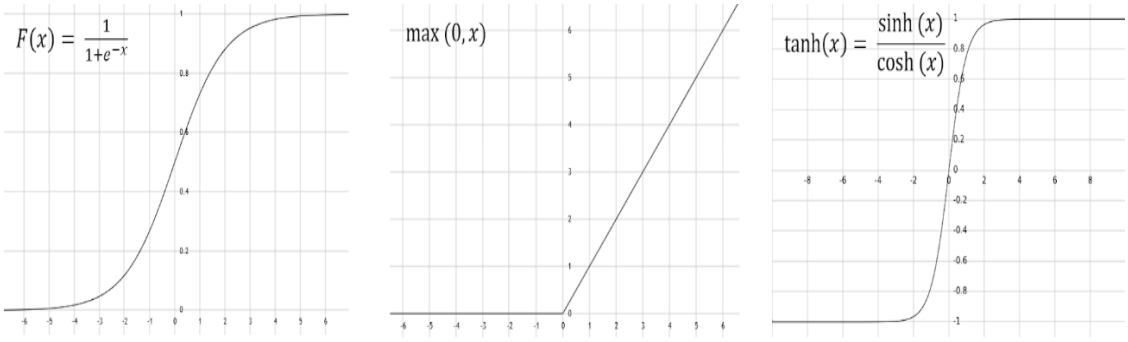
Kuvassa 1 solmu ottaa vastaan syötteet x_1 ja x_2, jotka kerrotaan omilla painokertoimillaan w. Summain ∑ summaa tulot ja epälineaarinen aktivointifunktio f käsittelee saadun summan, josta muodostuu solmun ulostuloarvo y. (Hagan 2014, 2-2.)



Kuva 1. Solmun rakenne (Kuva: Jani Koski).

3.2 Aktivointifunktiot

Aktivointifunktioiden tehtävä on muuntaa syötearvojen käsittelyn jälkeinen summa valitun epälineaarisen kaavan avulla. Aktivointifunktion avulla muunnetaan summaimen arvo ja määritellään eteenpäin syötettävien arvojen rajat. Kuvassa 2 vaaka-akseli ilmaisee aktivointifunktioon syötettäviä arvoja ja pystyakseli aktivointifunktion tulostamaa arvoa.

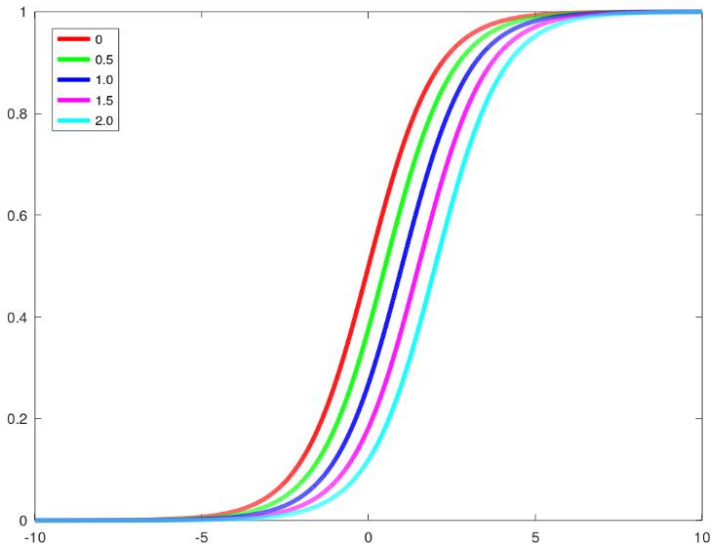


Kuva 2. Sigmoid, ReLU ja Tanh -aktivointifunktiot (Kuva: Jani Koski).

Yleisiä käytössä olevia aktivointifunktioita ovat esimerkiksi Sigmoid, ReLU (Rectified Linear Unit) ja Tanh. Aktivointifunktion valintaan vaikuttavat esimerkiksi ratkaistavan ongelman laatu sekä käytössä olevat arvorajat.

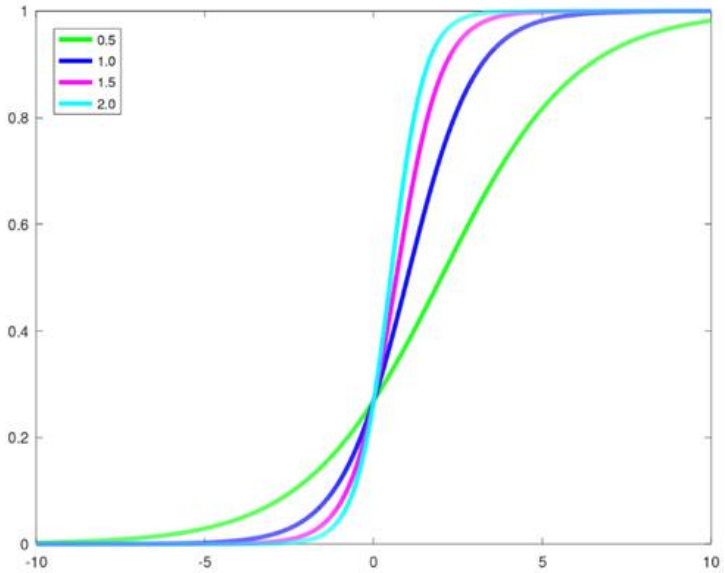
Softmax-aktivointifunktio, joka edellisistä poiketen on riippuvainen muiden saman kerroksen solmujen arvoista, on hyödyllinen neuroverkon viimeisen kerroksen solmuissa, kun tulokseksi halutaan saada todennäköisyys eri luokille. Se muuntaa viimeisen kerroksen solmuille tulleet arvot niiden välisille suhteellisille osuuksille, joiden summa on yksi. Luokittelijan antama todennäköisin luokitus on silloin sen solmun liukulukuarvo, joka on lähimpänä numeroa yksi.

Aktivointifunktion säätely tapahtuu vakiotermin ja solmun painokertoimien avulla. Vakiotermin painotetuilla arvoilla säädetään aktivointifunktion käyrää oikealle tai vasemmalle erilaisten kynnysarvojen tehokkaamman käsittelyn mahdollistamiseksi (kuva 3) (Mohammed, Khan & Bashier 2016).



Kuva 3. Vakiotermin painokertoimen vaikutus sigmoid-aktivointifunktioon (Kuva: Jani Koski).

Syötearvojen painokertoimilla puolestaan vaikutetaan aktivointifunktion jyrkkyyteen (kuva 4).



Kuva 4. Syötearvon painokertoimen vaikutus sigmoid-aktivointifunktioon (Kuva: Jani Koski).

3.3 Neuroverkkojen rakenteet

Verkostossa on erityyppisiä solmuja kerroksittain. Solmut ja kerrokset voidaan jaotella karkeasti kolmeen päätyyppiin, joista jokaisella on oma tehtävänsä: Ensimmäisenä on syötekerros, sen jälkeen piilokerrokset ja viimeisenä ulostulokerros.

Syötekerroksen syötesolmut saavat sisääntuloarvonsa suoraan käytettävästä datalähteestä. Ne poikkeavat muista solmuista siinä, että ne ottavat vastaan syötteen suoraan alkuperäisestä datanäytteestä sekä välittävät sen sellaisenaan eteenpäin ilman painokertoimen, summaimen ja aktivaatiofunktion suorittamaa muokkausta. Syötesolmuja tulee olla sama määrä kuin käytettävässä datassa on piirteitä.

Piilokerrokset tekevät varsinaisen oppimistyön optimointialgoritmien avulla. Piilokerroksia on erityyppisiä. Luvuissa 3.3.1 ja 3.3.2 käsitellään piilokerroksista tarkemmin täysin kytketty kerros, alinäytteistyskerros sekä konvoluutiokerros.

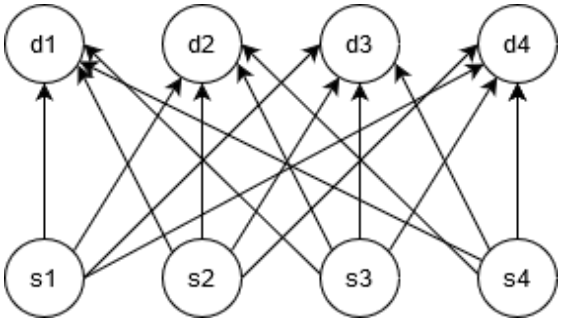
Ulostulokerros on verkoston viimeinen solmukerros ja se esittää lopulliset verkon laskemat arvot datasyötteelle. Luokittelijan tapauksessa sen sisältämien solmujen lukumäärän tulee vastata ennalta tiedettyjen luokkien määrää.

Suurella ja monimutkaisella verkkorakenteella voidaan tavoitella monimutkaisia kuvantunnistusoperaatioita. Jos tehtävä on yksinkertainen, ei kuitenkaan kannata luoda monimutkaista verkkotopologiaa. Suositeltava käytäntö on käyttää minimimäärää solmuja ja kerroksia, jotka johtavat riittävään tarkkuuteen algoritmille tarkoitetun tehtävän suorittamisessa. (Lantz 2015, 229.)

3.3.1 Monikerrosperseptroni

Monikerrosperseptronin kaikki kerrokset ovat täysin kytkettyjä eli niiden kaikkien kerrosten jokainen solmu on kytkettynä kaikkiin edeltävän kerroksen solmuihin (kuva 5). Kuvan d-kerroksessa on parametreja eli painotettuja kytköksiä 4 x 4 +

4. Vakiotermikytkökset lisäävät parametrien määrää 4:llä. Monikerrosperseptroni vaatii paljon laskutoimituksia, ja kuvantunnistukseen suositellaankin käytettäväksi harvemmin kytkettyjä konvoluutioneuroverkkoja osittain tästä syystä.



Kuva 5. Täysin kytketty d-kerros (Kuva: Jani Koski).

Tällaisillakin perusrakenteilla kuitenkin päästään jo melko hyviin tuloksiin ainakin sellaisessa kuvantunnistustehtävässä, jossa rajojen tunnistaminen on keskeistä. Luvussa 6 esitelty monikerrosperseptroni sai hyvän kokonaistarkkuuden yksinkertaisessa kuvantunnistustehtävässä. Pullonkaulaksi tällaisissa ratkaisuissa muodostuu ainakin laskutoimitusten määrä isommissa verkotuksissa.

3.3.2 Konvoluutioneuroverkko

Konvoluutioneuroverkoilla on saatu erittäin hyviä tuloksia kuvantunnistuksessa, johtuen niiden kyvystä irrottaa erilaisia piirteitä kuvadatasta. Ne ovatkin yleisesti käytettyjä ratkaisuja kuvaluokitteluun. Kuuluisia ja menestyksekkäitä konvoluutioverkotuksia ovat esimerkiksi GoogleNet ja LeNet-5, jotka kummatkin ovat voittaneet vuosittaisen ImageNet -yhteisön järjestämän ImageNet Large Scale Visual Recognition Challenge -kuvantunnistuskilpailun.

Suurin osa moderneista kuvantunnistukseen käytettävistä konvoluutioneuroverkoista hyödyntävät seuraavanlaista perusrakennetta: Konvoluutiokerrokset vuorottelevat alinäytteistyskerrosten kanssa *n* kertaa, ja verkoston lopuksi on pieni määrä täysin kytkettyjä piilokerroksia.

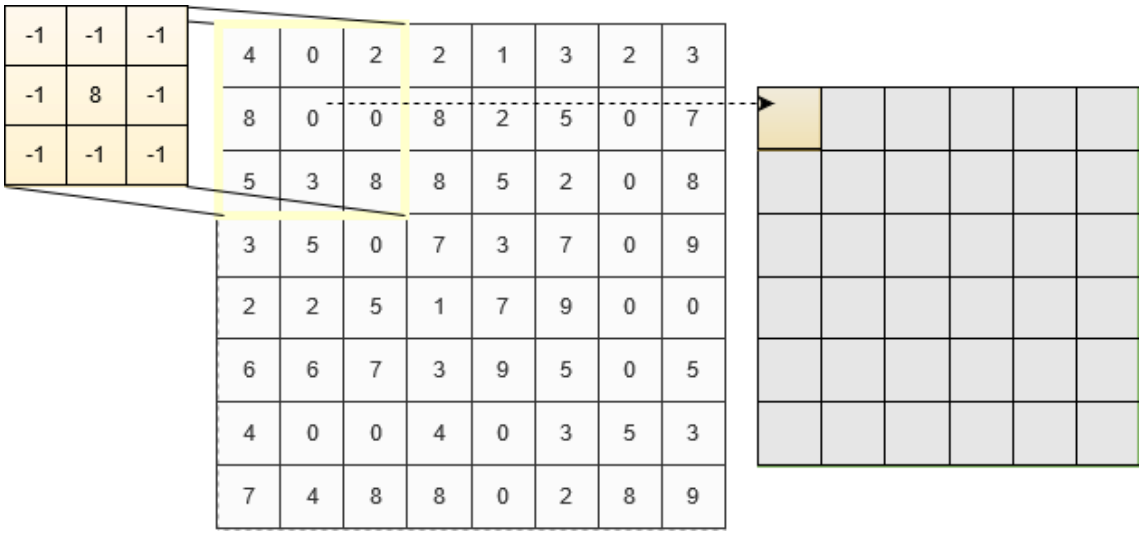
Konvoluutioverkot ovat harvemmin kytkettyjä kuin perinteisemmät monikerrosperseptronit ja siksi periaatteessa nopeampia, koska solmujen välisiä laskutoimituksia on vähemmän (Goodfellow, Bengio & Courville 2016, 335). Toisaalta konvoluutio-operaatioiden tuloksena syntyvät piirrekartat lisäävät kuvadataa, joten laskutoimitukset tältä osin lisääntyvät myöhemmissä kerroksissa.

Konvoluution ideana on irrottaa datasta piirteitä, kuten reunoja, konvoluutiokernelin avulla. Konvoluutiokerneli on *n* x *n* pikselin matriisi, joka liikkuu kuvadatan päällä määritellyin askelluksin, kertoo kuvadatan pikseliarvot alkioissaan olevilla luvuilla, ja luo niistä piirrekartan. Tämä prosessi toistetaan erilaisilla konvoluutiokerneleillä, kunnes haluttu määrä piirrekarttoja on luotu.

Kuvassa 6 pyrin selventämään konvoluutiokernelin toimintaa. 3x3[1] pikselin kokoinen vastaanottava kenttä (engl. *receptive field*) eli kernelimatriisi on kuvadatan päällä, ja sen alkioiden arvot kerrotaan kuvadatan vastaavassa sijainnissa olevan pikseliarvon kanssa. Suodattimen keskusalkio osoittaa piirrekartassa sijainnin, johon matriisin alueelta saadut tulot summataan. Keskusalkion ulostulo on siis piirrekartan yhden pikselin arvo.

Kuvan 6 tapauksessa suodatin ei ylitä kuvadataa, joten reunimmaisia pikseleitä ei huomioida piirrekartassa, ja siitä tulee pienempi kuin alkuperäinen kuvadata. Jos reunimmaiset pikselit halutaan mukaan, ratkaisuna on ympäröidä kuva täytearvoilla, esimerkiksi nollilla (padding-parametri). Käytännössä lienee usein käytännöllisintä käyttää täytearvoja konvoluutiokerroksen asetuksissa. Stride-parametrin arvon eli askelluksen koon olisi hyvä mielestäni olla kokoa 1 leveys- ja korkeusdimensioissa pienissä kuvissa, kuten myöhemmin käyttämissäni 28 x 28 pikselin kuvissa, jotta piirrekartta säilyisi alkuperäisen kokoisena ja jokaisesta pikselistä saataisiin kaikki piirteet talteen.

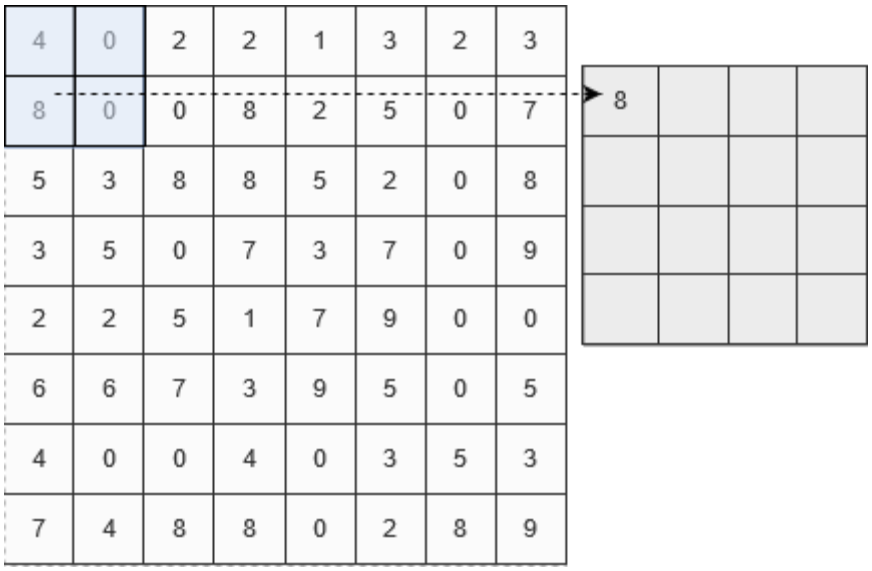
[1] 3x3 on pienin mahdollinen kerneli, missä on keskusalkio. 1x1 kerneli voidaan nähdä ennemminkin lineaarikuvauksena (Simoyan, Zisserman 2015, 2).



Kuva 6. 3x3 Konvoluutiokerneli, 8x8 kuvadata ja 6x6 piirrekartta (Kuva: Jani Koski).

Piirrekarttojen luominen lisää kaksiulotteiseen kuvadataan syvyysdimension, ja näin ollen seuraaville kerroksille jää yleensä enemmän laskettavaa. Tätä laskentaa helpottaa konvoluutioneuroverkossa yleisesti käytettävä alinäytteistys, joka luo tiivistelmän pikselidatasta poistamalla variaatiota leveys- ja korkeusdimensioissa, ehkäisten näin myös ylisovittamista. Yleisimmin käytössä on niin sanottu max pool -alinäytteistys, joka toimii seuraavanlaisesti: max pool -alinäytteistyssuodatin askeltaa kuvadatan läpi ja valitsee datasta korkeimman arvon suodattimen sisältä. Käytettäessä esimerkiksi 2x2 pikselin kokoista alinäytteistyssuodatinta saadaan 75 % pienempi kuvadata leveys- ja korkeusdimensioissa syvyyden eli piirrekarttojen lukumäärän säilyessä ennallaan (kuva 7).

Eri lähteet määrittelevät alinäytteistyksen hieman eri tavalla. Alinäytteistys voidaan määritellä omaksi neuroverkkokerrokseksi tai sen voidaan toisaalta ymmärtää kuuluvan konvoluutiokerrokseen. Koska alinäytteistyskerroksella ei ole omia parametreja, niin usein se määritellään kuuluvaksi konvoluutiokerrokseen. (Deeplearning.ai 2020.)



Kuva 7.Max pool -alinäytteistys, ja tuloksena syntyvä kuvadata kun Stride-parametri on 2 leveys- ja korkeusdimensioissa (Kuva: Jani Koski).

Verkoston lopussa olevat kerrokset ovat tiheitä eli täysin kytkettyjä. Ne tasoittavat (engl. *flatten*) konvoluutiossa syntyneen kolmiulotteisen datan yksiulotteiseksi ulostulokerroksen käyttöön.

3.4 Neuroverkon koulutus ja optimointi

Monikerroksiset neuroverkot ovat mustalaatikkoalgoritmejä, eli niiden koulutuksen jälkeiset sisäiset logiikat eivät ole suoraan ymmärrettävissä ulkopuoliselle tarkastelijalle. Koulutusvaiheessa verkosto ohjautuu sopeuttamaan toimintaansa vastaamaan koulutusdatassa annettuja syötteitä ja luokituksia. Toisin sanoen verkosto oppii approksimoimaan epälineaarista funktiota painokertoimien säätelymekanismin avulla.

Vastavirtafunktio (engl. *backpropagation*) on yleisesti käytetty algoritmi verkon virheiden paikallistamiseen. Gradienttimenetelmä on toinen osa koulutusprosessia. Niiden perusidea on seuraava: Koulutustapahtuman aikana koulutettavan neuroverkon antamaa luokitusta verrataan datanäytteeseen sisällytettyyn oikeaan luokitukseen. Tämän jälkeen vastavirtafunktio etenee verkoston lopusta takaisin alkuun päin etsien virhetulokseen eniten vaikuttavia solmuja. Näiden sol-

mujen painoarvoja muokataan gradienttimenetelmän avulla, jotta verkoston tuottamaa virhemarginaalia saadaan minimoitua (LeCun, Bengio & Hinton 2015, 1–2).

Koulutusdatan kaikkien datanäytteiden yhtä kokonaista läpikäyntiä kutsutaan epookiksi. Gradienttimenetelmä tarvitsee optimointiin useamman epookin. Koulutustapahtumassa optimaalinen epookkien määrä vaihtelee. AMLS:n moniluokittelijan koulutustapahtuman perusasetus on 100 epookkia.

3.5 Datan valmistelu

Koulutukseen tarvitaan määrältään ja variaatioltaan riittävä määrä kuvadatanäytteitä, jotta on mahdollista saavuttaa mallin laadukas yleistämisen aste. Yksittäisten datanäytteiden tulee sisältää kaikki koulutuksessa käytettävät piirteet eli pikseliarvot. Mikäli data ei ole variaatioltaan laajaa, algoritmi oppii ali- tai ylisovittamaan eli se ei tunnista koulutuksen jälkeen tarjottavista uusista datanäytteistä luokan erilaisia variaatioita. Olemassa olevaa kuvadataa voidaan käyttää uudelleen esimerkiksi kiertämällä kuvadataa, jotta malli oppii tunnistamaan objekteja myös eri asennoissa.

Data normalisoidaan yleensä välille 0–1, koska aktivointifunktioiden sisääntulojen arvorajat, jotka vaikuttavat ulostulevaan signaaliin ovat suhteellisen kapeita. AMLS:ssa normalisoinnin voi asettaa osaksi moniluokittelijan toiminnallisuutta.

Koulutetut mallit tarvitsevat lisäksi testidataa mallin tarkkuuden analysointiin. Koulutus- ja testidatan välinen suhde vaihtelee, mutta usein testidataa on noin 15 % – 30 % kokonaisdatamäärästä.

Datan variaation arvioimiseen ja ylisovittamisen ehkäisemiseksi voidaan apuna käyttää ristiinvalidointia. Ristiinvalidointi kouluttaa ja testaa mallin vuoroin eri osilla datasta sekä tuottaa metriikat mallien tarkkuuksien vertailemiseen. Jos eri dataosioista rakennettujen mallien keskinäinen variaatio on suhteellisen suurta,

voidaan epäillä datan tasalaatuisuutta. Koska ristiinvalidointi kouluttaa mallin useita kertoja, sen haittapuoli on ajankäytöllinen. (Microsoft 2019a.)

4 Azure Machine Learning Studio

Microsoftin AMLS on Platform as a Service -tyyppinen koneoppimisen työympäristö. Työympäristö sisältää graafisen käyttöliittymän, näytteistysdatasettejä, muokattavien algoritmien kirjaston sekä analyysityökaluja. Lisäksi käytössä ovat palvelimet algoritmien koulutukseen tarvittavaan laskentaan sekä palvelut ja rajapinnat valmiiden web-palvelujen julkaisemiseen, käyttöön ja hallintaan. (Microsoft 2019b.)

Ilmaisversiossa moduulien käytön yläraja eksperimenttikohtaisesti on 100 ja tallennustilan yläraja on 10 GB. Web-palveluiden maksimimäärä on 2. (Microsoft 2020.) Koulutustapahtuman maksimiajoaika on Microsoftin mukaan tunti, mutta olen ajanut koulutustapahtumia 24 tuntia 5 minuuttia ennen suorituksen aikakatkaisua, mikä on sama aika kuin moduulikohtainen maksimiajoaika maksullisessa standard-versiossa.

Koneoppimisen päätyypeistä lähinnä ohjatun oppimisen algoritmit ovat toteutettuna palveluun. Poikkeuksena on ohjaamattomaan oppimisen K-Means Clustering -algoritmi, jota voi käyttää muun muassa kuvansegmentoinnissa, mutta jolla kuvantunnistukseen ei ole ainakaan ilmeistä käyttöä.

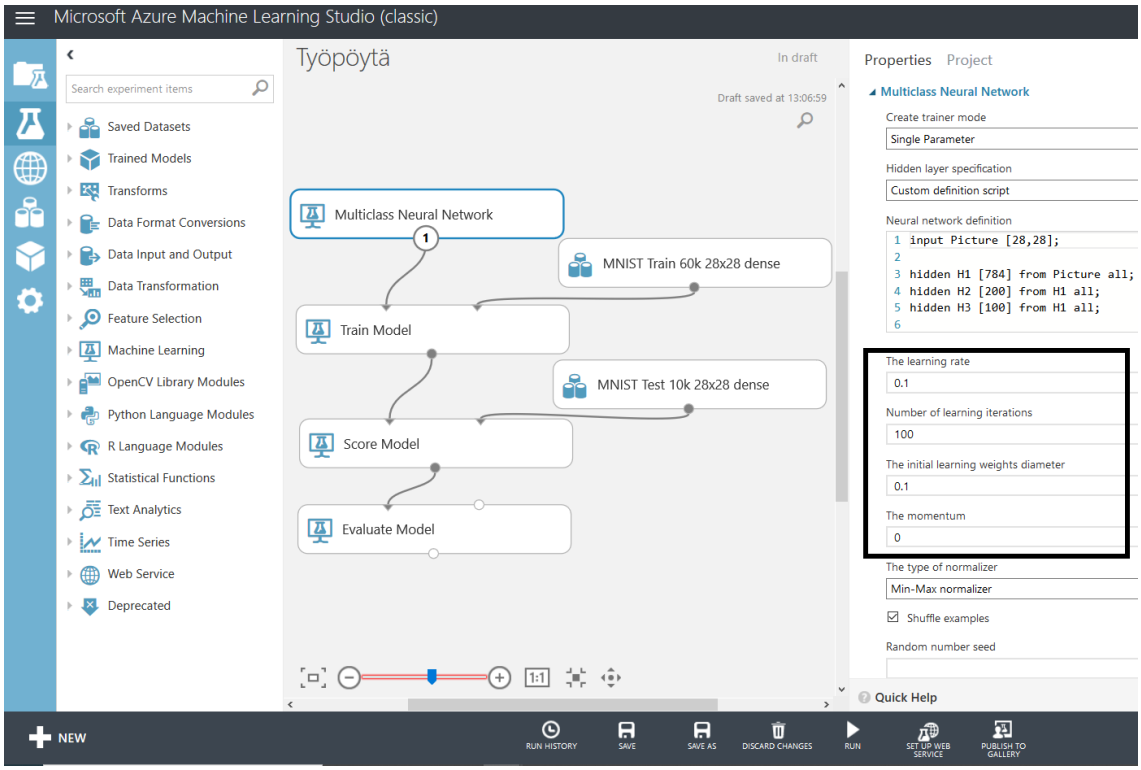
Neuroverkkoja hyödyntäviä moduuleita AMLS:ssa on kolme: Kahden luokan luokittelu, moniluokittelu ja regressio. Moniluokittelun moduulia käytetään kuvantunnistusprojektien rakentamiseen luvussa 5.

4.1 Työympäristö ja moduulit

Koneoppimisprojektin rakentaminen tapahtuu graafisella alustalla. Koneoppimisprojektissa tarvittavat toiminnallisuudet ovat jaoteltuna moduuleihin, jotka näyttäytyvät graafisina toisiinsa kytkettävinä palkkeina

Kuvassa 8 projektin työpöydälle on koottu moniluokittelevan neuroverkon tarvitsemat tärkeimmät perusmoduulit: Koulutus- ja testidata, Multiclass Neural Network, Train Model, Score Model ja Evaluate Model -moduulit. Oikealla ovat neuroverkkomoduulin asetukset, jossa kaikki verkon määrittelyt tehdään. Verkon rakenteen määrittelyskriptin lisäksi olennaisia ovat kuvassa suorakaiteen sisällä olevat optimointiasetukset. Näiden alla ovat datan normalisoijan asetukset.

Tällaisessa käyttötapauksessa asetuksia määritellään ainoastaan Multiclass Neural Network sekä Train Model -moduuleissa. Esimerkiksi datan muokkaamiseen tarvittavat moduulit eivät kuvan käyttötapauksessa ole tarpeen.



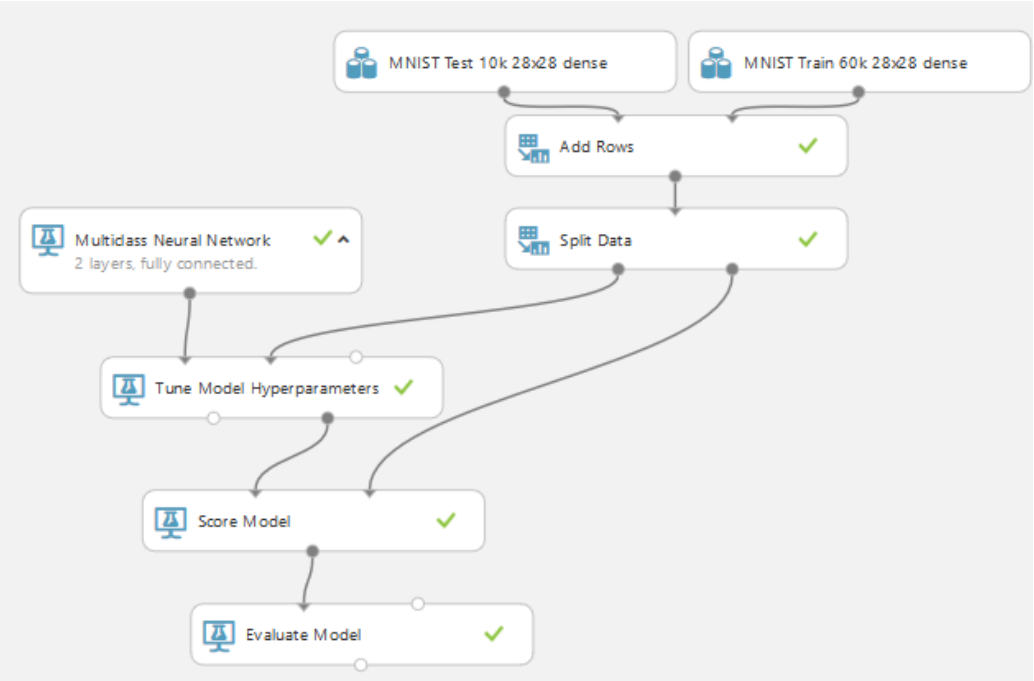
Kuva 8. Työympäristö ja tärkeimmät moduulit (Kuva: Jani Koski).

Neuroverkon määrittely tapahtuu Net# -kielellä ja sen avulla voidaan määrittää:

- Kerrosten tyypit ja niiden parametrit
- Kerrosten solmujen lukumäärät eri dimensioissa
- kerrosten väliset kytkennät
- aktivointifunktiot. (Microsoft 2018.)

Perusasetuksilla, ilman määrittelyskriptin luomista, Multiclass Neural Network -moduulin verkkotopologia on täysin kytketty eli monikerrosperseptronin kaltainen yhden piilokerroksen kytkentämalli. Aktivointifunktion oletusasetuksena on sigmoid (Microsoft 2018).

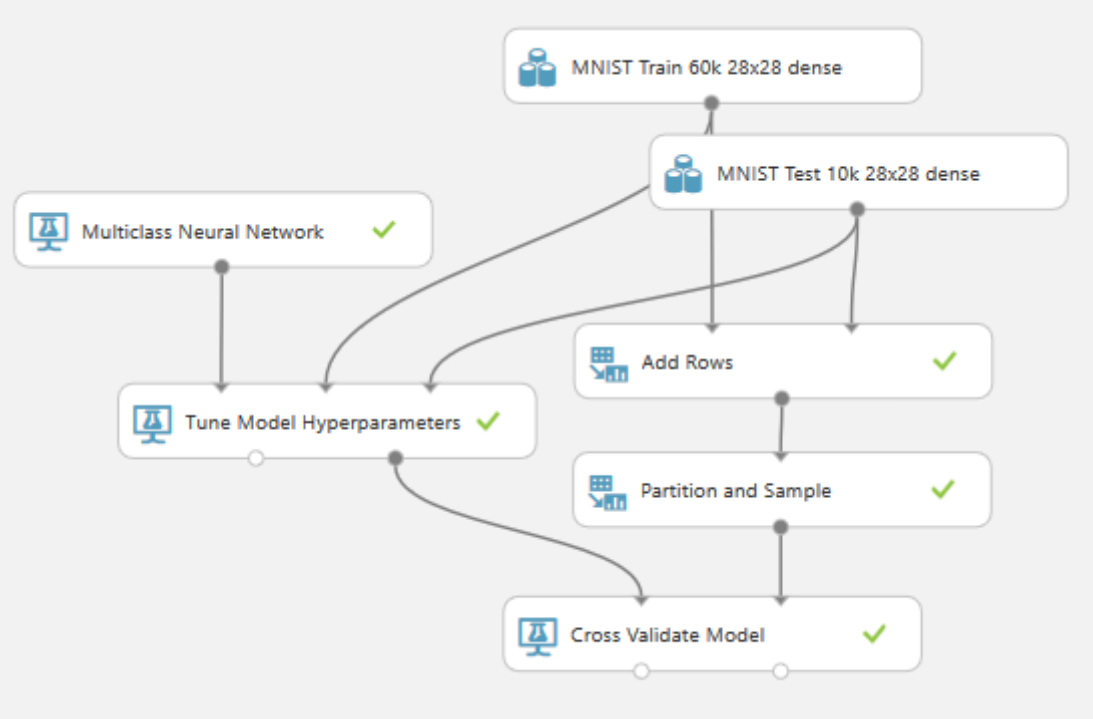
Kuvan 9 asetelmassa kaksi datasettiä on yhdistetty ja tehdään erilainen koulutus- ja testidatan välinen jakauma. Train Model -moduulin sijaan käytetään Tune Model Hyperparameters -moduulia, joka automatisoi optimointiasetusten etsimisen. Se on ajankäytöllisesti raskaampi Train Model -koulutukseen verrattuna. Kun haluttu verkotus on löytynyt, kannattaa tarvittaessa lopullinen koulutus viedä läpi tällä moduulilla Train Model -moduulin sijaan.



Kuva 9. Tune Model Hyperparameters, Add Rows ja Split Data -moduulit (Kuva: Jani Koski).

Kuvassa 9 testidataa ei ole kytketty Tune Model Hyperparameters -moduulin oikealla olevaan valinnaiseen kytkentäpaikkaan. Koulutusaikojen tilastot, jotka esitetään luvuissa 5.1.1 ja 5.1.2, viittaavat koulutuksen keston huomattavaan pitkittymiseen, kun testidatan pisteytys tehdään vasta myöhemmin.

Ristiinvalidointi jakaa perusasetuksella datan 10 osaan. Käyttämällä Partition and Sample -moduulia voidaan ristiinvalidoinnin tekemiin datajakaumiin tehdä muutoksia ja säästää näin tarvittaessa aikaa. Kuvassa 10 malli koulutetaan ja sitä käytetään ristiinvalidoinnissa luomaan metriikat datan tasalaatuisuuden arviointiin. Ristiinvalidoinnin tuloksissa on dataosiokohtaiset tarkkuuksien keskihajonnat. Niiden erot tulisivat olla mahdollisimman pieniä eri osioiden välillä.



Kuva 10. Ristiinvalidointi (Kuva: Jani Koski).

4.2 Web-palvelut

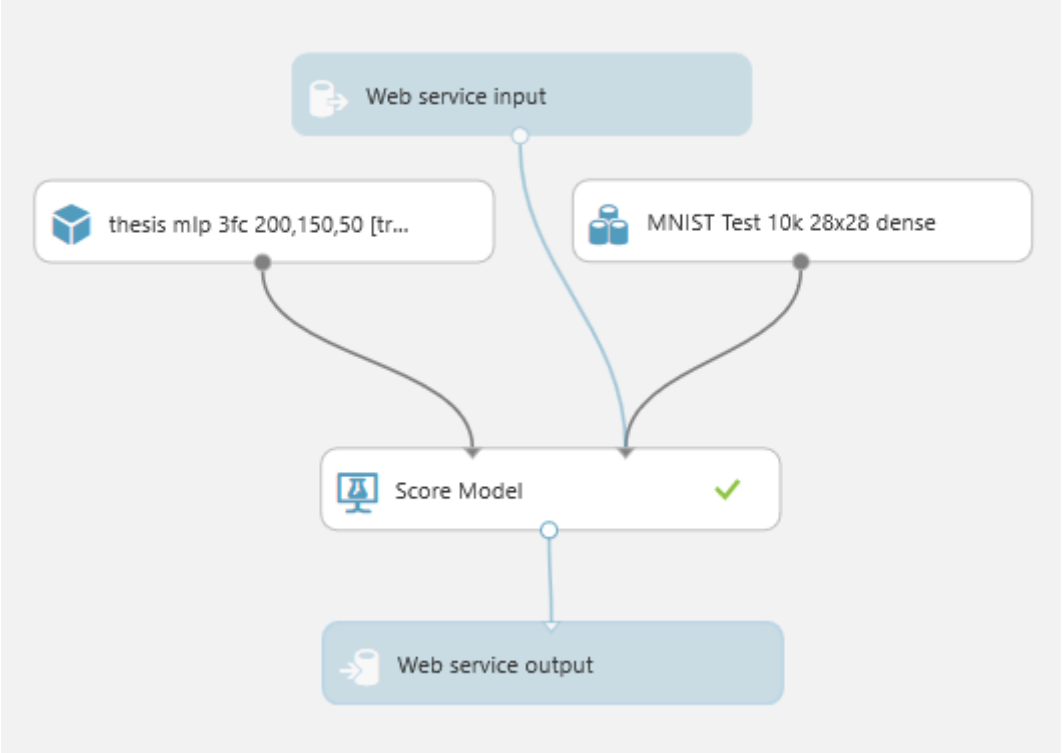
AMLS:n web-palvelut tarjoavat rajapinnan, jolla valmiita malleja voidaan hyödyntää sovelluksissa. Malli on vain julkaistava web-palveluna, jonka jälkeen palvelun REST-rajapinta on käytettävissä. Web-palvelulle luodaan julkaisemisen jälkeen

sivut, joiden kautta palvelua voi tarkastella, hallinnoida, päivittää ja testata. Sivuilta löytyvät palvelun käyttöön tarvittavat rajapinnan URI, API-avain sekä koodiesimerkkejä rajapinnan kutsusta (C#, Python, Python 3+, R). (Microsoft 2017a.)

Koko web-palvelun luomisprosessi on täysin automatisoitu, mutta muokattavissa samaan modulaariseen tapaan kuin luvussa 4.1 esitellyt projektit. Lisäksi käytössä ovat ainakin Web service input ja Web service output -moduulit.

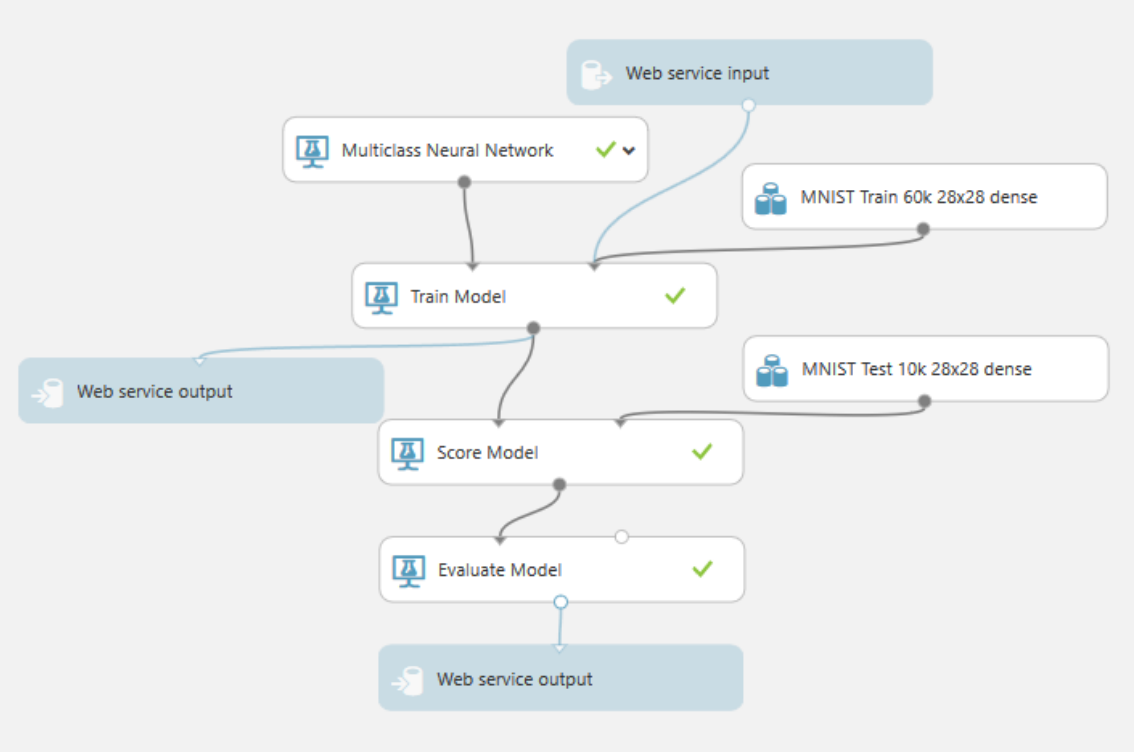
Palvelua voidaan hyödyntää pyyntö/vastaus -tyyppisesti, jolloin sille lähetetään oikeanmuotoinen kuvadatanäyte, josta palautuu mallin antama vastaus. Erätyyppisessä kyselyssä voidaan lähettää useampi kuvadatanäyte kerralla. Web-palvelun mallia on myös mahdollista kouluttaa uudelleen luomalla uudelleenkoulutukseen soveltuva erillinen web-palvelu. (Microsoft. 2017a.)

Web-palvelun luonti lisää projektiin input ja output -moduulit ja saattaa muuttaa hieman projektin rakennetta, kuten asettaa Apply Transformation -moduulin, jos datamuokkauksia on tehtävä syötteelle ennen pisteytystä (kuva 11).



Kuva 11. Web palvelun moduulit (Kuva: Jani Koski).

Mallia uudelleenkouluttavaan web-palveluun pitää lisätä Web service output -ulostulo Train Model -moduuliin. Kuvan 12 tapauksessa käytetään web-palveluun syötetyn datan lisäksi jo olemassa olevaa koulutusdataa. Palvelu palauttaa uuden mallin sekä mallin tarkkuuden metriikat.



Kuva 12. Uudelleenkouluttavan web-palvelun rakenne (Kuva: Jani Koski).

5 Toteutus

Opinnäytetyön toteutuksen osiossa luotiin rakennetuista neuroverkkokonfiguraatioista web-palveluita, jotka otettiin käyttöön Node.js:llä toteutetussa web-sovelluksessa. Koulutus- ja testidatana käytettiin MNIST-datasettiä, joka koostuu 60 000 koulutus- ja 10 000 testikuvasta. 28 x 28 pikselin mustavalkokuvat sisältävät käsin piirrettyjä numeroita väliltä 0–9 (LeCun, Cortes & Burges 2020). Tämä on tunnettu neuroverkkojen koulutukseen käytetty datasetti, ja valmiiksi sisällytetty AMLS:on.

5.1 Web palvelut

Kaksi erilaisilla neuroverkkorakenteilla toimivaa web-palvelua toteutettiin. Seuraavissa alaluvuissa kuvataan niiden rakenteet ja tulokset.

5.1.1 Monikerrosperseptroni

Kuvassa 13 on toteutetun monikerrosperseptronin määrittely. Rakenteessa on 4 kerrosta: syötekerros (Picture), kolme piilokerrosta (H1, H2, H3) ja ulostulokerros (Result). Optimoinnin jälkeen rakenteen testivaiheen pisteytyksen antamaksi kokonaistarkkuudeksi muodostui 98.25 %

```
input Picture [28,28];

hidden H1 [200] from Picture all;
hidden H2 [150] from H1 all;
hidden H3 [50] from H2 all;

output Result [10] softmax from H3 all;
```

Kuva 13. Monikerrosperseptronin määrittely Net#-kielellä (Kuva: Jani Koski).

Täysin kytketty kerros määritellään avainsanalla *all*. Aktivointifunktiona on piilokerroksissa oletusfunktio sigmoid ja ulostulokerroksessa softmax. Parametrejä eli painotettuja kytköksiä verkossa on 191 210 (taulukko 1).

Taulukko 1. Monikerrosperseptronin parametrit

Kerros	Dimensiot	Koko	Parametrit
input	28, 28	764	0
h1	200	200	153 000
h2	150	150	30 150
h3	50	50	7 550
output	10	10	510
Verkon parametrit			**191 210**

Taulukossa 2 on esitelty monikerrosperseptronin eri koulutustapahtumien ajallista kestoa ja testidatan pisteytyksen tuottamia kokonaistarkkuuksia kun verkon

automaattista optimointia ei käytetä, tai sitä käytetään eri asetuksilla. Iteraatiot-sarake ilmaisee koulutustapahtumaan käytettyjen epookkien määrän. Punaisella merkittyihin koulutusaikoihin vaikuttaa todennäköisesti se, ettei testidata ole ollut näissä tapahtumissa optimointimoduuliin kytkettynä, vaan testidatan pisteytys on suoritettu erikseen myöhemmässä Score Model -moduulissa.

Taulukko 2. Monikerrosperseptronin koulutustapahtumat

!td = testidata ei ole kytketty suoraan TMH -moduuliin				
OA = Kokonaistarkkuus		TMH = Tune Model Hyperparameters -asetukset		
Koulutustapahtuma	**Iteraatiot**	**KoulutusAika**	**OA**	**TMH**
1	100	0.08.24.638	98.24%	ei käytössä
2	450	0.13.23.213	**98,25 %**	random sweep, 5
3	1800	0.25.23.491	98,07 %	entire grid
4	450	2.09.41.581	**98.25%**	random sweep, 5, **!td**
5	1800	3.35.42.680	98.02%	entire grid, **!td**

5.1.2 Konvoluutioneuroverkko

Kuvassa 14 on toteutetun konvoluutioverkon määrittely. Verkko sisältää syöte- ja ulostulokerroksen lisäksi kaksi konvoluutiokerrosta (C1, C2) ja kaksi alinäytteistyskerrosta (P1, P2), joiden jälkeen on yksi täysin kytketty yksiulotteinen kerros (H3). Optimoinnin jälkeen rakenteen testivaiheen pisteytyksen antamaksi kokonaistarkkuudeksi muodostui 98.96 %.

```
const { T = true; F = false; }

input Picture [28, 28];

hidden C1 [5, 28, 28]
    from Picture convolve {
        InputShape      = [28, 28];
        KernelShape     = [ 3, 3];
        Stride          = [ 1, 1];
        Padding         = [T, T];
        MapCount        = 5;
    }

hidden P1 [5, 14, 14]
    from C1 max pool {
        InputShape      = [5, 28,
28];
        KernelShape     = [1, 2, 2];
        Stride          = [1, 2, 2];
    }

hidden C2 [35, 14, 14]
    from P1 convolve {
        InputShape  = [5, 14, 14];
        KernelShape = [1, 3, 3];
        Stride      = [1, 1, 1];
        Padding     = [F, T, T];
        MapCount    = 7;
    }

hidden P2 [35, 7, 7]
    from C2 max pool {
        InputShape  = [35, 14, 14];
        KernelShape = [1, 2, 2];
        Stride      = [1, 2, 2];
    }

hidden H3 [100]
    from P2 all;

output Result [10] softmax
    from H3 all;
```

Kuva 14. Konvoluutioneuroverkon määrittely Net#-kielellä (Kuva: Jani Koski).

Konvoluutiokerros määritellään avainsanalla *convolve* ja se tarvitsee vähintään *InputShape*, *KernelShape* -parametrit. Konvoluutio- ja alinäytteistyskerroksen dimensioihin vaikuttavat kerroksen omat parametrit (kernelin muoto, askelkoko, täytearvojen käyttö, painojen jakaminen) sekä edellisen kerroksen dimensiot. Liitteessä 1 määrittelystä on laajennettu versio, josta konvoluutio- ja alinäytteistyskerrosten dimensioiden laskutapa käy paremmin ilmi.

Konvoluutioneuroverkon parametrit ilmenevät taulukosta 3. Siinä on noin 9 % vähemmän parametreja kuin toteutetussa monikerrosperseptronissa. Konvoluutiokerrosten parametrien vähyys tulee taulukossa selkeästi esille. Suurin osa painotetuista kytkennöistä on täysin kytketyssä kerroksessa FC3. Tämä konvoluutioneuroverkko on laskennallisesti kevyempi käyttää ja tarkempi kuin luvussa 5.1.1 esitelty monikerrosperseptroni.

Taulukko 3. Konvoluutioneuroverkon parametrit

Kerros	Kerneli	Askellus	Dimensiot	Koko	Parametrit
input			28, 28	764	0
C1	3	1	5,28,28	3 920	50
P1	2	2	5,14,14	980	0
C2	3	1	35,14,14	6 860	1 610
P2	2	2	35,7,7	1 715	0
FC3			100	100	171 600
output			10	10	1 010
Verkon parametrit					**174 270**

Taulukossa 4 on esitelty konvoluutioneuroverkon eri koulutustapahtumien ajallista kestoa ja testidatan pisteytyksen tuottamia kokonaistarkkuuksia, kun verkon automaattista optimointia ei käytetä, tai sitä käytetään eri asetuksilla. Iteraatiot-sarake ilmaisee koulutustapahtumaan käytettyjen epookkien määrän.

Taulukko 4. Konvoluutioneuroverkon koulutustapahtumat

!td = testidata ei ole kytketty TMH -moduuliin				
TMH = Tune Model Hyperparameters			OA = Kokonaistarkkuus	
Koulutus	**Iteraatiot**	**KoulutusAika**	**OA**	**TMH**
1	100	0.51.51.106	98.86%	ei käytössä
2	450	12.38.27.572	98,85 %	random sweep, 5, **!td**
3	1 800	2.07.52.262	98,95 %	entire grid
4	450	1.17.13.789	98.83%	random sweep, 5
5	1 800	20.39.07.297	**98,96 %**	entire grid, **!td**
6	426	1.07.11.057	98,94 %	random sweep, 6
7	426	11.38.30.877	98,80 %	random sweep, 6 , **!td**

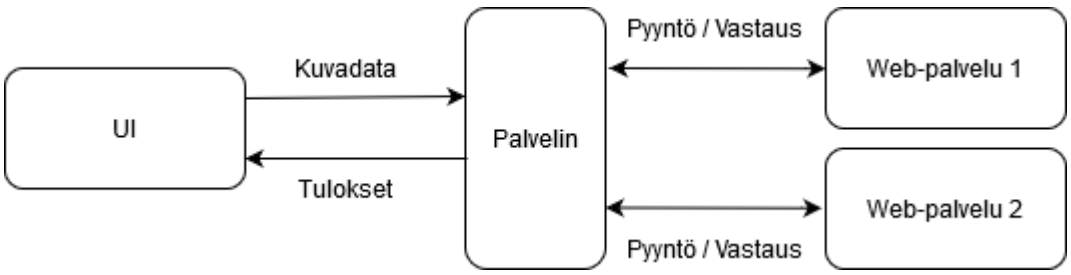
Konvoluutioverkon koulutus kestää kauemmin kuin monikerrosperseptronin koulutus. Punaisella merkittyihin koulutusaikoihin vaikuttaa todennäköisesti se, ettei

testidata ole ollut näissä tapahtumissa optimointimoduulin valinnaiseen kytkentäpaikkaan kytkettynä, vaan testidatan pisteytys on suoritettu erikseen myöhemmässä Score Model -moduulissa.

Hieman suurempiakin tarkkuuksia oli mahdollista saavuttaa, mutta verkon laskennallinen raskaus kasvoi huomattavasti pienen parannuksen myötä. Koska nämä ovat vielä suhteellisen keveitä ja yksinkertaisia rakenteita, päädyin ottamaan ne mukaan web-palveluiksi. Muita syitä olivat kuvantunnistustehtävän suhteellinen yksinkertaisuus ja ylisovittamisen välttäminen.

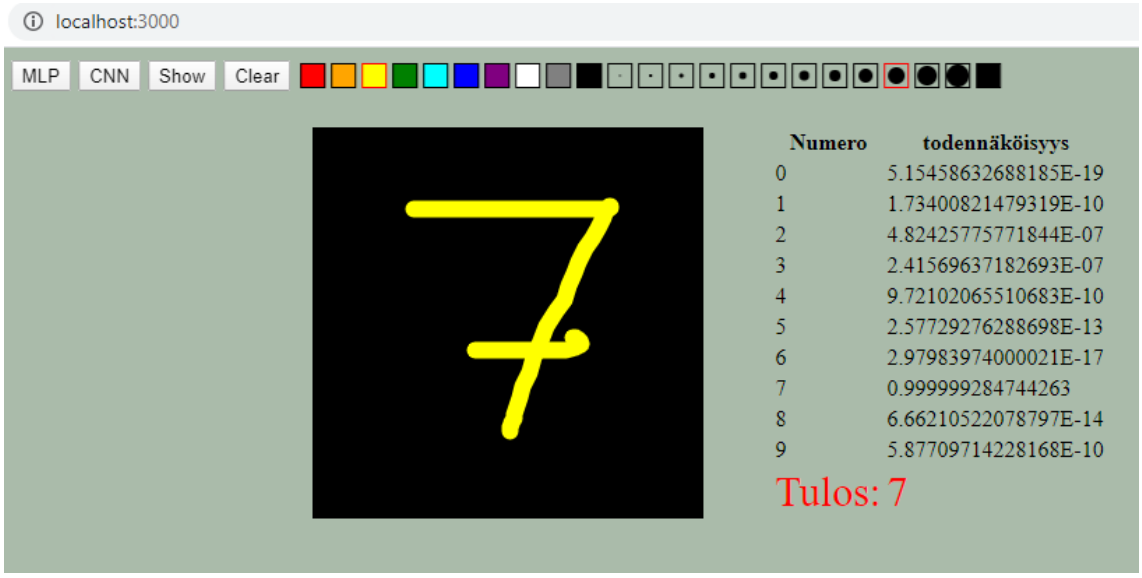
5.2 Web-sovellus

Sovellus käyttää express-kehikkoa hyödyntävää node.js-palvelinta. Staattisten tiedostojen lähettämisen lisäksi palvelin valmistelee ja lähettää valitulle web-palvelulle kyselyn sekä parsii palautuneesta vastauksesta web-palvelun kuvalle antamat luokitukset. Luokitukset palautetaan vastauksena selaimelle (kuva 15).



Kuva 15.Web-sovelluksen rakenne (Kuva: Jani Koski).

Web-sovelluksen käyttöliittymä on toteutettu HTML, CSS ja JavaScript -kielillä. Sovelluksessa käyttäjä piirtää hiirellä selaimen piirtonäkymään numeron, ja valitsee web-palvelun, jolle kuvadata lähetetään (kuva 16).



Kuva 16. Sovelluksen käyttöliittymä (Kuva: Jani Koski).

Piirros esiprosessoidaan selaimessa ensin mustavalkokuvaksi, sitten pienennetään 28x28 pikselin kuvaksi. Sovelluksen apuikkunassa prosessi on visuaalisesti nähtävissä (kuva 17). Prosessoinnista saatuun kuvadatataulukkoon lisätään myöhemmin kyselyssä tarvittava yksi alkio ja se lähetetään node.js-palvelimelle, joka valmistelee kyselyn ja lähettää sen valitulle AMLS:n web-palvelulle käsiteltäväksi.

Kuva 17. Web-sovelluksen käyttöliittymä ja apuikkuna (Kuva: Jani Koski).

Kuvassa 18 on web-palvelulle lähetettävän pyynnön rakenne. Pyyntöön asetettavan kuvadatan tulee olla 785-alkioinen kokonaislukutaulukko, joka sisältää 784 harmaasävyarvoa jokaiselle pikselille arvovälillä 0–255. Lisäksi taulukko sisältää

kokonaisluku tyyppisen alkion taulukon 1. indeksissä, jossa verkon uudelleen-koulutuksen tapauksessa olisi tieto kuvan luokasta.

```
let  options = {
  method: 'POST',
  url: webServiceUri,
  headers: {
    'Content-Type': 'application/json',
    Authorization: `Bearer ${apikey}`,
  },
  body: JSON.stringify({
    Inputs: {
      input1: {
        ColumnNames: [
          'Label',
          'f0',
          'f1',
          'f2', // 'f3'...'f782',
          'f783',
        ],
        /* Values-taulukkoon 785 alkion kuvadatataulukko */
        Values: [],
      },
    },
    GlobalParameters: {},
  }),
}
```

Kuva 18. Web-palvelulle lähetettävän JavaScript-pyynnön rakenne (Kuva: Jani Koski).

Kuvassa 19 on esimerkki JavaScript-pyynnön lähettämisestä. Käytössä tulee tässä tapauksessa olla NPM:n tarjoama Request-paketti (Microsoft 2017b).

```
const request = require('request');
//Kuvan 18 options-olio argumentiksi
request(options, (error, response) => {
  if (error) throw new Error(error);
  console.log(response.body);
});
```

Kuva 19. Esimerkki JavaScript-pyynnön tekemisestä (Kuva: Jani Koski).

Vastaus palautuu web-palvelulta palvelimelle, joka lähettää edelleen sovellukselle vastauksesta parsitut, kuvissa 16 ja 17 oikealla näkyvät arvot. Nämä arvot ovat softmax-aktivointifunktion antamat luokkakohtaiset liukulukuarvot lähetetylle kuvalle sekä eksplisiittisemmin ilmaistu tieto ennustetusta todennäköisimmästä luokasta.

5.3 Käytettävyys

AMLS:n graafinen käyttöliittymä esittää työympäristön ja moduulit erittäin selkeästi, ja moduulien peräkkäin kytkeminen on pitkälle intuitiivista. Alkuun pääsemiseksi alustalla on useita opettelemiseen tarkoitettuja projekteja ja tarjolla oleva dokumentaatio on melko kattavaa. Matalan tason tietämys algoritmeista tai ohjelmointitaito eivät ole välttämättömiä alkuun pääsemiseksi. Näistä syistä käytön aloittamisen kynnys on hyvin matalalla.

Tällainen helppokäyttöisyys saattaa tosin tulla asetusten määrän kustannuksella. Vaikuttaisi siltä, että esimerkiksi kaikkia optimointiasetuksia ei saisi käyttöön. Esimerkiksi gradienttimenetelmän tyypin valintaa en löytänyt. En osaa ottaa kantaa Python ja R-kielten mahdollisuuksiin laajentaa AMLS:n toiminnallisuutta tältä osin.

Laskentatehon määrittäminen on mahdotonta, mutta prosesseja voi ajaa rinnakkain ottamalla maksullisen standard-version käyttöön (Microsoft 2020). Ilmaisversiossa vain yhtä tapahtumaa on mahdollista ajaa kerrallaan, joten usean projektin samanaikaiset suoritukset eivät tältä osin onnistu. Suhteellisen lyhyen moduulin maksimiajoajan takia pidempien koulutustapahtumien onnistunut suorittaminen on epävarmaa. Kuten luvussa 4 todettiin, ilmaisversiossa moduulin maksimiajoaika on tunti, mutta tästä huolimatta olen pystynyt ajamaan 24 tuntia 5 minuuttia koulutustapahtumia ennen aikakatkaisua. Tunnin aikakatkaisua ei ole tullut vastaan myöskään ristiinvalidointia tai muita moduuleja ajaessa. 24 tunnin aikaraja ei ole ongelma luvun 5 kokoluokkaa olevissa projekteissa, kun automaattista optimointia käytetään nopeammalla tavalla eli niin, että testidata on suoraan kytket-

tynä automaattisen optimoinnin moduuliin. Toisaalta 24 tunnin aikaraja tulee luultavasti vastaan nopeasti, kun verkon parametrien lukumäärä kasvaa isompien kuvien tunnistamiseen luodussa syvemmässä neuroverkkorakenteessa.

Net#-kielen kirjoittaminen on hidasta ja työlästä AMLS:n tekstieditorissa verrattuna kehittyneempiin tekstieditoreihin. AMLS tarjoaa pelkistetyn, Notepad-sovelluksen kirjoitusnäkymää muistuttavan tekstieditorin Net#-kielen kirjoittamiseen. En löytänyt Net# lisäosia Microsoftin Visual Studio Code -editoriin, jotka voisivat helpottaa virheenkorjausta ja lisätä käyttömukavuutta koodia rakennettaessa.

AMLS:n projekteissa vastaan tulevien ongelmien ratkaisemiseen käytössä on keskustelufoorumi. Tässä työssä kohdattujen ongelmien osalta keskustelufoorumista ei ollut löydettävissä täsmällistä apua. Foorumi ei ole kovin aktiivinen ja monet siellä esitetyt kysymykset ovat jääneet vaille vastauksia tai niihin on vastattu vasta pitkän ajan kuluttua.

Omien kokemusteni perusteella arvioin AMLS:n käytettävyyden olevan hyvällä tasolla edellä mainittujen rajoitusten puitteissa. Työnkulut ovat suoraviivaisia ja työskentely moduulien parissa on sujuvaa ja miellyttävää. AMLS sopii hyvin ainakin pienempien neuroverkkorakenteiden ja pienempien kuvien kanssa työskentelyyn.

6 Pohdintaa

Pääsin suurimmaksi osaksi opinnäytetyölle asetettuihin tavoitteisiin. Tavoitteiden rajauksiin olisi pitänyt kiinnittää opinnäytetyön alkuvaiheessa enemmän huomiota. Nyt tavoitteet jäivät osin liian yleisluontoisiksi määritelmiksi, eikä tekeminen ollut niin hyvin kohdennettua kuin se olisi voinut olla.

Tietoperustan kokoaminen kesti odotettua kauemmin alalla käytettävän matemaattisen terminologian vuoksi. Tästä syystä teorian täydentäviä kuvauksia piti etsiä eri lähteistä ja toisaalta varmistaa tietojen oikeellisuutta. Siinä mielessä olin

onnekkaassa asemassa, että tietoa oli paljon saatavilla. Neuroverkkojen teoria oli minulle täysin uutta ja olen tekemääni teorian esitykseen melko tyytyväinen. Aihe on tietysti varsin laaja ja siitä olennaisen suodattaminen raporttiin selkeällä tavalla muodostui mielenkiintoiseksi tehtäväksi.

AMLS:n käytettävyyden arviointia en ehtinyt toteuttamaan sellaisella suunnitelmallisuudella kuin oli tarkoitus. Olin priorisoinut muut tavoitteet tätä tärkeämmiksi, ja niihin kului aikaa arvioitua enemmän.

Tarkoituksenani oli toteuttaa web-sovellukseen useampia erilaisen neuroverkkorakenteen sisältäviä web-palveluita sekä mallia uudelleenkouluttava web-palvelu, joka käyttäisi web-sovelluksessa piirrettyjä numeroita koulutusaineistona. Myöhemmin selvisi, että AMLS:n ilmaisversion web-palvelujen maksimimäärä on kaksi, joten se rajoitti tämän suunnitelman toteuttamista. Web-palvelut riittivät luvussa 5.1 esitettyjen palvelujen esittelemiseen.

Web-sovellus toimii kuten pitää, tosin kaikki numerot eivät tunnistu yhtä helposti kuin toiset. Erityisesti numerot 6, 7 ja 9 eivät käytännössä tunnistu kokonaistarkkuuden ilmaisemalla säännöllisyydellä. Mahdollinen syy voi olla ylisovittaminen. Web-sovellus tunnistaa numerot vain oikein päin piirrettyinä, koska koulutusdatassa numerot ovat olleet oikein päin. Jatkokehityksessä tämä tulisi ottaa huomioon lisäämällä kuvia eri kiertokulmissa, jotta neuroverkkorakenteet tunnistaisivat eri asennoissa olevia objekteja.

Parasta opinnäytetyön toteutuksessa teorian oppimisen lisäksi oli web-sovelluksen ja web-palveluiden rakentaminen. Toteutin web-sovellukseen tavoitteiden mukaiset toiminnallisuudet ja voin soveltaa opinnäytetyössä oppimiani teoreettisia tietoja sekä käytännön taitoja. Tämä avaa uusia näköaloja tulevien projektien mahdollisiin sisältöihin.

Jatkokehitysaihe voisi olla isompien kuvien käytettävyys AMLS:ssa huomioiden moduulin suorituksen aikarajan. Neuroverkon ensimmäisen piilokerroksen para-

metrien määrä riippuu syötekerroksen eli kuvadatanäytteen koosta. Tätä vaikutusta neuroverkkojen koulutusaikaan ei tullut tämän opinnäytetyön puitteissa tutkittua.

7 Lähteet

Deeplearning.ai. 2020. CNN Example. https://www.coursera.org/lecture/convolutional-neural-networks/cnn-example-uRYL1. 18.5.2020.

Goodfellow, I., Bengio, Y. & Courville, A. 2016. Deep Learning. Cambridge: MIT Press.

Hagan, M.T., Demuth, H.B., Beale, M.H. & De Jesús, O. 2014. Neural Network Design 2nd Edition. Oklahoma: Martin Hagan.

Lantz, B. 2015. Machine Learning with R. Birmingham: Packt Publishing Ltd.

LeCun, Y., Bengio, Y. & Hinton, G. 2015. Deep Learning. Nature. https://s3.us-east-2.amazonaws.com/hkg-website-assets/static/pages/files/DeepLearning.pdf. 1.4.2020.

LeCun, Y., Cortes, C. & Burges, C. 2020. THE MNIST DATABASE of handwritten digits. http://yann.lecun.com/exdb/mnist/. 15.4.2020

Microsoft. 2017a. Deploy an Azure Machine Learning Studio (classic) web service. https://docs.microsoft.com/en-us/azure/machine-learning/studio/deploy-a-machine-learning-web-service. 12.4.2020.

Microsoft. 2017b. How to consume an Azure Machine Learning Studio (classic) web service. https://docs.microsoft.com/en-us/azure/machine-learning/studio/consume-web-services. 8.4.2020.

Microsoft. 2018. Guide to Net# neural network specification language for Azure Machine Learning Studio. https://docs.microsoft.com/en-us/azure/machine-learning/studio/azure-ml-netsharp-reference-guide. 25.3.2020.

Microsoft. 2019a. Cross-Validate Model. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/cross-validate-model. 10.5.2020.

Microsoft. 2019b. What is Azure Machine Learning Studio? https://docs.microsoft.com/en-us/azure/machine-learning/studio/what-is-ml-studio. 17.4.2020.

Microsoft. 2020. Machine Learning Studio Pricing. https://azure.microsoft.com/en-us/pricing/details/machine-learning-studio/. 30.3.2020.

Mohammed, M., Khan, M. & Bashier, E. Machine Learning: Algorithms and Applications. https://learning.oreilly.com/library/view/machine-learning/9781315354415/. Boca Raton: CRC Press.

Mueller, P., Massaron, L. 2016. Machine Learning for Dummies. New Jersey: John Wiley & Sons.

Segaran, T. 2007. Programming Collective Intelligence. Sebastopol: O'Reilly Media.

Seinäjoen ammattikorkeakoulu. 2020. Automation in Network. https://www.seamk.fi/yrityksille/tki-projektit/automation-in-network/. 24.4.2020.

Simoyan, K., Zisserman, A. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. Oxford University Press. https://arxiv.org/pdf/1409.1556.pdf 6.4.2020.

Turing, A.M. 1950. Computing Machinery and Intelligence. Oxford University Press. https://doi.org/10.1093/mind/LIX.236.433. 11.10.2019.

Konvoluutioverkon määrittelyskripti (Net#)

```
/* Mukaeltu, alkuperäinen versio: https://gal-
lery.azure.ai/Experiment/Neural-Network-Convolution-
and-pooling-deep-net-2 */

// Muuttujien määrittely

const { T = true; F = false; }

const {
// Kuvan koko
ImgW = 28;
ImgH = 28;

// 1. konvoluutiokerros, parametrit...
C1Maps = 5; // Myös 1. kerroksen syvyysdimensio
C1KernW = 3;
C1KernH = 3;
C1StrideW = 1;
C1StrideH = 1;
// ...ja dimensiot kun padding on true.
C1OutW = (ImgW - 1) / C1StrideW + 1;
C1OutH = (ImgH - 1) / C1StrideH + 1;

// 1. alinäytteistyskerros, parametrit...
P1KernW = 2;
P1KernH = 2;
P1StrideW = 2;
P1StrideH = 2;
// ...ja dimensiot kun padding on false.
P1OutW = (C1OutW - P1KernW) / P1StrideW + 1;
P1OutH = (C1OutH - P1KernH) / P1StrideH + 1;

// 2. konvoluutiokerros, parametrit...
C2Maps = 7;
C2KernW = 3;
C2KernH = 3;
C2StrideW = 1;
C2StrideH = 1;
// ...ja dimensiot, padding true.
C2OutW = (P1OutW - 1) / C2StrideW + 1;
C2OutH = (P1OutH - 1) / C2StrideH + 1;
C2OutZ = C2Maps * C1Maps;

// 2. alinäytteistyskerros, parametrit...
P2KernW = 2;
P2KernH = 2;
P2StrideW = 2;
P2StrideH = 2;
// ...ja dimensiot, padding false.
P2OutW = (C2OutW - P2KernW) / P2StrideW + 1;
P2OutH = (C2OutH - P2KernH) / P2StrideH + 1;
}
```

Konvoluutioverkon määrittelyskripti (Net#)

```
// Kerrosten määrittely

input Picture [ImgH, ImgW];                    // 28, 28

hidden C1 [C1Maps, C1OutH, C1OutW]             // 5, 28, 28
  from Picture convolve {
InputShape  = [ImgH, ImgW];                    // 28,28
KernelShape = [C1KernH, C1KernW];              // 3, 3
Stride      = [C1StrideH, C1StrideW];          // 1, 1
Padding     = [T, T];
MapCount    = C1Maps;                          // 5
  }

hidden P1 [C1Maps, P1OutH, P1OutW]             // 5, 14, 14
  from C1 max pool {
InputShape  = [C1Maps, C1OutH, C1OutW];        // 5, 28, 28
KernelShape = [1, P1KernH, P1KernW];           // 1, 2, 2
Stride      = [1, P1StrideH, P1StrideW];       // 1, 2, 2
  }

hidden C2 [C2OutZ, C2OutH, C2OutW]             // 35, 14, 14
  from P1 convolve {
InputShape  = [C1Maps, P1OutH, P1OutW];        // 5, 14, 14
KernelShape = [1, C2KernH, C2KernW];           // 1, 3, 3
Stride      = [1, C2StrideH, C2StrideW];       // 1, 1, 1
Sharing     = [F, T, T];
Padding     = [F, T, T];
MapCount    = C2Maps;                          // 7
  }

hidden P2 [C2OutZ, P2OutH, P2OutW]             // 35, 7, 7
  from C2 max pool {
InputShape  = [C2OutZ, C2OutH, C2OutW];        // 35, 14, 14
KernelShape = [1, P2KernH, P2KernW];           //  1, 2, 2
Stride      = [1, P2StrideH, P2StrideW];       //  1, 2, 2
  }

hidden H3 [100]
  from P2 all;

output Result [10] softmax
  from H3 all;
```