TensorFlow.js

Käsialantunnistus

Hannu Oksman

Joose Seppälä

Antti Tarvainen

Harjoitustyö

Joulukuu 2018

Tieto- ja viestintätekniikan tutkinto-ohjelma

Tekniikan ja liikenteen ala

Sisällysluettelo

[1 Johdanto 2](#_Toc531720779)

[2 Neuroverkot 2](#_Toc531720780)

[2.1 Yleistä 2](#_Toc531720781)

[2.2 Motivaatio neuroverkkojen taustalla ja Sigma-neuroni ja konvoluutio-neuroverkko 3](#_Toc531720782)

[3 Käsinkirjoitettujen numeroiden tunnistaminen 4](#_Toc531720783)

[3.1 Mallin luonti 4](#_Toc531720784)

[3.2 Mallin opettaminen 6](#_Toc531720785)

[3.3 Tunnistus/ennustus 7](#_Toc531720786)

[4 Pohdinta 8](#_Toc531720787)

[Lähteet 8](#_Toc531720788)

[Liitteet 9](#_Toc531720789)

# Johdanto

Tutustuimme avoimeen TensorFlow.js -JavaScript-kirjastoon, jolla voi kouluttaa ja suorittaa koneoppimistoteutuksia selaimella ja Node.js -JavaScript-moottorilla. Luimme dokumentaatioita, tutoriaaleja ja katsoimme videoita. Oman neuroverkkototeutuksen kirjoittaminen on yllättävän yksinkertaista. Pienimmillään alle kymmenen skriptiriviä HTML-tiedostossa riittää. (TensorFlow.js n.d.)

Toteutimme käsin kirjoitettujen numeroiden tunnistamiseen kykenevän neuroverkon. Käytämme valmista dataa verkon opettamiseen ja selainkäyttöliittymässä voi piirtää numeroita ja mallimme tulkitsee sitä. Esittelemme tätä opintojakson seminaaripäivänä. Ohjelmakoodimme on saatavilla GitHub:sta ja sen voi rakentaa npm install –komennolla (vaatii Node.js-asennuksen).

Esittelemme tässä raportissa lyhyesti käyttämäämme konvoluutio-neuroverkkoa, joka on tehokas kuvantunnistuksessa. Sen jälkeen käymme läpi TensorFlow.js:n syntaksia koodiotteella, jossa verkkomme alustetaan. Lopuksi pohdimme tehtyä työtä ja jatkokehityskohteita.

# Neuroverkot

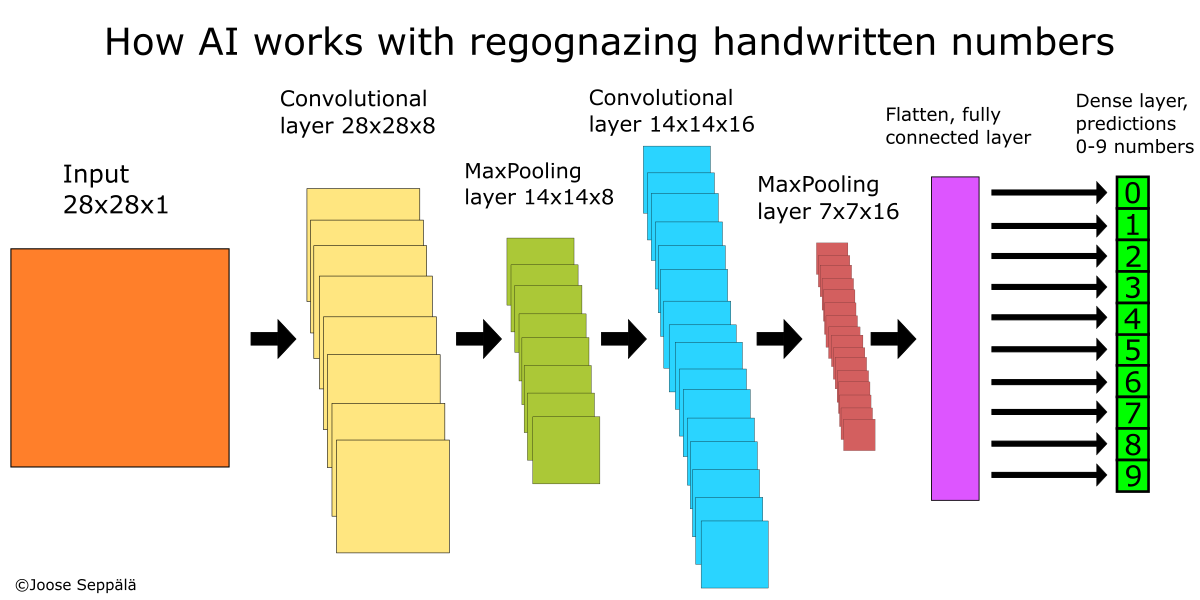
## Yleistä

Neuroverkot ovat yksi monista mahdollisista tekoäly- ja koneoppimisratkaisuista. Tyypillisesti ne sopivat hyvin yhden selkeästi rajatun asian havaitsemiseen ja niiden havaintotarkkuus on erittäin riippuvainen sille syötetystä opetusdatasta. Esimerkiksi meidän käyttämässä opetusdatassa on mustalla taustalla olevia valkoisia numeroita, jolloin verkko oppii yhdistämään tietyt vaaleat alueet kuvassa tiettyihin numeroihin. Neuroverkon havaitsemistarkkuus heikkenisi merkittävästi, jos vaihtaisimme taustan valkoiseksi ja numerot mustiksi, jolloin verkko ei enää tunnistaisi sen oppimia muotoja. Verkkoa pitäisi opettaa uudelleen toisenlaisella testidatalla. Emme ehtineet toteuttaa toimivaa ominaisuutta, jolla tätä olisi voinut demonstroida.

## Motivaatio neuroverkkojen taustalla ja Sigma-neuroni ja konvoluutio-neuroverkko

Keinotekoisilla neuroverkoilla mallinnetaan ihmisaivojen toimintaa, joka kykenee erilaisissa olosuhteissa luokittelemaan havainnot tehokkaasti. Aivot tallentavat tietoa malleina, jolloin se soveltuu potentiaalisesti monenlaisiin havaintotilanteihin. Esimerkiksi objektien, kuten toisten ihmisten kasvojen, havainnointi on mahdollista eri katselukulmista. (Taari 2017, 12-13.)

Neuroverkoissa on neuroneiksi kutsuttuja solmuja, joihin tyypillisesti syötetään monia arvoja ja ne generoivat syötteeksi yhden arvon. Sigma-neuronit voivat käsitellä liukuvia lukuja binaarien sijasta, joten ne ovat monikäyttöisiä verkkomallien toteutuksissa. Siis syötteiden ja tulosteiden arvot ovat tarkemmat. (Mts. 13.)

Konvoluutio-neuroverkko on monikerroksinen myötäkytkentäverkko, jossa edellisten neuronikerrosten tulos vaikuttaa seuraavaan kerroksiin. Konvoluutioverkossa kaikki edellisen kerroksen neuronit eivät kuitenkaan vaikuta kaikkiin seuraavan kerroksen neuroneihin. Neuronit käsittelevät pientä osaa kuvasta ja keskittyvät siinä oleviin eroavaisuuksiin kuvien luokittelussa. Tavoitteena on jatkuvasti edetä tarkemmasta kuvasta epätarkempaan, jolloin karkeustason noustessa verkko kestää pientä vääristymistä. Toisin sanoen verkko tunnistaa esimerkiksi tietyn numeron oikein, vaikka sitä ei aina piirtäisi täsmälleen samalla tavalla. (Mts. 20-22.)

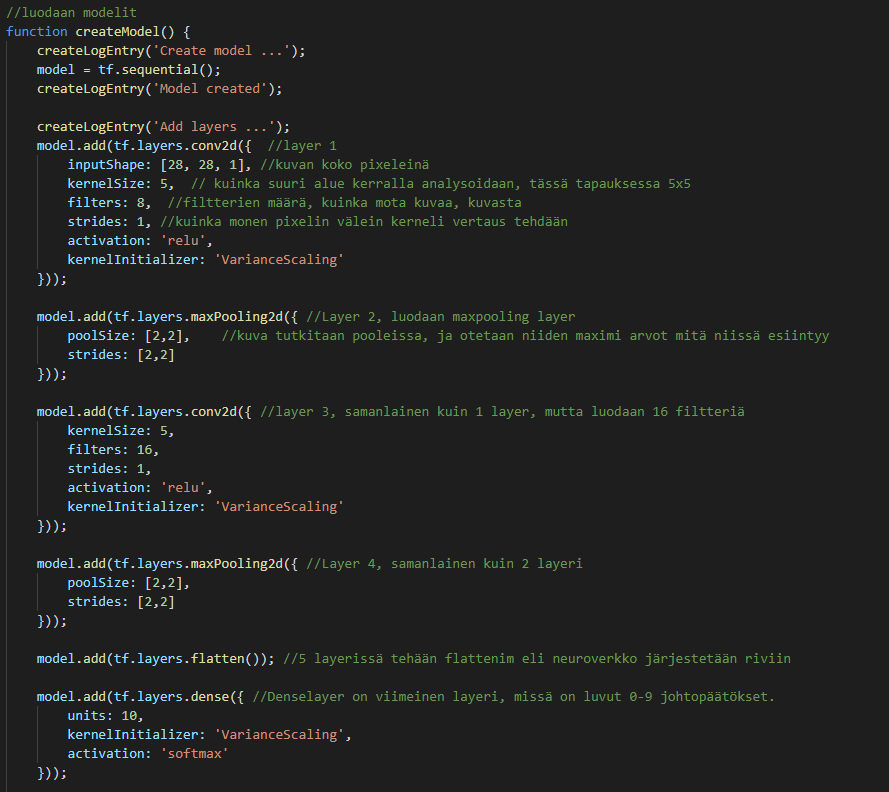
Kuva , neuroverkon rakenne ohjelmassamme.

# Käsinkirjoitettujen numeroiden tunnistaminen

Kävimme läpi Googlen TensorFlow.js-tutoriaalin mallin opettamisesta numeroiden tunnistamiseen. Käytimme MNIST-tietokannan (LeCun, Y. & Cortes, C. & Burges, C. N. d.) valmiita kuvia mallin opettamiseen ja lisäsimme ohjelmaan kentän, johon voi piirtää numeron ja antaa mallille tunnistettavaksi.

## Mallin luonti

Ohjelma aloitetaan myötäkytkentämallin (sequential) luomisella.



Kuva , koodista ote, missä luodaan neuroverkon tasot.

Malliin luodaan kuusi kerrosta, ensimmäisenä on kaksiulotteinen konvoluutiokerros (convolutional).

Kerrokselle määritellään erilaisia parametrejä:

* InputShape [28, 28, 1] (rivi, kolumni, syvyys). Malliin syötettävän datan muoto. Data on siis kokoelma kuvia, joiden koko on 28x28 pikseliä syvyydellä 1, koska värejä on vain yksi (mustavalko) (Machine Learning Glossary 2018)
* KernelSize 5. Suodatinikkunan koko 5x5 pikseliä, dataa käsitellään suodatinikkunan kokoisissa paloissa (mt.)
* Filters 8. Kuinka monta kernelSize:n kokoista ikkunaa otetaan käsittelyyn syötetystä datasta. Kuva jaetaan pienempiin osiin, joita käsitellään erikseen. (Training on Images n. d.)
* Strides 1. Määrittelee kuinka monen pikselin yli suodatinikkuna liukuu. Tässä tapauksessa ikkuna aloittaa vasemmasta yläkulmasta ja siirtyy yhden pikselin aina oikealle. Tilan loppuessa sivuttaissuunnassa ikkuna siirtyy takaisin vasempaan laitaan yhden pikselin yläreunasta ja jatkaa prosessia (Machine Learning Glossary 2018)
* Activation ‘relu’. Yleinen funktio (rectified linear unit) kuvien tunnistuksessa, joka saa syötteenä edellisen kerroksen syötteiden painotetun summan. Jos syöte on negatiivinen tai nolla, palauttaa 0 ja jos syöte on positiivinen, palauttaa syötteen arvon (mt.)
* KernelInitializer ‘varianceScaling’. Alustaa satunnaisesti mallin painokertoimet. (API Reference n. d.)

Toisena kerroksena lisäämme maxpooling2d-kerroksen. Tällä kerroksella interpoloidaan ensimmäisen kerroksen kuva niin, että se on puolet alkuperäisestä koosta.

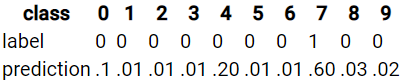
Kolmantena kerroksena luodaan toinen convolutional2d kerros, mutta isommalla filtterien määrällä.

Neljäntenä kerroksena maxpooling-kerros taas interpoloi kuvan puoleen koosta.

Viidentenä kerroksena edellisen kerroksen tulos muutetaan vektoriksi.

Kuudentena, viimeisenä kerroksena lisäämme tiheän kerroksen, jossa suoritetaan lopullinen luokittelu. Tiheään kerrokseen syötetään parametrit:

* Units: 10. Luvut 0-9
* KernelInitializer ‘varianceScaling’
* Activation ‘softmax’. Tekee todennäköisyyjakauman aikaisemmin määriteltyihin luokkiin. Luokka, jolla on isoin todennäköisyys, on tunnistamisen lopputulos.



Kuva , tunnistettu numero on 7 todennäkyöisyysjakaumalla.

Lopuksi malli pitää koota opetusta ja arviointia varten. Tämä toteutetaan model.compile() funktiolla. Funktioon määrittelemme optimointi ja loss funktiot:

* Optimizer train.sgd(0.15). Stochastic gradient descent. Gradient descent tekniikka minimoi häviön laskemalla häviön gradientit käyttämällä mallin parametrejä, jotka on käsitelty opetusdatalla. SGD käyttää yhtä satunnaisesti valittua dataa koko datan asemesta (Machine Learning Glossary 2018). Oppimisnopeutena 0.15 (Training on Images n. d.)
* Loss ‘categoricalCrossentropy’. Mittaa eroa edellisen kerroksen todennäköisyysjakauman ja dataan liitetyn tunnisteen välillä. Palauttaa pienemmän häviöarvon, jos todennäköisyysjakauman arvio on lähellä tunnistetta ja isomman arvon, jos ei. Opetuksen aikana malli päivittää parametrejaan pienentääkseen häviöarvot koko datasetissä. (API Reference n. d.)

## Mallin opettaminen

Data haetaan Googlen tarjoamasta MNIST APIsta. Datana käytetään valmiiksi luokiteltuja kuvia valkoisista numeroista mustalla pohjalla.

Opettamista varten määritellään erän koko (batch\_size) ja opetuserien määrä. Yksi erä kertoo, kuinka monta kuvaa mallin on käsiteltävä ennen parametrien päivitystä. Opetuserien määrä ilmaisee, kuinka monen erän verran mallia opetetaan. (Training on Images n. d.)

Opetus tapahtuu for-loopissa ja tensorien käsittely tidy() funktion sisällä poistaa tarpeettomat tensorit muistista käytön jälkeen. (Training on Images n. d.)

Data haetaan Googlen tarjoamalla tavalla, jota emme käy tässä läpi.

Data saadaan muodossa [batch\_size, 784] (28\*28 = 784), mutta malli odottaa dataa muodossa [batch\_size, 28, 28, 1]. Reshape() funktiolla saamme muokattua datan haluamaamme muotoon. (Training on Images n. d.)

Model.fit(batch.xs, batch.labels, {batchSize: BATCH\_SIZE, epochs: 1}) funktiolla tapahtuu varsinainen opetus. Syötteeksi vaaditaan:

* x (tensori tai taulukko tensoreita). Tässä tapauksessa x = kuvat numeroista
* y. Tunnisteet, numero, jota kuva esittää
* BatchSize. Määrittelimme erän kooksi 64 kuvaa
* Epochs. Määrittää kuinka monta kertaa koko opetusprosessi tehdään. (Training on Images n. d.)

## Tunnistus/ennustus

Predict(batch) funktioon saadaan syötteenä yksi kuva. Kuvan avulla funktio yrittää suorittaa ennustuksen. Kuvasta otetaan talteen alkuperäinen tunniste, jotta voimme verrata ennustuksen tulosta ja alkuperäistä tunnistetta.

Itse tunnistaminen tapahtuu kutsumalla funktiota mallin avulla model.predict(batch.xs.reshape([-1, 28, 28, 1]). Tämä palauttaa todennäköisyysjakauman taulukkona, arvot taulukossa ovat väliltä 0 ja 1. Taulukosta poimitaan suurimman todennäköisyyden luku ja verrataan tätä aikaisemmin poimittuun kuvan tunnisteen lukuun. Jos tunnisteen luku ja poimittu arvo ovat samat, on tunnistus onnistunut. (Eschweiler 2018.)

# Pohdinta

Työ onnistui hyvin. Pääsimme tunnistamaan omia käsin kirjoitettuja numeroita, mitkä tekoäly tunnisti onnistuneesti pääsääntöisesti. Mielenkiintoista olisi ollut vielä opettaa tekoälyä juurikin omilla piirtämillä numeroilla ja katsoa, kuinka hyvin se oppisi niistä. Tämä tosin olisi vienyt niin paljon aikaa, että sitä ei ehditty tämän laajuisessa tutkimuksessa toteuttamaan.

TensorFlow.js on hyvä valmis ohjelmisto paketti alkaa kokeilemaan ja koodaamaan itse tekoälyä. Sen asentaminen on helpompaa kuin python versio Windowsille. Python versiota kokeiltiin asentaa, mutta toimintaympäristöä ei saatu toimimaan koneilla.

Lähteet

API Reference. N. d. TensorFlow.js tietotyyppien, luokkien jne. kuvaukset. Viitattu 3.12.2018. <https://js.tensorflow.org/api/0.13.3/>.

Core Concepts in TensorFlow.js. N. d. Dokumentaatio. Viitattu 4.12.2018.  
<https://js.tensorflow.org/tutorials/core-concepts.html>.

Eloranta, E. 2003. Geofysiikan kenttäteoria. Verkkoversio. Viitattu 3.12.2018.  
<http://www.julkari.fi/bitstream/handle/10024/123567/stuk-a198.pdf>.

Eschweiler, S. 2018. TensorFlow.js Crash Course - Machine Learning For The Web – Handwriting Recognition. Video. Viitattu 3.12.2018. <https://www.youtube.com/watch?v=QJQTIp5McV8>.

Machine Learning Glossary. 2018. Googlen käsitelista koneoppimisesta. Viitattu 4.12.2018. https://developers.google.com/machine-learning/glossary/.

LeCun, Y. & Cortes C. & Burges, C. N. d. MNIST DATABASE of handwritten digits. Tietokanta käsinkirjoitetuista numeroista. Viitattu 3.12.2018. <http://yann.lecun.com/exdb/mnist/>.

Taari, N. 2017. Ilmakuvien analysointi digitaalisesti TensorFlow ohjelmistolla. Opinnäytetyö, AMK. Jyväskylän ammattikorkeakoulu, tekniikan ja liikenteen ala, tietotekniikan koulutusohjelma. Viitattu 2.12.2018. <https://www.theseus.fi/bitstream/handle/10024/131097/Opinnaytetyo-julkaisu.pdf?sequence=1&isAllowed=y>.

Training on Images: Recognizing Handwritten Digits with a Convolutional Neural Network. N. d. Tutoriaali kuvantunnistuksesta. Viitattu 4.12.2018.  
<https://js.tensorflow.org/tutorials/mnist.html>.

TensorFlow.js. N.d. Tuote- ja dokumentaatiosivu. Viitattu 2.12.2018.  
<https://js.tensorflow.org/>.

Liitteet

DrawTensor, 2018. Lähdekoodi harjoitustyöhön.  
<https://github.com/Jooseba/DrawTensor>