jamk.fi

Koneoppimisen hyödyntäminen web-sovellusten kehityksessä

Ari Aittomäki

Opinnäytetyö
Toukokuu 2018
Tekniikan ja liikenteen ala
Insinööri (AMK), mediatekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Aittomäki, Ari	Opinnäytetyö, AMK	24.05.2018
	Sivumäärä 32	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Koneoppimisen hyödyntäminen web-sovellusten kehityksessä

Tutkinto-ohjelma
Mediatekniikan koulutusohjelma

Työn ohjaaja(t)
Manninen, Pasi

Toimeksiantaja(t)
Nordcloud Solutions

Tiivistelmä

Opinnäytetyön toimeksiantajana toimi Nordcloud Solutions. Toimeksiantaja oli aikaisemmin kehittänyt web-sovellusten toteuttamisessa käytettävää suunnittelujärjestelmää, joka kattaa kaikki web-sovellusten eri suunnitteluelementit. Näihin elementteihin kuuluu muun muassa typografia, värit, tekstin koot ja näistä muodostuvat käyttöliittymien rakennuspalikat.

Suunnittelujärjestelmä tarjoaa nykyisellään suurille tuote-ekosysteemeille tehokkaan tavan luoda uusia palveluja käyttämällä jo olemassa olevia komponentteja. Vaikka kyseinen järjestelmä onkin käytössä, on web-sovellusten suunnittelu- ja toteutusvaiheessa turhaa toistoa. Toimeksiantaja halusikin selvittää, että onko suunnittelutyön automatisointi mahdollista käyttäen nykyisiä teknologioita ja kuinka järkevää se olisi.

Koneoppimista hyödyntämällä on mahdollista toteuttaa monimutkaisiakin esineitä tunnistava ohjelma. Ongelmaksi muodostui koulutusaineiston määrä, joka oli pääsääntöisesti piirrettävä käsin.

Käsin piirretyt käyttöliittymäkomponentit olivat kuitenkin muodoltaan yksinkertaisia, joten perinteisempi lähestyminen kuvantunnistukseen oli perusteltua. MSER ja siirto-oppimista hyödyntämällä uudelleen koulutettu MobileNet-arkkitehtuuriin pohjautuva neuroverkko yhdessä antoivat riittävän suorituskyvyn. Nämä teknologiat todistivat, että halutunlainen toiminnallisuus on mahdollista toteuttaa pienelläkin koulutusaineistolla.

Tuotantokelpoisen tarkkuuteen ja toimivuuteen ei päästy, joten työ jäi prototyypin tasolle. Jatkokehityksen jälkeen tämänkaltainen tuote voisi olla osana jotain suurempaa suunnittelujärjestelmää.

Avainsanat (asiasanat)

Koneoppiminen, komponenttikirjasto, automatisointi

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Aittomäki, Ari	Bachelor's thesis	24.05.2018
		Language of publication: Finnish
	Number of pages: 32	Permission for web publication: x

Title of publication

Utilizing machine learning in web application development

Degree programme

Media Engineering

Supervisor(s)

Manninen, Pasi

Assigned by

Nordcloud Solutions

Abstract

This study was assigned by Nordcloud Solutions. The assigner had previously developed a design system that can be used during web development. The design system would cover all elements used in a web application including colors, typography, fonts and the building blocks that they form.

The design system would offer large product ecosystem the ability to create new products with existing components and maintain consistency across different products. Web design and development process still contain too much repetition, even with the design system in place. The assigner wanted to find out if there was a way to eliminate some parts of the process, and how feasible would it be.

Utilizing machine learning enables to produce a software that can recognize vast number of different objects. The biggest problem with this was the amount of available training data. All the data was going to be drawn by hand.

Hand drawn elements were simple enough; thus, a more traditional approach to image recognition was justified. Utilizing MSER algorithm and retraining a MobileNet based neural network using transfer learning gave the desired outcome. These technologies proved that a complex image recognition is doable with a small amount of data.

Production grade accuracy and functionality were not achieved. Hence, the work remains in a prototype status. After further development, these kinds of products could be featured as part of a design system.

Keywords/tags (subjects)

Machine Learning, Pattern Library, Automatization

Miscellaneous

Sisältö

Kuviot

1 Työn lähtökohdat

1.1 Tausta ja tavoitteet

Tämän opinnäytetyön tavoitteena oli toteuttaa prototyyppi sovelluksesta, joka muuttaa käyttöliittymäkuvan verkkosivuksi. Tämä edellyttää, että käytössä on komponenttikirjasto, johon on määritelty, miltä mikäkin komponentti näyttää. Komponenttikirjaston avulla uusien sivujen luominen on huomattavasti nopeampaa, sillä sivun rakentamiseen tarkoitetut rakennuspalikat on jo määritelty.

Esimerkkinä web-suunnittelija on määrätty toteuttamaan uusi sivu asiakkaalle. Palaverin aikana suunnittelija kuuntelee asiakkaan vaatimukset ja luonnostelee myöhemmin verkkosivun. Tämä luonnos muutetaan oikean näköiseksi käyttöliittymäksi jollain käyttöliittymäprototyyppien luomiseen tarkoitetulla työkalulla. Asiakkaan hyväksynnän jälkeen prototyyppi annetaan web-kehittäjälle, joka puolestaan kirjoittaa varsinaisen koodin.

Sen sijaan, että luonnos toteutettaisiin palaverin jälkeen, se tehtäisiinkin palaverin aikana yhdessä asiakkaan kanssa. Verkkosivusta piirrettäisiin rautalankamalli ja siitä otettaisiin kuva, jonka avulla käyttöliittymä muodostettaisiin jo palaverin aikana.

Tämä mahdollistaisi ideoiden realisoitumisen nopeammin, ja niitä voitaisiin testata välittömästi. Vaikka kaikkia haluttuja komponentteja ei saataisikaan näkyviin palaverin aikana, muodostuisi silti realistisempi kuva lopputuloksesta ja siitä, toimiiko jokin idea tai asettelu käytännössä. Myös sisältöä voitaisiin alkaa työstämään heti kun on jotain viitettä siitä, minkälaisia elementtejä sivustolla esiintyy.

Opinnäytetyöllä haluttiin todentaa, onko nykyisillä teknologioilla mahdollista toteuttaa halutunlainen prototyyppi ja kuinka käyttökelpoinen se on verrattuna vaadittaviin resursseihin.

1.2 Toimeksiantaja

Opinnäytetyön toimeksiantajana oli Nordcloud Solutions, joka tunnettiin aikaisemmin nimellä SC5 Online Oy. Nordcloud Solutions on pilvinatiivien sovellusten suunnitteluun ja toteuttamiseen erikoistunut ohjelmistotalo. Tällaisten sovellusten suunnittelussa ja toteutuksessa hyödynnetään niin alan tuoreimpia design-trendejä kuin uusimpia teknologioita ja tekoälyä

Syksyllä 2017 SC5 Online Oy ja pilvipalveluintegrointia tarjoava Nordcloud yhdistyivät. Yhdistymisen takana oli kasvava digitalisaatio sekä alati pilvipalvelualustalle siirtynyt sovelluskehitys. Yhtiöiden yhteinen liikevaihto vuonna 2016 oli 30 miljoonaa ja työntekijöitä 250. (Korpimies 2017.)

1.3 Aikaisempia julkaisuja

AirBnB julkaisi sisäiseen käyttöön tarkoitetun prototyypin, joka kykeni muuttamaan videokuvasta löydetyt käyttöliittymäkomponentit verkkosivuksi. Ajatuksena oli, että jos koneoppimisalgoritmit voivat tunnistaa kiinalaisia käsinkirjoitettuja symboleita hyvällä tarkkuudella, pitäisi 150 käyttöliittymäkomponentin tunnistaminen olla helppoa (Wilkins N.d.)

Emil Wallner julkaisi vuonna 2018 tutkimuksen, jossa käyttöliittymäluonnokset muutetaan HTML- ja CSS-koodiksi käyttämällä syväoppimista. Työssään hän opetti neuroverkon generoimaan kuvasta koodia, joka eroaa tässä opinnäytetyössä sekä AirBnB’n lähestymistavasta siten, että valmista komponenttikirjastoa ei käytetä. Neuroverkko luo kaiken tarvittavan koodin ilman ihmisen puuttumista asiaan.

Tony Beltramelli julkaisi vuonna 2017 prototyypin palvelusta, jossa käyttöliittymäkuvat voidaan muuttaa suoraan koodiksi tai Sketch-ohjelman käyttämäksi tiedostoksi. Prototyypin ajatuksena oli opettaa kone tunnistamaan graafisia käyttöliittymiä nykyisen työnkulun parantamiseksi.

2 Käytetyt teknologiat

2.1 Koneoppiminen ja kuvantunnistus

Tensorflow

Tensorflow on avoimen lähdekoodin kirjasto numeerisille laskutoimituksille, jossa käytetään hyväksi tiedonkulkukaavioita. Tensorflow'n pääasiallinen tarkoitus on koneoppimisen ja syväoppivien neuroverkkojen toteuttamiseen tarvittavien mallien luominen, mutta kykenee myös muihin tehtäviin. Tensorflow tukee Java-, C-, Go- ja Python-ohjelmointikieliä. Alun perin kirjasto kehitettiin Googlen kehitys ja tutkimustiimin tarpeita varten, mutta myöhemmin se julkaistiin myös kaikkien saataville. (About Tensorflow n.d.)

Koneoppimisessa tarvitaan usein paljon laskutoimituksia, joiden suorittaminen on huomattavasti nopeampaa näytönohjaimella kuin prosessorilla. Tästä syystä Tensorflow'sta löytyykin tuki Nvidian valmistamille näytönohjaimille laskutoimitusten nopeuttamiseksi. Muita suosittuja koneoppimiskirjastoja ovat Pythonille suunniteltu PyTorch, Caffe sekä Keras, joista kaikki tukevat myös näytönohjaimen käyttöä laskutoimituksien suorittamiseen. Nvidian mukaan erityisesti Caffe suoriutuu laskutoimituksista 65% nopeammin uusimpaan Pascal-arkkitehtuuriin pohjautuvia näytönohjaimia käytettäessä (GPU-Accelerated Caffe n.d).

OpenCV

OpenCV on avoimen lähdekoodin konenäkö- ja koneoppimiskirjasto. Kirjasto sisältää 2500 algoritmia, joita voidaan käyttää esimerkiksi kasvojen ja esineiden tunnistamiseen, kohteiden havainnoimiseen sekä tunnistamaan monenlaisia liikkuvia esineitä. Kirjastossa on rajapinnat C++-, Python-, Java- ja Matlab-ympäristöille sekä tuki Windows, Linux, Android ja Mac OS käyttöjärjestelmille. (OpenCV-kirjaston kuvaus n.d.)

2.2 Web-teknologiat

Flask

Flask on mikroarkkitehtuuri web-sovellusten suorittamiselle. Flask on rakennettu Python-ohjelmointikielen päälle ja toimii kuin mikä tahansa muukin web-palvelin. Etuna Flaskin käyttämisessä on usein koneoppimisessa käytettävän Pythonin suorittaminen suoraan palvelimella ilman välikäsiä. Flask mukautuu tarvittaessa useisiin erilaisiin käyttötarkoituksiin usean eri lisäosansa ansiosta. (Flask web development, one drop at a time n.d.)

Storybook

Storybook on käyttöliittymäkomponenttien toteuttamiseen tarkoitettu kirjasto, joka on kirjoitettu JavaScriptilla. Storybookin avulla voidaan luoda ns. Pattern Library eli kirjasto, joka sisältää kaikki käyttöliittymän rakennuspalikat. Tämän kirjaston avulla uusien sivujen luominen on paljon nopeampaa ja komponenttien päivittäminen on vaivattomampaa, sillä kirjasto on eristetty varsinaisesta sovelluksesta. Storybook tukee Angular-, Vue- sekä React-kirjastoja. (Introduction n.d.)

React

React on JavaScriptillä kirjoitettu kirjasto käyttöliittymien luomiseen. Reactilla voidaan rakentaa kapseloituja komponentteja, jotka hallitsevat oman tilansa. Näitä komponentteja käyttämällä voidaan luoda monimutkaisiakin käyttöliittymiä. Komponenttien ansioista React soveltuu erinomaisesti käytettäväksi Storybookin kanssa. (React - A JavaScript library for building user interfaces n.d.)

Reactin kanssa suositellaan käytettävän JavaScriptin syntaksia muuttavaa JSX-kieltä. JSX on olio-ohjelmointikieli ja käännetään suoritusvaiheessa puhtaaksi JavaScriptiksi. JSX muistuttaa kieltä, jota normaalisti käytetään käyttöliittymän mallin (engl. template) tekemiseen. (Introducing JSX n.d.)

3 Mallikirjasto

3.1 Yleistä

Mallikirjasto on työkalu, jolla voidaan kerätä ja jakaa suunnitellussa käytettäviä malleja ja ohjeita niiden käytöstä. Perinteisessä tyylikirjastossa keskitytään usein fonttien tyyleihin, väreihin ja ikoneihin. Mallikirjasto sen sijaan sisältää usein laajemman kattauksen suunnittelumalleja. (Khomaltova 2017, 11 - 12.)

Joillekin tiimeille systemaattinen lähestyminen suunnitteluun ja digitaalisten tuotteiden rakennukseen on mahdotonta ilman mallikirjastoa (engl. Pattern library). Tehokkain mallikirjasto luodaan yhteistyössä suunnittelijan, kehittäjän ja asiakkaan kanssa. Tällöin vältytään tilanteelta, jossa kehittäjä ei tunne kaikkia saatavilla olevia malleja tai ne eivät ole käytännöllisiä. Malleja saatetaan määritellä liian tarkasti tietynlaiseen aineistoon, jolloin yksikin ylimääräinen merkki saattaa työntää tärkeää tietoa pois näkyvistä. (Mts. 243 - 245.)

3.2 Suunnittelumalli

Mallikirjasto koostuu yksittäisistä suunnittelumalleista (engl. desing pattern), joita kutsutaan usein nimellä malli (engl. pattern). Ne ovat toistuvia ja uudelleen käytettäviä elementtejä, joiden avulla voidaan ratkaista suunnitteluongelmia. Mallit ovat toistuvia käyttöliittymäelementtejä, kuten interaktioita, painikkeita, tekstikenttiä, värejä ja fontteja. Näitä malleja yhdistelemällä luodaan tuotteen käyttöliittymä. Digitaalisissa tuotteissa malleja voidaan käyttää esimerkiksi helpottamaan tuotteen käyttämistä tai tarjoamaan käyttäjälle palautetta. (Khomaltova 2017, 18 - 23.)

Suunnittelumallit voidaan jakaa toiminnallisiin ja havainnollisiin malleihin. Toiminnalliset mallit (engl. functional patterns) ovat konkreettisia käyttöliittymän rakennuspalikoita. Niiden tarkoituksena on mahdollistaa ja sallia toimintojen tekeminen käyttöliittymässä. Toiminnallinen malli voi esimerkiksi olla käyttöliittymän osa, joka pyytää kirjautumaan sisään ja tai valitsemaan haluamansa reseptin. Toiminnallisilla malleilla halutaan siis rohkaista käyttäjää tietynlaiseen käyttäytymiseen. Havainnolliset mallit

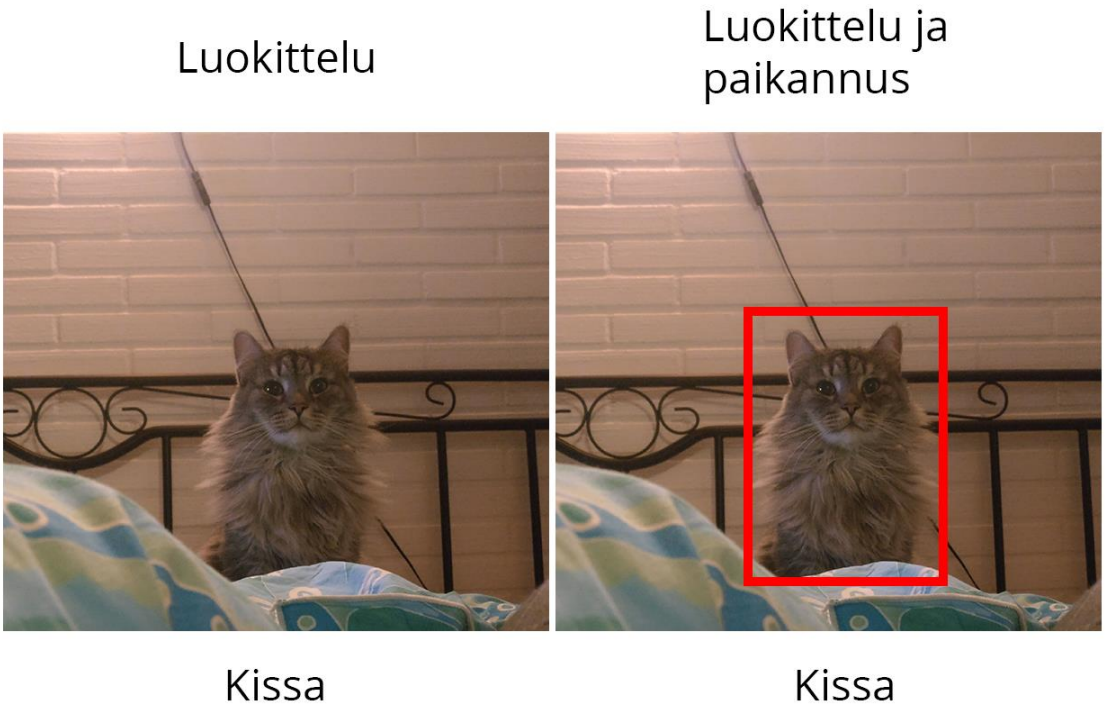
(engl. perceptual patterns) sen sijaan edustavat niitä piirteitä toiminnallisissa malleissa, jotka tekevät niistä yksilöllisen tietylle brändille tai yritykselle. Havainnollisiin malleihin kuuluvat muun muassa typografia, väripaletti, asetelmat, ikonit, tekstuurit, välitykset ja animaatiot. Toiminnalliset mallit siis edustavat sitä, mitä käyttäjät haluat ja tekevät, kun taas havainnolliset mallit heijastavat käyttäjän tunteita tai sitä, mitä käyttäjä tekee intuitiivisesti. (Mts. 63 – 89.)

4 Hahmon- ja kuvantunnistus

4.1 Yleistä

Ihminen on erityisen hyvä havainnoimaan hahmoja. Tämä on kuitenkin vaikea tehtävä tietokoneelle (Image recognition 2018). Hahmontunnistuksella (engl. pattern recognition) tarkoitetaan syötteeksi saadun luonnollisen kohteen, esimerkiksi kuvan tai videon, mittaamista ja havainnoimista. Näiden mittausten ja havaintojen perusteella on mahdollista toteuttaa automaattista analyysiä, jonka avulla kohde voidaan tunnistaa. (Koistinen 2002.)

Kuvasta voidaan haluta löytää esimerkiksi kissa (ks. Kuvio 1). Tällöin luokittelua (engl. image classification) voidaan käyttää hyödyksi kuvasta löytyvän esineen luokitteluun, eli kertomaan mikä tunnistimelle opetetuista luokista kuvassa on. Mikäli halutaan myös paikantaa kuvasta löydetty kohde, on kyseessä objektin tunnistus (engl. object detection). Objektin tunnistuksella voidaan ratkaista ongelma, jossa samassa kuvassa on useampi kohde samasta tai eri luokasta. (Sachan 2017.)



Kuvio 1. Kohteen luokittelu ja paikannus

4.2 Tunnistusteknologioita

4.2.1 MSER

Maximally Stable Extremal Regions eli MSER on menetelmä yhtenäisten alueiden havaitsemiseen. MSER algoritmi irrottaa kuvasta alueita, jotka pysyvät muuttumattomina laajalla alalla raja-arvoista. Kaikki tietyn raja-arvon alle jäävät arvot muutetaan valkoisiksi ja suuremmat tai yhtä suuret muutetaan mustiksi. (Del Bimbo n.d.)

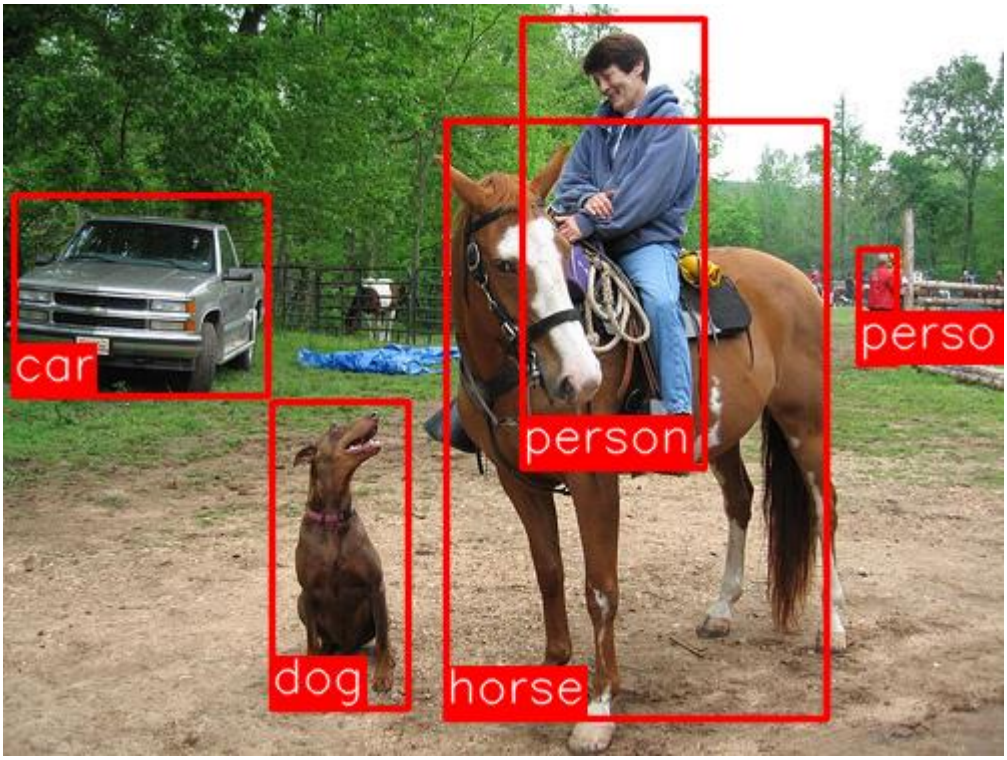
MSER edustaa perinteisempää tapaa tunnistaa kuvasta yhtenäisiä alueita. Menetelmä toimii parhaiten silloin, kun kuvan sisältämät alueet ovat selkeästi rajattuja. MSER itsessään ei kerro, mitä kuvassa on, mutta sitä voidaan käyttää kiinnostavien alueiden rajaamiseen. MSER-algoritmia voidaan käyttää yhdessä reunojen tunnistamiseen tarkoitetun Canny Edge -tunnistimen kanssa parantamaan MSER-algoritmin suorituskykyä (Sneha & Vinod 2017).

4.2.2 Single Shot Multibox -tunnistin

Yksi uusimmista menetelmistä kohteiden tunnistamiseen kuvasta tai videosta on käyttää Single Shot Multibox -tunnistinta, josta käytetään lyhennettä SSD (Eng. Single Shot Multibox Detector). Tämä menetelmä edustaa yhtä useasta tavasta tunnistaa kohteita käyttämällä koneoppimista.

SSD käyttää syvää neuroverkkoa kohteiden tunnistamiseen ja piirtää jokaisen kuvassa tai videossa esiintyvän mahdollisen kohteen ympärille sitä rajaavan laatikon. SSD palauttaa todennäköisyyden prosentteina siitä, mikä kuvassa on ja kuinka varma se on siitä. SSD piirtää kuvaan tai videoon huomattavasti useamman laatikon verrattuna muihin menetelmiin. Tämän ongelman ratkaisemiseksi käytetään tekniikkaa nimeltä Non-Maximum Supression, jossa poistetaan lähellä tai päällekkäin esiintyvät laatikot, jotka uskovat löytäneensä saman kohteen. (Liu, Anguelov, Erhan, Szegedy, Reed, Fu & Berg 2015.)

SSD:n lisäksi muita suosittuja koneoppimista hyödyntäviä menetelmiä ovat R-CNN, Fast R-CNN, Faster R-CNN ja Yolo. Kaikki edellä mainitut menetelmät tuottavat samanlaisen lopputuloksen, jossa kuvasta löytyneiden kohteiden ympärille on piirretty niitä rajaavat laatikot (ks. kuvio 2). Myös kuvantunnistusta tarjoavia palveluita on saatavilla lukuisia. Esimerkiksi Amazonin julkaisema Rekognition Image ja Rekognition Video.



Kuvio 2. R-CNN-tunnistimen piirtämät laatikot (Forson 2017)

5 Koneoppiminen

5.1 Yleistä

Koneoppimisella tarkoitetaan järjestelmää tai ohjelmaa, joka on rakentaa ennustavan mallin sille annetusta aineistosta. Järjestelmä käyttää tätä opittua mallia ennusteiden tekemiseen aineistosta, jota se ei ole koulutuksessa käytettävässä aineistossa kohdannut. Koneoppimisella tarkoitetaan myös tieteenalaa, joka tutkii näitä ohjelmia tai järjestelmiä. (Machine Learning Glossary n.d.)

Koneoppimismallit ovat luonteeltaan todennäköisyyspohjaisia. Tällä tarkoitetaan, että koulutetut mallit palauttavat todennäköisyyksiä. Esimerkiksi koulutetulle koneoppimismallille voidaan antaa kuva kissasta, ja malli palauttaa prosentuaaliset todennäköisyydet sille, että kuvan eläin on kissa tai jokin muu mallille koulutettu luokka.

5.2 Oppimismenetelmät

Oppiminen on jaettu yleensä kolmeen peruskategoriaan: Vahvistettu, valvottu ja valvomaton oppiminen. Näiden menetelmien lisäksi on siirto-oppiminen.

5.2.1 Vahvistettu oppiminen

Ihminen oppii asioita itsestään saavuttaakseen parhaita palkkioita. Vahvistettua oppimista (engl. reinforcement learning) on, kun malli opettelee yrityksen ja erehdyksen kautta saavuttamaan parempia tuloksia. Vääristä vastauksista voidaan rangaista, kun taas oikeista voidaan palkita. Malli muodostaa oman tietämyksensä pelkkien syötteiden perusteella, kuten näön tai jonkin muun sensorisyötteen, ilman käsin rakennettuja ominaisuuksia tai heuristiikkaa. (Silver 2016.)

5.2.2 Valvottu oppiminen

Valvotussa oppimisessa mallille syötettävä aineisto on merkitty ja haluttu lopputulos on tiedossa. Opetusprosessin aikana malli tekee ennustuksia, ja mikäli ennuste on väärä, ilmoitetaan sille oikea vastaus. Tätä jatketaan niin kauan, kunnes malli on saavuttanut halutun tarkkuuden. Käyttötarkoituksena tällaiselle mallille on yleensä luokittelu ja regressio. (Brownlee 2013.)

5.2.3 Valvomaton oppiminen

Valvomattomassa oppimisessa mallille syötettävää aineistoa ei ole merkitty eikä lopputulosta ole tiedossa. Tässä tilanteessa malli yrittää löytää aineistosta rakenteita esimerkiksi yhdistelemällä samankaltaisia näytteitä. Valvomatonta oppimista voidaan käyttää esimerkiksi klusterointiin, algoritmiseen dimensionpudotukseen tai assosiaatiosääntöanalyysiin. (Brownlee 2013.)

5.2.4 Siirto-oppiminen

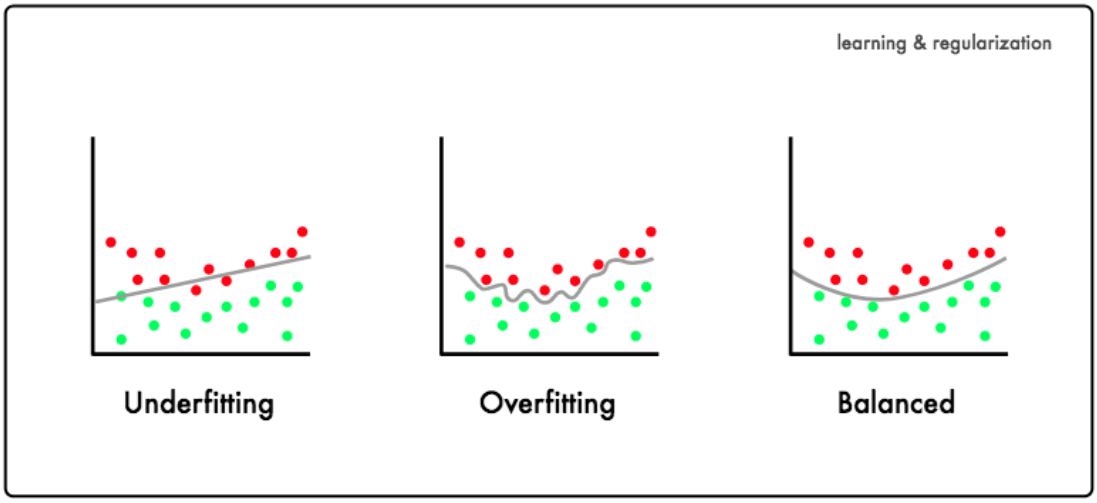
Siirto-oppiminen on koneoppimismallin opetusmenetelmä, jossa aikaisemmin toisenlaiseen tehtävään opetettu malli opetetaan suorittamaan toisenlainen tehtävä (Torrey & Shavlik 2009). Hyvän syväoppimismallin rakentaminen alusta asti on hidasta sen vaatiman aineiston vuoksi. Esimerkiksi mallin kouluttaminen ImageNet-kuvakirjastolla, jossa on 1,2 miljoonaa kuvaa, kestää 2-3 viikkoa saatavilla olevasta laitteistosta riippuen. Siirto-oppiminen mahdollistaa olemassa olevan mallin opettamisen uudestaan huomattavasti lyhyemmässä ajassa kuin alusta asti kouluttamalla. Tästä

esimerkkinä on Googlen luoma Inception-V3-mallin, joka on koulutettu ImageNet-kuvakirjastolla. Mallin uudelleen kouluttaminen, jotta se tunnistaisi uusia luokkia, voidaan toteuttaa alle tunnissa. (Thompson 2016.)

Verkossa on tarjolla useita palveluita, jotka tarjoavat mahdollisuuden kouluttaa uudestaan malleja omalla aineistolla. Näistä esimerkkinä on Microsoftin omistama Customvision-palvelu, jossa pienelläkin aineistolla voidaan luoda toimiva kuvia tunnistava malli.

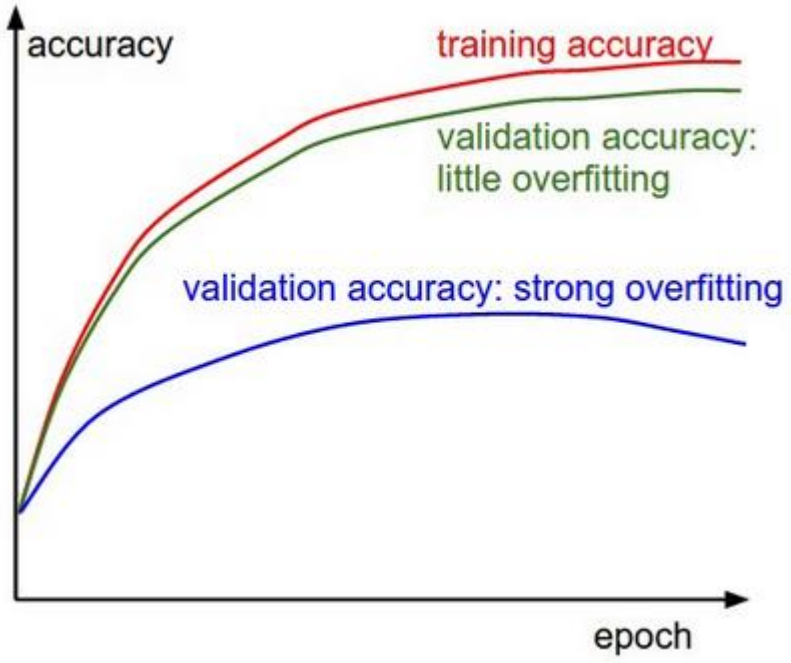
5.3 Ylisovittaminen

Yleisin kohdattu ongelma niin tilastotieteessä kuin koneoppimisessa on ylisovittaminen (ks. kuvio 3). Sana ylisovittaminen siitä, kun tilastollinen malli sovitetaan vastaamaan aikaisempia havaintoja liian tarkasti. Tällöin vaarana on, että malli sovitetaan aineiston kohinaan, eli merkityksettömään aineistoon, eikä sen perusrakennetta löydetä. Ylisovittamisen vastakohtana on alisovittaminen, jota ilmaantuu silloin kun malli ei hyödynnä aineistoa tarpeeksi tehokkaasti. (Silver 2012, 170-174.)



Kuvio 3. Kuvaajat sovitetuista malleista. Vasemmalla alisovitettu, keskellä ylisovitettu ja oikealla hyvin sovitettu malli (Wallner 2017)

Käytännössä ylisovittaminen ilmenee koulutus- ja validointiaineiston välisenä tarkkuuserona. Kuviossa 4 on esitetty kaksi erilaista tilannetta, jossa ylisovittamista voidaan havaita. Sininen suora kuvastaa tilannetta, jossa ylisovittaminen on todella vahvaa ja vihreä suora kuvastaa heikkoa ylisovittamista. Tämä tilanne voidaan korjata lisäämällä normalisointia tai suurentamalla aineiston kokoa.



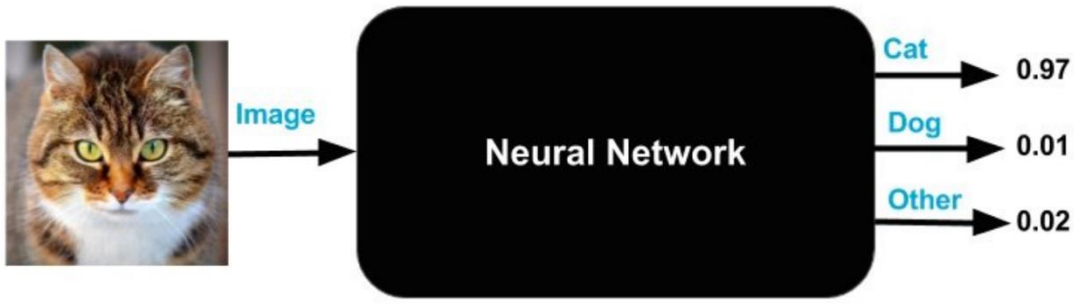
Kuvio 4. Käytännön esimerkki sovittamisesta (Kožnarová 2017)

Ristiinvalidointi on tehokas tapa vähentää ylisovittamista. Muita ylisovittamisen estämiseen käytettyjä tekniikoita ovat normalisointi, opetuksen aikainen lopettaminen ja ominaisuuksien poisto. Esimerkiksi kuvitellaan tilanne, jossa on sadan datapisteen aineisto. Sen sijaan, että aineisto jaettaisi 2:8 suhteella opetus- ja testausaineistoksi, se jaetaan kymmeneksi 20 datapisteen lohkoksi. Tämän jälkeen malli opetetaan yhdeksää lohkoa käyttäen ja testataan käyttäen yhtä lohkoa. Tämä operaatio toistetaan, kunnes kaikki lohkot ovat olleet opetus- ja testiaineistona. (The 5 Levels of Machine Learning Iteration 2017.)

5.4 Neuroverkko

Neuroverkko on matemaattinen malli, jonka rakenne ja toiminta pohjautuvat aivojen toimintaan. Verkko koostuu neuroneista ja niitä yhdistävistä synapseista, jotka opetetaan suoriutumaan halutusta tehtävästä opetusaineiston avulla. Prosessi jäljittelee ihmisen tapaa oppia, jossa havainnon jälkeen tarkistetaan, osuiko arvaus oikeaan. Tämän tiedon perusteella neuronien välisiä painoja päivitetään ja verkko oppii suoriutumaan annetusta tehtävästä paremmin. (Kangasniemi 2017.)

Neuroverkon perusperiaate voidaan kuvata mustana laatikkona, kuten kuviossa 5 on esitetty. Vasemmalla on neuroverkolle syötettävä kuva ja oikealla on prosentuaaliset arvot siitä mihin luokkaan kuvassa esiintyvä kohde kuuluu.

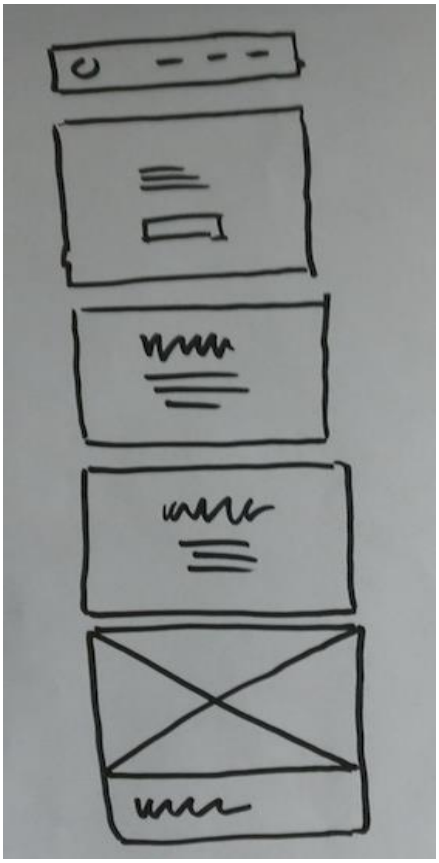


Kuvio 5. Neuroverkon perusperiaate (Mallick 2017)

6 Case study: Käyttöliittymähahmotelmien muuttaminen verkkosivuksi

6.1 Aloitus

Asiakkaan kanssa ollaan pitämässä palaveria uuden verkkosivun tiimoilta. Valmiiseen tuote-ekosysteemiin halutaan lisätä uusi kampanjasivusto. Kuvitteellisen verkkosivun luonnostelma on piirretty taululle (ks. kuvio 6). Luonnos halutaan muuttaa mahdollisimman nopeasti verkkosivuksi, jotta asettelu voidaan viimeistellä ja miettiä sisältöä. Jotta tämä onnistui, täytyy luonnoksessa esiintyvät komponentit löytää ja luokitella. Vasta tämän jälkeen voidaan koota verkkosivu.

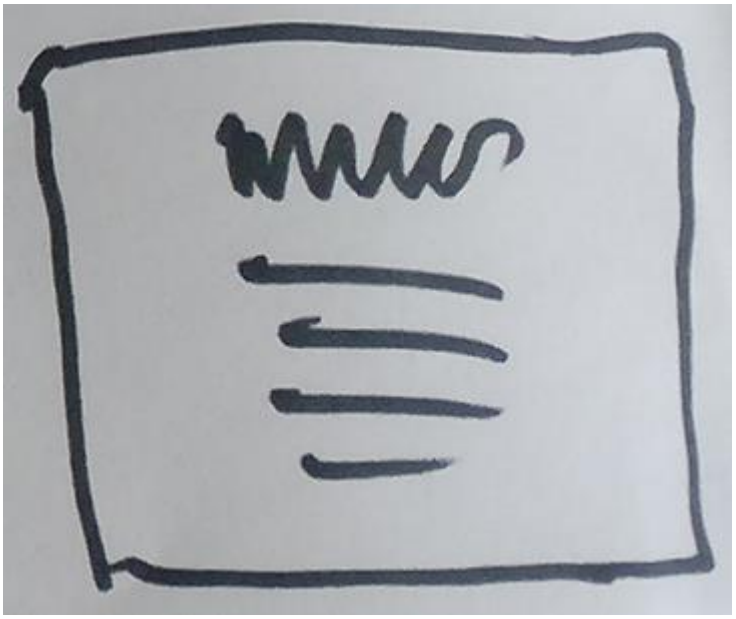


Kuvio 6. Luonnos kuvitteellisesta verkkosivusta

6.2 Aineiston luominen

Ennen varsinaisen ohjelmoimisen aloittamista on kuitenkin kerättävä aineistoa, jolla luokittelija voidaan kouluttaa. Luokittelijana käytetään uudelleen koulutettua neuroverkkoa. Mahdollisia käyttöliittymäkomponentteja voi olla käytännössä loputtomasti. Mahdollisuuksia täytyi kuitenkin rajoittaa, sillä jokaisen esiintymän opettaminen neuroverkolle ei ole käytännössä mahdollista.

Aineistoon valikoitui neljä eri luokkaa, joista jokainen edustaa yhtä käyttöliittymän rakennuspalikkaa (ks. kuvio 7). Nämä neljä luokkaa valikoitui siitä syystä, että niiden avulla voidaan luoda perustarpeet täyttävä käyttöliittymä. Sen lisäksi niiden piirtämien on yksinkertaista.

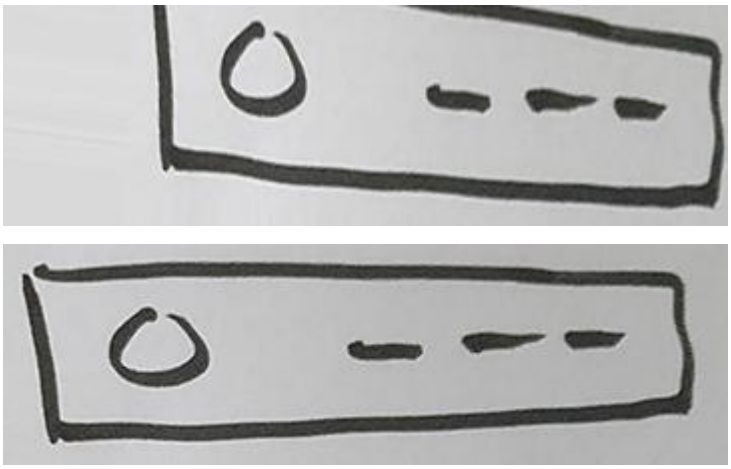


Kuvio 7. Yksi aineiston luokista: sisältölohko

Sen sijaan, että jokainen kuva olisi piirretty käsin, luotiin ns. synteettistä aineistoa. Tämä tarkoittaa, että jokaisesta luokasta piirrettiin ensin 15 kuvaa, jonka jälkeen Keras-koneoppimiskirjastosta löytyvällä Python-koodilla luotiin jokaisesta kuvasta 20 hieman erilaista kuvaa. Synteettisen aineiston luonti toteutettiin seuraavalla ohjelmointikoodilla:

```
datagen = ImageDataGenerator(
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True,
  fill_mode='nearest')
    …
for img in img_array:
  if gfile.Exists(img):
    loaded_img = load_img(img)
      x = img_to_array(loaded_img)
      x = x.reshape((1,) + x.shape)
      i = 0
      for batch in datagen.flow(x, batch_size=1,
                save_to_dir=output_dir,
                save_prefix=output_prefix,
                save_format='jpg'):
          i += 1
          if i > 20:
            break
```

Kuvan mittasuhteita vääristettiin, suurennettiin ja käänneltiin (ks. kuvio 8), jotta saatiin paljon erilaisia kuvia. Luodut kuvaobjektit tallennettiin tämän jälkeen JPG-tiedostoiksi. Ainoa rajoituksena oli komponenttien tunnistettavuus, joka piti ottaa huomioon kuvia muokatessa. Lopputuloksena saatiin aineisto, jossa oli 315 kuvaa jokaisesta komponentista eli yhteensä 1260 kuvaa.



Kuvio 8. Muokattu ja alkuperäinen kuva

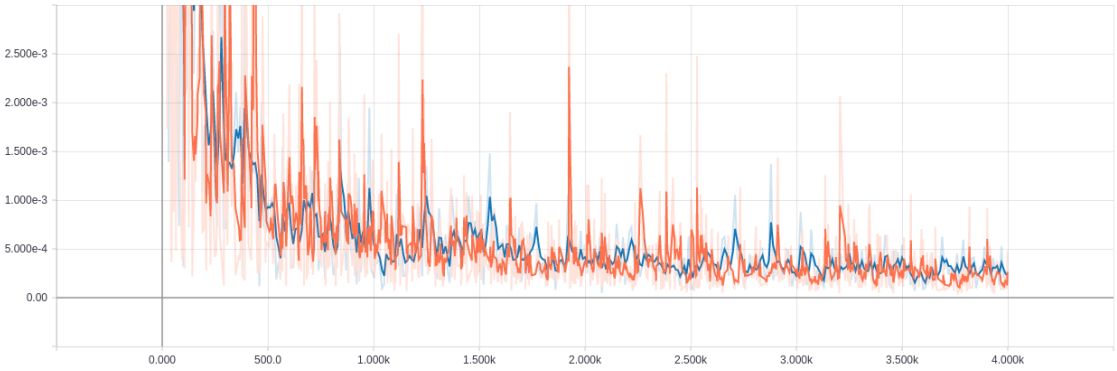
6.3 Luokittelijan kouluttaminen

Kouluttaminen voidaan aloittaa, kun aineisto on luotu. Verrattain pieni aineisto kuitenkin rajoitti luokittelijaksi sopivan mallin valintaa. Alusta asti opetettu neuroverkkoon pohjautuva malli vaatisi useita satoja tuhansia kuvia, jotta sen tarkkuus olisi riittävällä tasolla.

Tensorflow-kirjasto tarjosi kuitenkin mahdollisuuden käyttää siirto-oppimista valmiiksi koulutetun mallin uudelleen kouluttamiseen. Tensorflow-kirjastosta löytyi Python-koodi, jolla voitiin kouluttaa malli tunnistamaan uusia luokkia. Aikaisemmin luotu synteettinen aineisto jaettiin kansioihin ja kansion nimeksi annettiin luokan nimi. Kansiot koottiin Dataset nimisen kansion alle ja koulutus käynnistettiin komentoriviltä käyttäen komentoa:

```
python retrain.py --image_dir=Dataset/
```

Oletusarvoisesti käytettiin Inception V3 -arkkitehtuuriin pohjautuvaa neuroverkkoa. Koulutus kesti noin kymmenen minuuttia. Lopputuloksena saatu malli teki virheellisiä ennusteita ja oli liian hidas käytettäväksi reaaliaikaisessa sovelluksessa. Kuviossa 9 on esitetty oletusarvoilla koulutetun tunnistimen häviöfunktion tulokset sovitettuna iteraatioiden määrään. Kuvaajassa esiintyy paljon kohinaa, mikä viittaa liian pieneen koulutuserään. Oletusarvoisesti jokaisessa koulutusvaiheessa tunnistettavien kuvien määrä on 100.



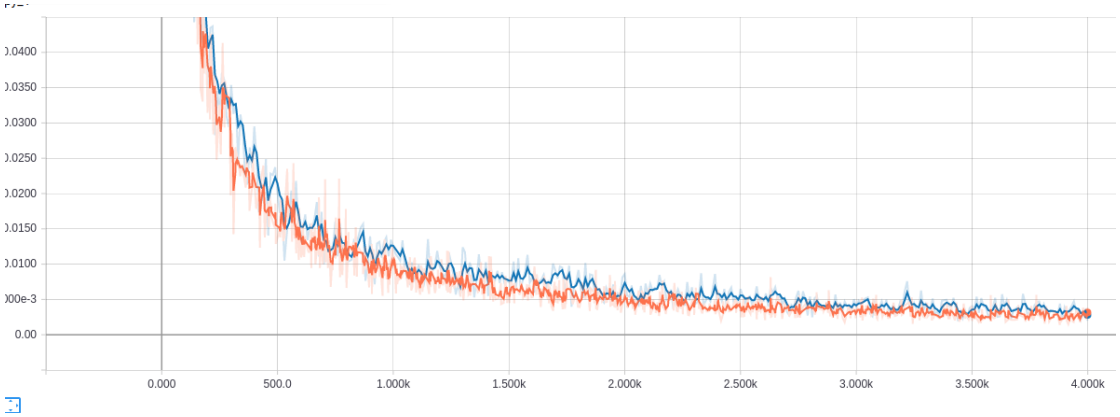
Kuvio 9. Häviöfunktio sijoitettuna koulutuskierrosten määrään

Yksi vaihtoehto oli muuttaa satunnaisten kuvien valoisuutta sekä kokoa parempien tulosten saavuttamiseksi. Myös vaihtamalla arkkitehtuuria Inception V3:sta Mobilenettiin, saavutettiin huomattavia nopeuseroja tunnistamisessa. Tämän lisäksi koulutuksessa annettiin parametrit, jotka lisäsivät skaalausta ja kirkkautta 20 prosenttiinkuvista. Myös oppimistahtia pudotettiin alkuperäisestä 0.01:stä 0.001:teen. Lopullinen malli koulutettiin seuraavalla komennolla:

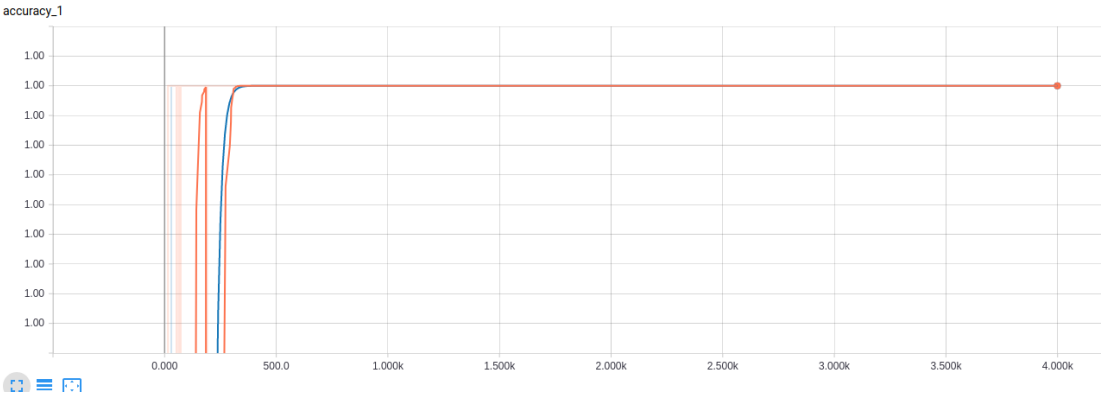
```
python retrain.py --architecture=mobilenet_1.0_224
--image_dir=Dataset/ --random-scale=20
--random_brightness=20 --learning_rate=0.001
```

Lopputuloksena saatu malli tunnisti huomattavasti paremmin vaikeammat ja epäselvemmät kuvat oikein (ks. Kuvio 10). Kuviossa 11 on esitettynä mallin tarkkuus, joka oli lähes 100% koko koulutuksen ajan.

Koulutus toteutettiin käyttämällä Nvidia GTX 1080 -näytönohjainta ja koulutuksen kesto oli noin yhden tunnin. Koulutuksen jälkeen malli tallennettiin PB-tiedostoon ja opitut luokat tekstitiedostoon.



Kuvio 10. Usean iteraation tuloksena saatu häviöfunktion kuvaaja



Kuvio 11. Mallin tarkkuus

6.4 Komponenttien rajaaminen

Ennen kuin komponentit voidaan luokitella, ne pitää irrottaa kuvasta. Kuvan koko tarkistetaan ensin. Mikäli se on pysty- tai leveyssuunnassa suurempi kuin 600 pikseliä, muutetaan sen kokoa säilyttämällä kuitenkin sen kuvasuhde. Kuva muutetaan seuraavaksi mustavalkoiseksi, jotta tunnistamisen helpottuisi.

Kuvasta löytyviä raja-arvoja korostetaan. Mikäli pikselin voimakkuus on suurempi kuin 130, asetetaan sen arvoksi täysin valkoinen eli 255. Mikäli taas pikselin voimakkuus on alle 130 annetaan sen arvoksi täysin musta eli 0. Käsiteltävästä kuvasta tulee täten kaksivärinen ja se toteutettiin seuraavalla ohjelmointikoodilla:

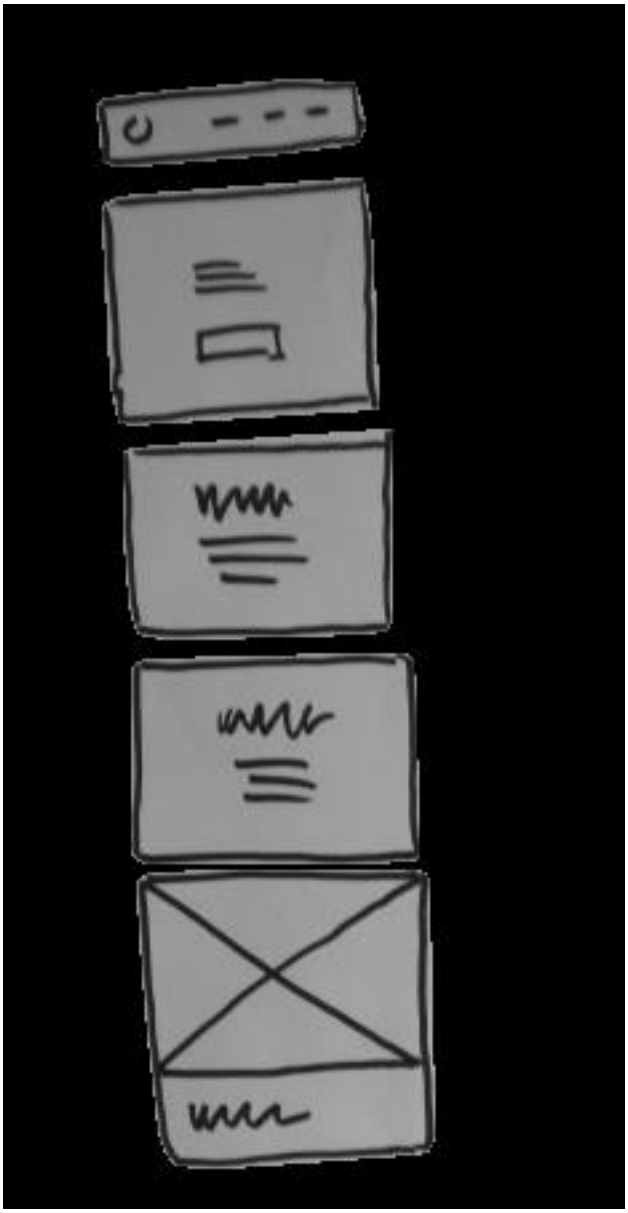
```
_,thresh = cv2.threshold(img, 130, 255,
cv2.THRESH_BINARY)
```

Komponenttien reunojen etsimiseen käytetään OpenCV-kirjastosta löytyvää MSER-metodia. MSER-metodi löytää todella monta erilaista kuviota, joiden perusteella ei voida tunnistaa komponentteja. Ensin MSER-algoritmi alustetaan ja tämän jälkeen käsiteltävästä kuvasta etsitään reunoja:

```
mser = cv2.MSER_create()
regions = mser.detectRegions(thresh)
```

Täten löydetyt alueet yhdistetään suuremmiksi kokonaisuuksiksi. Lähekkäin olevat alueet yhdistettiin, mikäli ne ovat pystysuunnassa vähintään 20 pikselin tai vaakasuunnassa vähintään 100 pikselin päässä toisistaan. Lopputuloksena saatiin usean sadan alueen sijaan muutama yhtenäinen alue, jotka tallennettiin erillisiin kuvatiedostoihin (ks. kuvio 12).

Käyttäjälle annetaan käyttöliittymässä mahdollisuus tarkastella tunnistettuja komponentteja ja poistaa ei-toivottuja komponentteja. Tämän vaiheen jälkeen varsinainen käyttöliittymä voidaan luoda.



Kuvio 12. Tunnistetut komponentit rajattuina

6.5 Komponenttien luokittelu

Tunnistamiseen käytettiin uudelleen koulutettua neuroverkkoa, joka oli tallennettuna PB-tiedostoon. Tiedosto sisälsi mallin painoarvot sekä rakenteen. Tämä tiedosto ladattiin muistiin ja sen sisältö luettiin seuraavalla tavalla:

```
def load_graph(model_file):
  graph = tf.Graph()
  graph_def = tf.GraphDef()
  with open(model_file, "rb") as f:
    graph_def.ParseFromString(f.read())
```

```
    with graph.as_default():
      tf.import_graph_def(graph_def)
    return graph
```

Kuva syötettiin tämän jälkeen mallille ja vastauksena saatiin luokan nimi, jolla on korkein todennäköisyys prosentteina.

6.6 Mallikirjaston luominen

Mallikirjaston luomiseen käytettiin Storybook-kirjastoa sekä Reactia. Reactin avulla kirjoitettiin komponentit käyttämällä JSX-kieltä. Yksittäinen komponentti koostuu kolmesta tiedostosta, joista ensimmäisessä tiedostossa määritettään komponentin rakenne. Seuraavassa määritettään tyylit, eli värit ja elementtien koot yms. Kolmannessa tiedostossa määritettään Storybookia varten tarvittavat tiedot. Näihin lukeutuu elementissä näytettävä sisältö ja mahdolliset muut muuttujat. Näitä tietoja käytetään vain silloin, kun elementtiä tarkastellaan mallikirjastossa ja ne eivät vaikuta sovelluksessa näytettävään komponenttiin.

Jokaisesta piirretystä luokasta luotiin oma komponentti. Komponentit luotiin siten, että niiden ulkoasua on helppoa muuttaa sille annettavilla parametreilla. Muutettaviin asioihin kuului esimerkiksi teksti, toiminnot ja taustakuva. Storybookin avulla jokaisen komponentin eri tilojen tarkastelu ja kehitys oli helppoa.

6.7 Käyttöliittymän luominen

Komponenttien tunnistus ja käyttöliittymän luominen aloitetaan lataamalla kuva Flaskilla tehdyllä sovelluksella. Sovellus oli käytännössä yksinkertainen verkkosivu minimaalisilla toiminnoilla. Käyttöliittymä vastasi kuvan lähettämisestä palvelimelle, jossa se käsiteltiin luvussa 6.4 kuvatulla tavalla kutsumalla seuraavan ohjelmointikoodin mukaista funktiota:

```
filenames = trace(file_name_and_path,
   file_name,
   UPLOAD_FOLDER)
```

Tämän vaiheen jälkeen kuvat syötettiin yksi kerrallaan luokittelijalle ja ennustukset, joiden varmuus on suurempi kuin 50%, palautettiin takaisin käyttöliittymään:

```
for file in filenames:
  predictions, results, labels =
  predict(UPLOAD_FOLDER + '/' + file)
    for i in predictions:
      if round(results[i]*100,2) > 50.00:
        output.append({
          "file":str('upload/' + file),
          "label":str(labels[i]),
          "confidence":str(round(results[i]*100,2
       ))})
```

Löydetyistä kuvista voitiin nyt valita halutut komponentit. Tämä vaihe lisättiin tunnistimen etsimisen aikana virheellisesti tunnistettujen komponenttien takia. Usein kuvissa saattaa esiintyä muita kokonaisuuksia, jotka saatetaan virheellisesti poimia ja tunnistaa. Parantamalla koko komponenttien rajaamis- ja tunnistamisprosessia voitaisiin tämä vaihe todennäköisesti poistaa kokonaan. Kun halutut kuvat oli valittu, lähetetään valinnat takaisin palvelimelle, jossa ne tallennetaan JSON-tiedostoon.

Varsinaisen käyttöliittymän luomiseen käytettiin React-sovellusta, jolle annetaan JSON-tiedostossa tunnistetut komponentit. JSON-tiedostossa olevien komponenttien nimien perusteella haetaan Storybookin luomasta komponenttikirjastosta halutut komponentit, jotka renderöitiin ruudulle siinä järjestyksessä, kuin ne luettiin käyttäen seuraavaa switch-case-ehtolausetta:

```
import {
  Header, Slide, Button, ImageWithText, Hero
} from '../lib/component-library'
                  …
const Home = () => {
  return (
    <div>
      {Object.keys(file).length !== 0 &&
    file.map(component => {
      switch(component.label) {
        case 'navigation':
          return (
            <div style={{background: 'black'}}>
              <Header />
            </div>
          )
        case 'textblock':
          return ( <Slide
             title="This is a title"
```

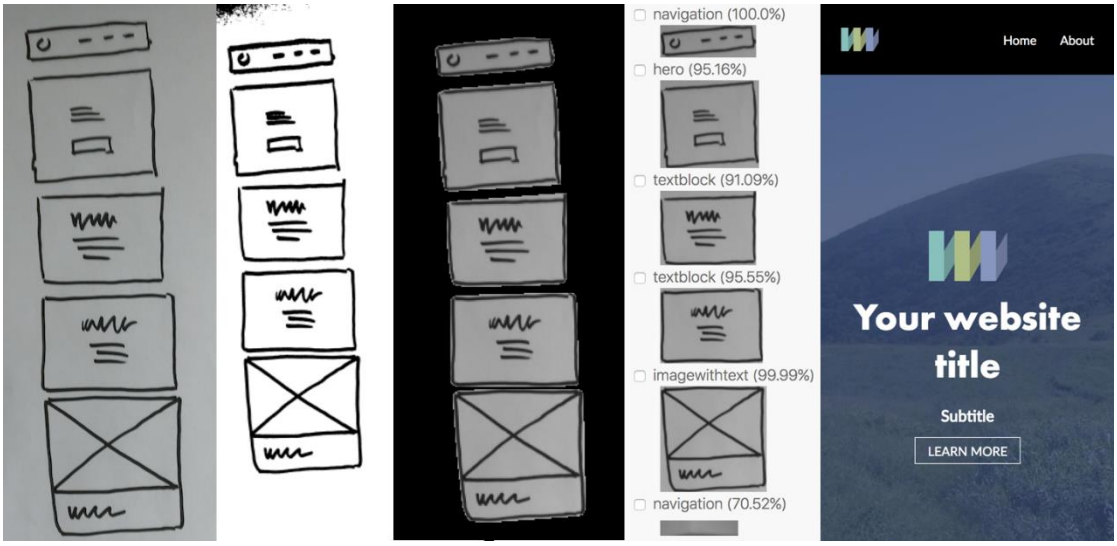
```
    bodyText="This is some text"
/>)
    …
```

6.8 Lopputulos

Kaikkien näiden vaiheiden jälkeen saadaan lopputuotteena verkkosivu, josta puuttui vain toiminnallisuus ja oikea sisältö. Tähän saatuun käyttöliittymään varsinaisen toiminnallisuuden lisääminen on haastavampaa kuin tapauksessa, jossa käyttöliittymä olisi koodattu käsin. Suurin syy tähän on, että luotuihin komponentteihin ei assosioida minkäänlaista tunnistavaa tekijää, jonka avulla niihin voitaisiin viitata.

Tehdyssä prototyypissä yhdistettiin perinteisiä kuvantunnistusmenetelmiä sekä uudempaa koneoppimiseen perustuvaa teknologiaa. MSER-algoritmia on yleisesti käytetty tekstin tunnistamiseen, mutta se soveltui myös hyvin piirrettyjen neliöiden ja pallojen tunnistamiseen.

Käsin kirjoitettu teksti ja käyttöliittymäkomponenttien rautalankamallit eivät eronneet toisistaan merkittävästi, minkä takia MSER:n hyödyntäminen oli järkevää. Varsinaisen luokittelun suorittava neuroverkko toteutettiin ensimmäisenä ja MSER:n yhdistäminen siten, että se toimi yhdessä neuroverkon kanssa oli helppoa. Kuviossa 13 on esitetty alusta loppuun polku, jossa kuvasta luotiin käyttöliittymä.



Kuvio 13. Lopullinen polku kuvasta verkkosivuksi

7 Tulokset

Mikäli käytössä on kattava komponenttikirjasto, voidaan uudelle tuotteelle suunnitella nopeasti lähes valmiit näkymät yhden palaverin aikana yksinkertaisesti piirtämällä se taululle tai paperille. Lopputulos on parempi, sillä suunnittelutilanteessa on jo saatu realistisempi kuva siitä, miltä lopullinen tuote tulee todennäköisesti näyttämään. Tämä helpottaisi myös sisällön suunnittelua, sillä on olemassa konteksti, jossa se esiintyisi.

Mikäli jokin yritys haluaa tällaisen järjestelmän käyttöön, täytyy kyseisellä yrityksellä olla komponenttikirjasto. Halutuista komponenttikirjaston kohteista täytyy tehdä käsin piirretty aineisto, jolla kuvantunnistin voidaan kouluttaa.

Tehokkaan kuvantunnistimen rakentaminen ilman kattavaa ainestoa on erittäin vaikeaa, ja kaikkien komponenttien piirtäminen useaan sataan kertaan ei ole todennäköisesti varteen otettava vaihtoehto. Siirto-oppimista hyödyntämällä voitaisiin kuitenkin toteuttaa malli, joka tunnistaisi komponentit konseptin testaamiseen riittävällä tarkkuudella pienestä aineistosta huolimatta.

Hyvin määritelty komponenttikirjasto on itsessään erittäin tehokas työkalu käyttöliittymien luomiseen. Luomisprosessin automatisoiminen ei täten tarjoaisi merkittävää hyötyä kuin tapauksessa, jossa komponenttikirjastossa on useita kymmeniä tai satoja komponentteja. Monen komponentin piirtäminen rautalankamallina kuitenkin muodostaa oman ongelmansa, sillä useat komponentit todennäköisesti muistuttaisivat toisiaan. Näin ollen myös luokittelijan olisi vaikea erottaa komponentteja toisistaan.

Tämän projektin lopputuloksena saatu prototyyppi todistaa, että tällaisen tuotteen toteuttaminen on mahdollista, mutta myös erittäin vaikeaa ja aikaa vievää. Mikäli järjestelmä saadaan kuitenkin pystytettyä, se voi potentiaalisesti helpottaa uusien tuotteiden lisäämistä jo olemassa olevaan ekosysteemiin. Myös asiakkaan kanssa vietetty aika saataisiin mahdollisimman tehokkaasti käytettyä hyödyksi.

8 Pohdinta

Projektin alussa työn varsinaisesta laajuudesta ja vaikeudesta ei ollut selvää kuvaa. Koneoppimiseen perehtyminen vei huomattavasti enemmän aikaa kuin olin alun perin ajatellut. Usean erilaisen kuvantunnistusmenetelmän kokeilemisen jälkeen oli ilmiselvää, ettei aineistoa ollut riittävästi ja se aineisto, joka oli saatavilla, ei ollut tarpeeksi laadukasta. Koneoppimisen osalta aineistoon pätee yksi sääntö: roskaa sisään, roskaa ulos. Tämä tarkoittaa sitä, että huonolla aineistolla saa vain huonoja tuloksia.

Alkuperäisessä aineistossa oli kymmenen luokkaa ja jokaisessa kymmenen käsin piirrettyä kuvaa. Kuvissa esiintyvät komponentit olivat liian pieniä kokonaisuuksia ja näyttivät rautalankamalleina esitettyinä liian samanlaisilta, jotta ne saataisiin tunnistettua oikein. Myös aineiston rajallinen määrä vaikeutti jo ennestään vaikeaa tunnistamista.

Riittävän aineiston kerääminen ei kuitenkaan ollut suoraviivaista. Käsin piirtäminen ei ollut järkevä vaihtoehto sen vaatiman ajallisien resurssien vuoksi ja synteettistä aineistoa ei voitu luoda loputtomasti. Ratkaisuna tähän oli piirrettävien komponenttiluokkien vähentäminen. Näistä käsin piirretyistä komponenteista voitiin luoda riittävä määrä synteettistä aineistoa, jotta luokittelija saataisi koulutettua.

Toimivan luokittelijan jälkeen ongelmaksi osoittautuivat kuvat, joissa komponentit ovat liian lähellä toisiaan tai heikosti rajattuja. Komponenttien rajaaminen ei onnistunut ja luokittelijalle päätyneet kuvat olivat liian epäselviä, jotta ne voitaisiin tunnistaa oikein. Tämä ongelma johtui MSER-algoritmin löytämien alueiden yhdistämisessä tapahtuvasta virheestä. Mikäli komponenttien yhdistämistä halutaan rajoittaa siten, että vain todella lähellä olevat alueet yhdistetään, on mahdollista, että koko komponentti ei päädy samaan ryhmään. Jäljelle jäävä osa komponentista voi päätyä osaksi toista ryhmää tai päätyä sellaisenaan tunnistimelle.

MSER-algoritmin korvaaminen vaihtoehtoisella OpenCV-kirjastosta löytyvällä reunojentunnistusalgoritmilla voisi ratkaista vaikeasti tunnistettavien komponenttien rikkoutumisen liian pieniin osiin, jolloin luokittelijalle päätyvien kuvien laatu olisi parempi.

Koneoppimisen tarve tämänkaltaisessa tuotteessa on lähestymistavasta riippuen joko todella suuri tai pieni. Mikäli kohteiden havainnointi ja luokittelu tehdään esimerkiksi Single Shot Multibox -tunnistimella, pitää aineistoa olla paljon. Mikäli taas havainnointi halutaan toteuttaa esimerkiksi OpenCV-kirjastoa hyödyntäen ja luokitella neuroverkolla, on tarvittavan aineiston määrä verrattain vähäinen.

Tämän hetkinen työnkulku piirtämisestä verkkosivuun on tarpeettoman pitkä ja tulevaisuudessa ongelmaan voitaisiin hakea ratkaisua esimerkiksi videokameraa hyödyntämällä. Tällöin kohteiden havainnointi voitaisiin toteuttaa reaaliajassa ja erilaisten asetelmien kokeileminen olisi vaivattomampaa.

Luotu verkkosivu tarvitsee kuitenkin sisältöä ennen kuin se voidaan julkaista. Verkosta löytyy tällä hetkellä useita erillisiä sisällönhallintapalveluita, joiden hyödyntäminen olisi järkevää. Tällöin sivun luonnin jälkeen voidaan jo alkaa miettiä sivun sisältöä ja saada suunnittelupalaverin aikana lähes valmis verkkosivu.

Toinen vaihtoehto olisi käyttää optista tekstintunnistusta ja tunnistaa komponenttien sisälle kirjoitettu teksti ja tulostaa se suoraan käyttöliittymään. Tämä kuitenkin vaikeuttaisi luokittelijan koulutusta, sillä käsin kirjoitettu teksti voi vääristää tunnistettavaa komponenttia.

Loppujen lopuksi prototyypin rakentaminen onnistui ja sen avulla saatiin todistettua, että se voi tulevaisuudessa olla hyödyllinen osa uusien tuotteiden suunnittelua ja toteutusta.

Lähteet

About. N.d. OpenCV-kirjaston kuvaus. Viitattu 10.4.2017. https://opencv.org/about.html

About Tensorflow. N.d. Tensorflow-kirjaton kuvaus. Viitattu 19.2.2018. https://www.tensorflow.org/

Beltramelli, T. 2017. Teaching Machines to Understand User Interfaces. Viitattu 4.3.2018. https://hackernoon.com/teaching-machines-to-understand-user-interfaces-5a0cdeb4d579

Brownlee, J. 2013. A Tour of Machine Learning Algorithms. Viitattu 10.4.2018. https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Del Bimbo, A. N.d. Region detectors. Viitattu 15.5.2018. http://www.micc.unifi.it/delbimbo/wp-content/uploads/2011/03/slide_corso/A34%20MSER.pdf

Flask web development, one drop at a time. N.d. Flask-sovelluskehyksen kuvaus. Viitattu 19.2.2018. http://flask.pocoo.org/

Forson, E. 2017. Understanding SSD MultiBox – Real-Time Object Detection In Deep Learning. Viitattu 24.5.2018. https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

GPU-Accelerated Caffe. N.d. Viitattu 21.05.2018. https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/caffe/

How to Retrain an Image Classifier for New Categories. 2018. Artikkeli Tensorflow-kirjaston verkkosivuilla. Viitattu 5.4.2018. https://www.tensorflow.org/tutorials/image_retraining

Image Recognition. 2018. Artikkeli Tensorflow'n verkkosivuilla. Viitattu 31.1.2018. https://www.tensorflow.org/tutorials/image_recognition

Introducing JSX. N.d. JSX-kielen kuvaus. Viitattu 24.5.2018. https://reactjs.org/docs/introducing-jsx.html

Introduction. N.d. Storybook-kirjaston kuvaus. Viitattu 19.2.2018. https://storybook.js.org/basics/introduction/

Kangasniemi, T. 2017. Neuroverkko oppii vaikka Väinämöiseksi. Artikkeli Dagmarin verkkosivuilla. Viitattu 7.2.2018. https://www.dagmar.fi/analytiikka/neuroverkko-oppii-vaikka-vainamoiseksi/

Kholmatova, A. 2017. Design Systems - A practical guide to creating design language for digital products. Smashing Media AG.

Koistinen, P. 2002. Tilastollinen hahmontunnistus. Viitattu 20.5.2018. http://www.math.helsinki.fi/petrin/hahmo02/hahmo.pdf

Korpimies, A. 2017. Pilvipalvelijat yhteen - tähtäävät Euroopan ykköseksi. Artikkeli Kauppalehden verkkosivulla. Viitattu 21.05.2018. https://www.kauppalehti.fi/uutiset/pilvipalvelijat-yhteen---tahtaavat-euroopan-ykkoseksi/yD7Ye7WD

Kožnarová, Z. 2017. Manuscripts Classification Using Convolutional Deep Learning Networks. Viitattu 24.5.2018. https://dspace.cvut.cz/bitstream/handle/10467/72972/F3-BP-2017-Koznarova-Zuzana-ManuscriptsClassificationUsingConvolutionalDeepLearningNetworks.pdf

Lee, H., Grosse, R., Ranganath, R. & Ng, A. 2009. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. Viitattu 23.1.2018. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.802&rep=rep1&type=pdf

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C-Y. & Berg, A. 2015. SSD: Single Shot Multibox Detector. Viitattu 9.2.2018. https://arxiv.org/abs/1512.02325

Machine Learning Glossary. N.d. Viitattu 21.05.2018. https://developers.google.com/machine-learning/glossary/#m

Mallick, S. 2017. Neural Networks: A 30,000 Feet View for Beginners. Viitattu 24.5.2018. https://www.learnopencv.com/neural-networks-a-30000-feet-view-for-beginners/

Modern Machine Learning Algorithms: Strengths and Weaknesses. 2017. Artikkeli EliteDataSciencen verkkosivuilla. Viitattu 6.2.2018. https://elitedatascience.com/machine-learning-algorithms

React – A JavaScript library for building user interfaces. N.d. React-kirjaston kuvaus. Viitattu 24.5.2018. https://reactjs.org/

Sachan, A. 2017. Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN, Yolo, SSD. Viitattu 20.5.2018. http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

Sneha, M., Vinod, K. 2017. Canny Edge Detection and MSER Features For Text Matching. Viitattu. 20.5.2018. http://www.digitalxplore.org/up_proc/pdf/306-150026881933-36.pdf

Silver, D. 2016. Deep Reinforcement Learning. Viitattu 10.4.2018. https://deepmind.com/blog/deep-reinforcement-learning/

Silver, N. 2012. Signaali ja kohina: miksi monet ennusteet epäonnistuvat, mutta jotkin eivät. Helsinki: Terra Cognita.

The 5 Levels of Machine Learning Iteration. 2017. Artikkeli EliteDataSciencen verkkosivuilla. Viitattu 23.1.2018. https://elitedatascience.com/machine-learning-iteration#micro

Thompson, S. 2016. Using Transfer Learning to Classify Images with Tensorflow. Viitattu 1.2.2018. https://medium.com/@st553/using-transfer-learning-to-classify-images-with-tensorflow-b0f3142b9366

Torrey, L. & Shavlik, J. 2009. Transfer Learning. Viitattu 6.2.2018. http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf

Wallner, E. 2017. Deep Learning for Developers: Tools You Can Use to Code Neural Networks on Day 1. Viitattu 20.5.2018. https://medium.freecodecamp.org/deep-

learning-for-developers-tools-you-can-use-to-code-neural-networks-on-day-1-34c4435ae6b

Wallner, E. 2018. Turning Design Mockups Into Code With Deep Learning. Viitattu 15.5.2018. https://blog.floydhub.com/Turning-design-mockups-into-code-with-deep-learning/

Wilkins, B. N.d. Sketching Interfaces: Generating code from low fidelity wireframes. Viitattu 6.5.2018. https://airbnb.design/sketching-interfaces/