Tampereen yliopisto

Teemu Eerola

NEUROVERKOT SULAUTETUISSA JÄRJESTELMISSÄ

Kandidaatintyö
Tekniikan ja luonnontieteiden tiedekunta
Tarkastaja: Mikko Salmenperä
Joulukuu 2021

TIIVISTELMÄ

Teemu Eerola: Neuroverkot sulautetuissa järjestelmissä
Kandidaatintyö
Tampereen yliopisto
Teknisten tieteiden kandidaatin tutkinto-ohjelma
Joulukuu 2021

Keinotekoisten neuroverkkojen hyödyntäminen koneoppimisongelmien ratkaisussa on yleistynyt tutkimuksessa ja teollisuudessa nopeasti. Niiden avulla saavutettu parannus suorituskyvyssä verrattuna aikaisempiin metodeihin muun muassa konenäön, kielentunnistuksen ja realistisen datan generoinnissa halutaan saada käyttöön myös sovelluksissa, joissa laskentakapasiteetti on rajoittunut. Erityyppiset laitteiston, laskennan ja ohjelmistojen optimointimenetelmät ovat mahdollistaneet neuroverkkojen käytön sulautetuissa järjestelmissä.

Tämän työn tarkoituksena on tutkia erilaisiin tehtäviin tarkoitettuja neuroverkkorakenteita, neuroverkon opetus- ja päättelyvaiheisiin liittyvää laskentaa sekä neuroverkkojen käyttöä sulautetuissa järjestelmissä. Erityisesti keskitytään sulautettujen järjestelmien kannalta keskeiseen neuroverkon päättelyn optimointiin, mutta työssä käsitellään myös suunnittelu- ja opetusvaiheen ratkaisuiden vaikutusta laskentaan. Käytännön kokeiden avulla työssä mitataan neuroverkon arkkitehtuurin valinnan, päättelyn eräkoon ja ohjelmallisten optimointimenetelmien vaikutusta neuroverkon suorituskykyyn päättelyssä. Mittauksia varten toteutettiin kuvanluokittelutehtävään tarkoitetuille neuroverkkomalleille testipenkki, jonka avulla mitattiin mallien päättelyn tarkkuutta, nopeutta sekä käyttömuistin kulutusta. Mittaukset suoritettiin Nvidian Jetson Nano -järjestelmällä, jolla lisäksi tutkittiin laitteiston optimointia neuroverkkolaskentaan.

Työn tulokset osoittavat, että neuroverkkoja on mahdollista käyttää päättelyyn sulautetussa järjestelmässä tehokkaasti ja reaaliaikaisesti. Merkittävää suoritusnopeuden parannusta saavutetaan käyttämällä laitteistona laskennan rinnakkaisuuden mahdollistavia suorittimia, kuten grafiikkaprosessoria. Sopivan neuroverkkoarkkitehtuurin valinnalla saadaan suunnitteluvaiheessa haettua kompromissi vaaditun tarkkuuden ja päättelynopeuden välille. Laitteistosta saatava hyöty voidaan maksimoida käyttämällä laitteistolle sopivaa eräkokoa mahdollisimman suuren rinnakkaisuuden saavuttamiseksi. Lisäksi kvantifiointia eli laskennassa käytettävän tarkkuuden vähentämistä ja käytettävälle laitteistolle laskentaa optimoivan ohjelmallisen optimointityökalun TensorRT:n avulla päättely todettiin noin 5-kertaa nopeammaksi ilman tarkkuuden menettämistä.

Avainsanat: Neuroverkko, koneoppiminen, sulautettu järjestelmä, Jetson Nano, TensorRT

Kokeellisen osuuden lähdekoodit saatavilla: https://github.com/teerol/kandi.

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

CPU	engl. Central Processing Unit, tietokoneen keskusprosessori
GPU	engl. Graphical Processing Unit, tietokoneen grafiikkaprosessori
FPS	engl. Frames Per Second, prosessointinopeuden yksikkö, ruutua sekunnin aikana
SGD	engl. Stochastic Gradient Descend, todennäköisyyteen perustuva menetelmä funktion minimin etsimiseen
TF-TRT	TensorFlow-TensorRT, TensorFlow:n sisäänrakennettu TensorRT-rajapinta.

1. JOHDANTO

Nykyisessä tietoyhteiskunnassa dataa syntyy joka hetki valtavia määriä. Sanotaankin, että *data on nykyajan öljy*. Valtavan datamäärän käsittely ja jalostaminen hyödylliseen muotoon on yksi tärkeimmistä tekoälyn ja koneoppimisen sovelluskohteista. Esimerkiksi kuvia, ääntä tai muuta dataa, josta ihmisen on helppo tehdä merkittäviä havaintoja, on ollut kuitenkin vaikea tulkita tietokoneen avulla. Koneoppimisen osa-alue, biologisista aivoista esikuvansa saaneet syvät neuroverkot, ovat osoittaneet tehokkuutensa tietyn tyyppisten ongelmien ratkaisussa. Neuroverkkojen vahvuusalueita ovat useat epälineaarisuutta mallintavat ihmismäiset tehtävät, joita on ollut vaikea ratkaista perinteisillä koneoppimisen tai signaalinkäsittelyn menetelmillä [1, luku 18.7]. Esimerkkinä tallaisesta tehtävästä ovat hahmon tunnistaminen kuvasta, puhutun kielen tunnistaminen ja luonnollisen kielen, puheen tai kuvien tuottaminen.

Neuroverkot ovat hyvin data- ja laskentakapasiteetti-intensiivisiä sekä verkon opetus että päättelyvaiheissa [1, luku 18.7]. Kuitenkin joissain aikakriittisissä tai vahvaa yksityisyyttä vaativissa sovelluksissa, havainnoitua dataa ei voida lähettää analysointia varten tehokkaille keskustietokoneille. Neuroverkkoa on tällöin pystyttävä käyttämään usein heikomman laskentakapasiteetin omaavissa laitteissa lähellä paikkaa, jossa havainnot tehdään. [2] Lähellä havainnointipaikkaa laitteet ovat usein sulautettuja järjestelmiä, joissa laskenta- ja muistikapasiteetti on rajoitettu esimerkiksi rajoitetun virrankulutuksen, painon, koon tai kustannuksen vuoksi. Tulevaisuudessa neuroverkkojen saavuttama suorituskyky halutaan saada käyttöön yhä useammissa sovelluksissa, joka lisää tarvetta erityyppisille neuroverkkolaskentaan erikoistuneelle laitteistolle sekä laskennan optimoinnille.

Tämän työn tarkoituksena on tutkia erilaisiin tehtäviin tarkoitettuja neuroverkkorakenteita, neuroverkon opetus- ja päättelyvaiheisiin liittyvää laskentaa sekä neuroverkkojen käyttöä sulautetuissa järjestelmissä. Erityisesti keskitytään sulautettujen järjestelmien kannalta keskeiseen neuroverkkojen käyttövaiheen eli päättelyn optimointiin. Työ keskittyy aiheen perusteiden tutkimiseen ja rajoittuu vain yksinkertaisimpiin ja perusteellisimpiin rakenteisiin, opetusparadigmoihin ja optimointimenetelmiin.

Työn luvussa 2 käsitellään neuroverkkojen teoriaa sekä opetus- ja päättelyvaiheen laskentaa ja sulautettujen järjestelmien erityisvaatimuksia. Luvussa 3 käsitellään optimointimenetelmiä alkaen neuroverkon arkkitehtuurin suunnittelusta ja edeten opetuksen ja päättelyn optimointiin.

Luvussa 4 esitellään työn yhteydessä suoritetun kokeellisen osuuden toteutus ja tulokset. Kokeissa neuroverkon päättelyvaiheen optimoinnin menetelmiä tutkittiin käytännössä Nvidian Jetson Nano -järjestelmällä. Tutkittuja menetelmiä ovat neuroverkon arkkitehtuurin valinta, päättelyn eräkoon vaikutus ja ohjelmallisten optimointimenetelmien vaikutus käyttäen mittareina päättelyn tarkkuutta, nopeutta ja käyttömuistin kulutusta. Lisäksi tehtiin mittauksia Jetson Nanon laitteiston erilaisten käyttötapojen vaikutuksesta päättelyn nopeuteen. Työn lopussa tehdään yhteenveto teoriaosuuden keskeisistä havainnoista ja vertaillaan niitä kokeellisessa osuudessa saatuihin tuloksiin.

2. NEUROVERKOT

Keinotekoinen neuroverkko on matemaattinen päättelymalli, jonka rakenne on saanut vaikutteita ihmisaivoista. Tässä työssä neuroverkolla tarkoitetaan aina keinotekoista neuroverkkoa, jos toisin ei mainita. Ajatus käyttää neuroverkkoja älykkääseen päätöksentekoon sai alkunsa vuonna 1943 psykologi Warren McCulloughin ja matemaatikko Walter Pittsin yhteistyöstä [4]. Alun innostuksen jälkeen tutkimuksen määrä aiheen ympärillä hiipui, sillä muilla menetelmillä saatiin lupaavampia tuloksia [3, luku 3]. Vuonna 2012 Toronton yliopiston tekoälytutkijat Alex Krizhevsky, Ilya Sutskever ja Geoffrey E. Hinton julkaisivat uraauurtavan tutkimuksen [5], jossa he esittelivät 650 000 päättelintä ja 60 miljoonaa parametria sisältävän neuroverkon kuvantunnistukseen. Erona aikaisempiin yrityksiin luoda toimiva neuroverkkomalli oli suurempi määrä parametreja ja opetusdataa sekä tietokoneiden valtavasti kehittynyt laskentateho [6, s. 1].

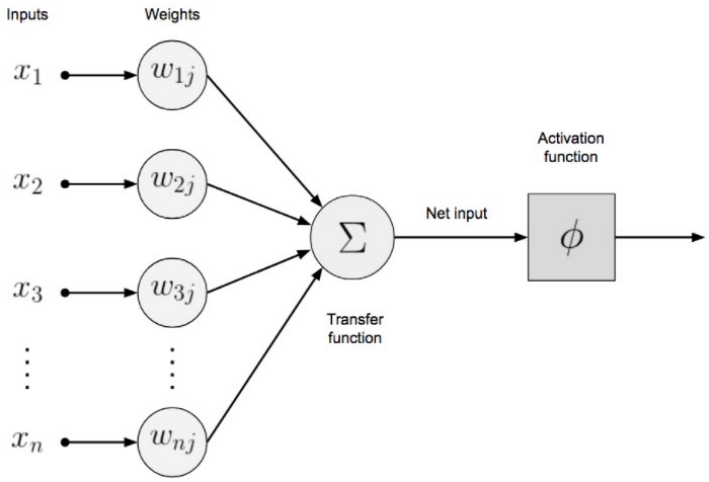
Nykyään neuroverkot ovat suuri kiinnostuksen kohde tutkimuksessa ja ala kehittyy valtavalla nopeudella. Niiden kyky mallintaa epälineaarisia prosesseja on johtanut moniin hyvin erityyppisiin käyttötarkoituksiin aina kvanttikemian mallintamisesta luonnollisen kielen prosessointiin ja osaksi autonomisten autojen ohjausjärjestelmiä [2][7]. Myös neuroverkkoihin pohjautuvien sovellusten kehittämiseen on luotu useita ohjelmistokirjastoja yleisimmille ohjelmointikielille. Esimerkiksi Googlen kehittämä avoimen lähdekoodin TensorFlow tarjoaa Python-rajapinnan neuroverkkojen rakentamiseen, opetukseen ja käyttöön [8].

Erona perinteisiin koneoppimisen menetelmiin neuroverkoilla ominaisuuksien etsiminen on voitu siirtää kokonaan neuroverkkojen omalle vastuulle. Perinteisesti ominaisuuksia on tunnistettu ihmisen luomien sääntöjen avulla. Neuroverkkojen tapauksessa ei aina edes saada selville, mitä ovat ne ominaisuudet, jotka neuroverkot yhdistävät oikeanlaiseen päätelmään. [3, luku 5] Usein neuroverkko on siis toiminnaltaan musta laatikko, jonka sisäistä toimintaa ei täysin ymmärretä.

2.1 Erityyppiset neuroverkkorakenteet

Seuraavissa alaluvuissa esitellään erityyppisiä neuroverkkorakenteita, niiden toimintaperiaatteita ja käyttötarkoituksia. Erityisesti tutkitaan eteenpäin kytkettyä ja konvoluutioneuroverkkoa, sillä niitä käytetään työn kokeellisessa osuudessa. Ensin syvennytään kaikille rakenteille yhteiselle ominaisuuteen päättelimeen.

Päättelin (engl. perceptron) on neuroverkon pienin yksittäinen osa. Se on matemaattinen operaattori, joka saa sisääntulona useita signaaleja ja antaa ulostulonaan yhden signaalin. Jokaiselle sisääntulosignaalille on oma painokerroin ja painotetut signaalit summataan. Tämän jälkeen seuraa aktivointifunktio, joka määrittää päättelimen ulostulon voimakkuuden painotettujen sisääntulosignaalien summan perusteella. Eri tilanteissa voidaan käyttää erilaisia aktivointifunktioita, kuten askelfunktiota, jatkuvaa askelfunktiota eli Sigmoidin funktiota ja lineaarista tai tasoitettua lineaarista aktivointia (engl. rectified linear, ReLu). Kuva 1 esittää päättelimen rakenteen *n*:lle sisääntulolle.



Kuva 1. *Päättelimen rakenne [3, luku 2]*

Matemaattisessa notaatiossa on käytännöllistä koota sisääntulosignaalit ja niitä vastaavat painot vektoreihin ***x*** ja ***w***, jolloin painotettujen sisääntulojen summa saadaan vektorien pistetulona. Päättelimen matemaattinen malli voidaan esittää muodossa

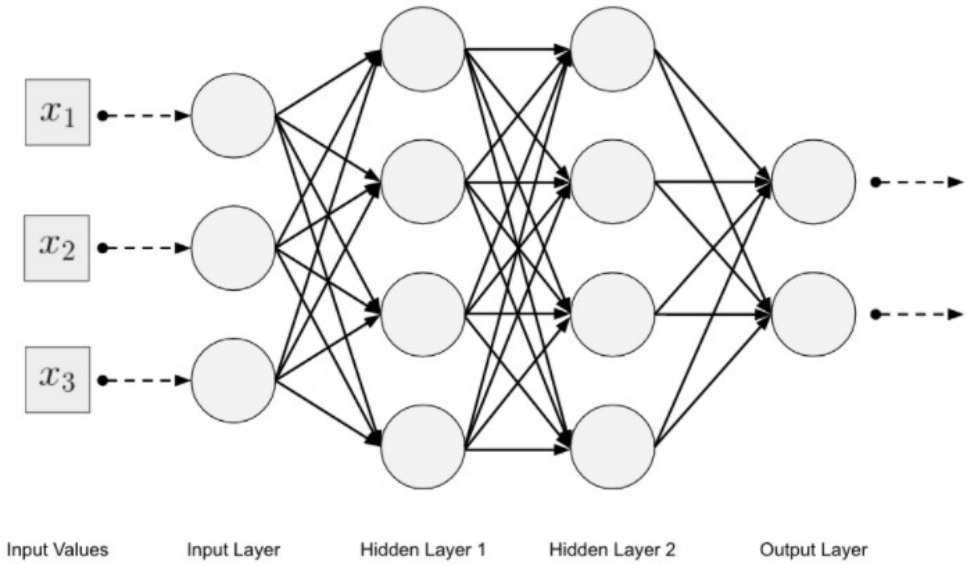
$$a = g(\boldsymbol{w} \cdot \boldsymbol{x} + b), \tag{1}$$

jossa *a* on ulostulosignaali, *g* on aktivointifunktio ja *b* mahdollinen vakiopoikkeama. [3, luku 2]

2.1.1 Eteenpäin kytketty neuroverkko

Usean päättelimen yhdistelmä muodostaa neuroverkon. Usein päättelimet kootaan kerroksiin, joissa jokaisen päättelimen ulostulo kytketään jokaiseen seuraavan kerroksen päättelimen sisääntuloon. Tällaista kerrosta kutsutaan täysin kytketyksi (engl. fully connected, FC) kerrokseksi. Samassa kerroksessa olevilla päättelimillä on hyvin usein sama aktivointifunktio, mutta niiden painot eri sisääntuloille vaihtelevat. Eri määrän päättelimiä omaavia kerroksia voidaan koota peräkkäin, jolloin syntyy neuroverkkoja. Kuvassa 2 esi-

tetyssä täysin kytketyssä neuroverkossa ympyrät esittävät yksittäistä päättelintä ja nuolet niiden välisiä yhteyksiä. Piilotetulla (engl. hidden) kerroksella tarkoitetaan kerrosta, joka ei ole verkon ensimmäinen tai viimeinen kerros. Eteenpäin kytketty neuroverkko tarkoittaa sellaista verkkoa, jossa informaatio virtaa verkon läpi eteenpäin ilman että edelliset sisääntulot vaikuttavat tulokseen. [1, s. 725]



Kuva 2. *Eteenpäin ja täysin kytketty neuroverkko kahdella piilotetulla kerroksella [3, luku 2].*

Opetettavien parametrien määrä täysin kytketyssä verkossa koostuu päättelimien painokertoimista. Painoja on yksittäisessä päättelimessä yhtä monta kuin edellisessä kerroksessa päättelimiä. Kerroksessa opetettavia parametreja on siis päättelimien lukumäärä kerrottuna edellisen kerroksen päättelimien lukumäärällä. Esimerkiksi kuvia datana käytettäessä verkon sisääntulona käytetään usein yhtä päättelintä kuvaamaan jokaisen kuvapisteen jokaista kolmea RGB-väriarvoa. Tällöin pelkästään sisääntulokerrokseen tulee 300 kertaa 200 pikselin kuvasta 300 x 200 x 3 = 180 000 päättelintä. Tähän useamman piilotetun kerroksen lisäämällä parametrien määrä kasvaa hyvin suureksi.

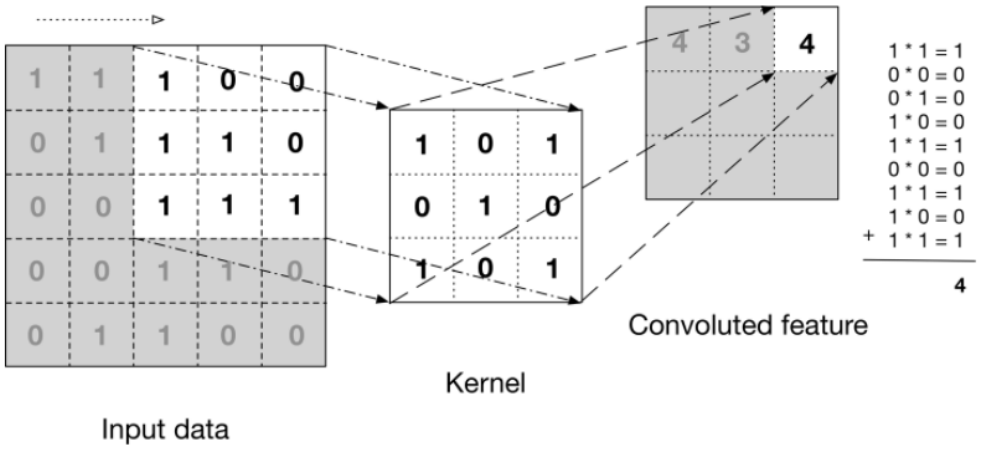
Täysin ja eteenpäin kytkettyjä verkkoja voidaan käyttää mallintamaan minkälaista dataa tahansa, kun kerrosten kokoa ja määrää sovitetaan sopivalla tavalla. Todella suuria verkkoja on laskennallisesti todella raskasta käyttää, ja siksi on kehitetty tehokkaampia menetelmiä löytämään ominaisuuksia datasta [9, luku 4].

2.1.2 Konvoluutioneuroverkko

Konvoluutioneuroverkko (engl. Convolutional Neural Network, CNN) on eteenpäin kytketty neuroverkko, joka perustuu konvoluutiokerroksiin. Näissä käytettävä matemaattinen operaattori konvoluutio on tarkoitettu kahden funktion yhdistämiseen. Neuroverkkojen kontekstissa data on usein diskreetti aikasarja tai diskreetti moniulotteinen matriisi kuten kuva. Sisääntulon ja painofunktion (engl. kernel tai filter) konvoluutio *S* pisteessä *i, j* saadaan kaksiulotteisessa tapauksessa laskettua kaavalla

$$S(i,j) = (\boldsymbol{I} * \boldsymbol{K})(i,j) = \sum_m \sum_n \boldsymbol{I}(i+m, j+n)\boldsymbol{K}(m,n), \quad (2)$$

jossa ***I*** on alkuperäinen matriisi, ***K*** painofunktio (myös matriisi) sekä *m* ja *n* painofunktion indeksejä. Konvoluutiota ***S*** kutsutaan aktivointi- tai ominaisuuskartaksi. [10, s. 327–329] Kuva 3 esittää konvoluution muodostumista 5 x 5 matriisin ja 3 x 3 painomatriisin välillä.



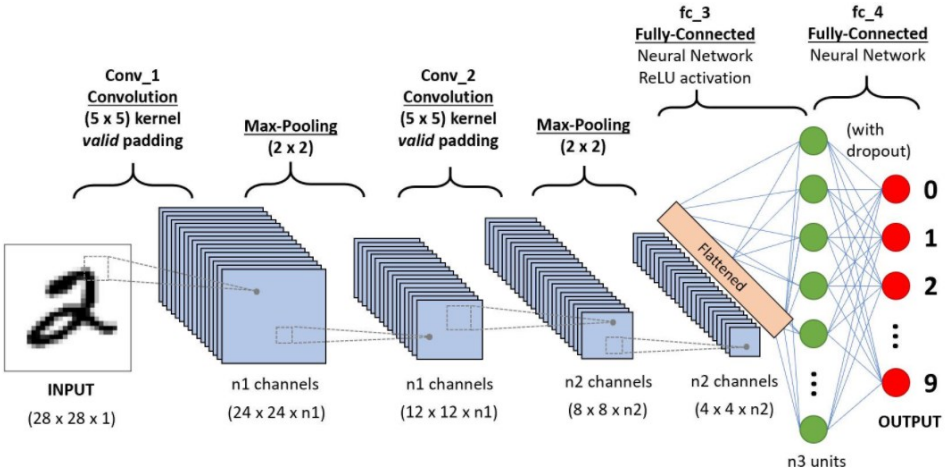
Kuva 3. *Konvoluutio kahden kaksiulotteisen neliömatriisin välillä [3, luku 4].*

Neuroverkkojen konvoluutiokerroksissa painofunktioilla on samat ulottuvuudet, mutta lähes aina pienemmät dimensiot kuin sisääntulofunktiolla, jolloin aktivaatiokartalla on samat tai hieman pienemmät dimensiot kuin sisääntulofunktiolla. Oikeanlaisilla painofunktioilla voidaan sisääntulosta erotella merkittäviä ominaisuuksia, kuten vertikaalisia tai horisontaalisia reunoja, kulmia ja alueita sekä näiden yhdistelmiä. [10, s. 330–331] Usein yhdessä konvoluutiokerroksessa käytetään useita eri painofunktioita erottelemaan datasta erityyppisiä ominaisuuksia. Näin ollen kerroksen ulostulo koostuu useasta aktivaatiokartasta, jotka ovat tuottaneet erilaiset painofunktiot samasta sisääntulosta. [3, luku 4]

Nykyisten syvien konvoluutioneuroverkkojen tapauksessa ominaisuuksien erottelu tapahtuu useassa peräkkäisessä konvoluutiokerroksessa. Tällöin konvoluutionkerroksen

sisääntulona toimivat joko annettu data tai edellisen konvoluutiokerroksen laskemat aktivointikartat. Tämä lisää ominaisuuksien abstraktiutta tasolle, josta on mahdollista erotella esimerkiksi kuvasta kasvot, jotka ovat usean ominaisuuden yhdistelmä.

Laskennan keventämiseksi syvissä konvoluutioneuroverkoissa käytetään konvoluutiokerrosten välissä alinäytteistyskerroksia (engl. pooling layer), joissa saatuja aktivointikarttojen kokoa pienennetään. Näissä kerroksissa voidaan käyttää erityyppisiä näytteistystekniikoita kuten keskiarvoistavaa- tai maksiminäytteistystä halutun tuloksen saamiseksi. Konvoluutio- ja alinäytteistyskerrosten jälkeen verkolle syötetystä datasta on ihannetapauksessa saatu eroteltua korkean abstraktion piirteitä. Kuvantunnistuksen tapauksessa nämä piirteet voidaan muuttaa vektoriksi ja syöttää yhdelle tai useammalle täysin kytketylle kerrokselle, jotka yhdistävät erilaiset ominaisuudet luokitteluksi. [3, luku 4]

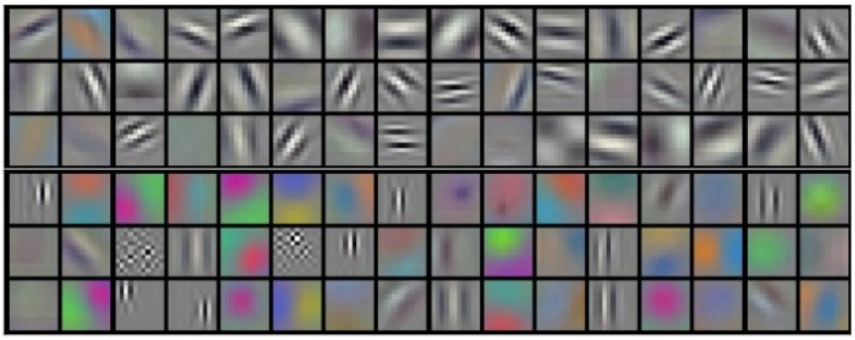


Kuva 4. *Kuvantunnistukseen suunnitellun syvän konvoluutioneuroverkon arkkitehtuuri [11].*

Kuvan 4 neuroverkko koostuu kahdesta konvoluutiokerroksesta, kahdesta alinäytteistyskerroksesta ja kahdesta täysin kytketystä kerroksesta. Sen arkkitehtuuri muistuttaa LeNet-5-mallia, joka on yksinkertainen konvoluutioneuroverkko käsin kirjoitettujen numeroiden tunnistamiseen [11].

Täysin kytkettyyn kerrokseen verrattuna konvoluutiokerrokset tarvitsevat huomattavasti vähemmän parametreja. Jos edellisen luvun tapaan tutkitaan 300 x 200 pikselin värikuvaa, jossa ensimmäisessä konvoluutiokerroksessa halutaan erotella 100 erilaista ominaisuutta käyttäen 11 x 11 x 3 painomatriisia, tarvitaan 36 300 parametria. Parametrien määrää voidaan vähentää edelleen käyttämällä jaettuja parametreja (engl. parameter sharing), jossa painomatriisissa voidaan syvyyssuunnassa käyttää asteittain samoja arvoja [10, s. 331–334].

Kuvassa 5 Krizhevskyn et al. [5] saamat 96 11 x 11 x 3 kokoista erilaista painofunktiota eli filtteriä kuvantunnistusneuroverkon ensimmäisessä konvoluutiokerroksessa. Osa filtereistä erottelee eri suuntaisia reunoja, osa erilaisia tekstuureja ja osa värejä. Vielä ensimmäisessä kerroksessa ominaisuudet eivät ole yhdistettävissä mihinkään lopputuloksena saatavaan luokkaan.



Kuva 5. *Opetetut painofunktiot Krizhevskyn et al. kuvantunnistusneuroverkon ensimmäisessä konvoluutiokerroksessa [5].*

Konvoluutioneuroverkot ovat tulleet erityisen tunnetuksi kyvystään prosessoida kuvia, mutta niitä voidaan käyttää myös yksiulotteisen datan kuten äänen tai tekstin analysointiin [10, s. 326]. Konenäkösovellukset ovat yksi nopeimmin konvoluutineuroverkot käyttöön ottanut ala [3]. Konvoluutineuroverkkojen toimivuus käytännön sovelluksissa on ollut yksi syy sille, miksi neuroverkkoja ympärille on keskittynyt paljon huomiota [3, luku 4].

2.1.3 Muut neuroverkkorakenteet

Takaisinkytketty neuroverkko (engl. Recurrent Neural Network, RNN) sisältää kerrosten välisiä takaisinkytkentöjä. Verkossa voi olla täysin kytkettyjä tai konvoluutiokerroksia kuten normaalissa eteenpäin kytketyssä neuroverkossa, mutta takaisinkytkennän ansioista ne kykenevät tuloksessa ottamaan huomioon myös edellisten sisääntulojen vaikutukset. Ulostulona voidaan verkon rakenteesta riippuen saada yksi tai useampi tulos jokaisella sisääntulolla tai yksi tulos useammasta peräkkäisestä sisääntulosta. Tämän ansioista takaisinkytketyt neuroverkot sopivat erityisesti ajallisten riippuvuuksien mallintamiseen ja generointiin. [3, luku 4]

Käytännön sovelluksia takaisinkytketyn neuroverkon käytöstä on esimerkiksi luonnollisen kielen tuottaminen tai keskusteluun osallistuminen, joissa edelliset valitut sanat tai virkkeet vaikuttavat siihen, mikä seuraava sana voi olla [3]. Kuvasta tehtävä sanallinen kuvaus on eräs tunnettu ongelma, jonka ratkaisussa yhdistyvät konvoluutiokerrokset ja takaisinkytketty rakenne [12].

Generatiivisilla neuroverkoilla tarkoitetaan neuroverkkoja, jotka kykenevät luomaan uutta dataa. Eräs tunnettu menetelmä tähän on autoenkoodaaja (engl. autoencoder), jossa sisääntulodatasta etsitään ominaisuuksia ja nämä ominaisuudet yritetään kopioida ulostuloon. Ominaisuudet erotellaan joko leveydeltään pienenevillä täysin kytketyillä kerroksilla tai konvoluutiokerroksilla pullokaulakerrokseen. Ulostulo saadaan pullonkaulakerroksesta levenevillä täysin kytketyillä tai dekonvoluutiokerroksilla. Käyttötarkoituksia autoenkoodaajalle ja muille generatiivisille verkoille ovat muun muassa kuvan keinotekoinen tarkennus, puuttuvien datapisteiden mallinnus tai kohinaisen datan siistiminen. [10, luku 14]

Vuonna 2014 Ian Goodfellow et al. [13] esittelivät generatiivisen kilpailevan verkoston (engl. Generative Adversarial Network, GAN). Siinä toinen generatiivinen eli dataa tuottava neuroverkko asetetaan kilpailemaan toista, erottelevaa neuroverkkoa vastaan. Erotteleva verkko saa sisääntulonaan dataa, jonka se yrittää luokitella kuuluvaksi alkuperäiseen harjoitusdataan eli aitoihin tai epäaitoihin generoituihin. Generatiivinen verkko puolestaan tuottaa satunnaisesta kohinasta dataa, joka syötetään erottelevalle verkolle. Lopputuloksena generatiivinen verkko oppii luomaan hyvin alkuperäistä harjoitusdataa vastaavaa materiaalia ja myös luokittelemaan datan aitouden. [13] Generatiivisia kilpailevia verkostoja voidaan käyttää esimerkiksi aidon oloisten kuvien, videoiden ja äänen tuottamiseen [3, luku 4].

2.2 Neuroverkon opetus

Yleisimmin neuroverkot opetetaan ohjatun oppimisen menetelmillä (engl. supervised learning), joissa verkolle esitetään opetusdataa [1, luku 18]. Oppimisen tavoitteena on minimoida neuroverkon antaman tuloksen ja halutun tuloksen välinen ero muuttamalla verkon parametreja. Syvät neuroverkot voivat oppimia myös muilla oppimisparadigmoilla kuten ohjaamatonta- tai vahvistusoppimista käyttäen. [3, luku 3] Seuraavissa kappaleissa esitellään neuroverkon ohjattuun oppimiseen tarvittavia metodeja ja oppimisen rajoitteita.

Halutun tuloksen ja neuroverkon tuottaman tuloksen mittaamiseen tarvitaan häviöfunktio (engl. loss function), joka kertoo kuinka ”kaukana” saatu tulos on halutusta. Erityyppisille päättelytehtäville on olemassa erilaisia häviöfunktiota, kuten esimerkiksi keskineliövirhe, logaritminen virhe ja ristientropiavirhe. Opetuksen tavoitteena on minimoida häviöfunktion arvo. [3, luku 2]. Jos neuroverkko esimerkiksi laskee jollekin datapisteelle arvon 0,9 ja harjoitusdatassa tämän datapisteen todellinen arvo on 1, saadaan näiden erotuksesta eräs arvo häviölle.

Yleinen tapa ratkaista optimointiongelma, kuten neuroverkon häviöfunktion minimointi, on käyttää funktion gradienttiin perustuvaa menetelmää. Gradientti kertoo, mihin suuntaan funktion arvot kasvavat voimakkaimmin, joten negatiivisen gradientin suunta kertoo voimakkaimman vähenemisen suunnan. Optimoinnin ideana on siirtyä negatiivisen gradientin suuntaa lyhyt sekä mahdollisesti lyhenevä askel ja toistaa tätä niin kauan kunnes löydetään minimikohta. Globaalin minimin löytämiseksi alkupisteitä voidaan valita satunnaisesti useita kappaleita ja valita optimoinnin jälkeen pienin tulos. [10, luku 4.3] Negatiivisen gradientin suuntaan otettavan askeleen pituus määritellään oppimisasteen (engl. learning rate) avulla.

Opetusdatasta saadun tuloksen ja todellisen tuloksen ero eli häviöfunktion arvo pystytään laskemaan vain neuroverkon viimeiselle kerrokselle. Syvissä neuroverkoissa kuitenkin myös piilotettujen kerroksien parametrit vaikuttavat viimeisen kerroksen tulokseen, mikä on gradienttia laskettaessa otettava huomioon. Gradientin laskemiseen syvissä verkoissa käytetään usein vastavirta-algoritmia (eng. back-propagation algorithm), jossa informaatio virtaa verkon läpi viimeisestä ensimmäiseen kerrokseen [10, s. 200].

Valitaan viimeisestä kerroksesta päättelin *i* ja sitä edeltävästä kerroksesta päättelin *j.* Niiden väliselle parametrille uusi arvo saadaan

$$W_{j,i} = W_{j,i} + \alpha\, a_j\, Err_i\, g'(in_j), \tag{3}$$

jossa $W_{j,i}$ on paino päättelimestä *j* päättelimeen *i*, α on oppimisaste, a_j *j* päättelimen aktivoinnin suuruus, Err_i virhe päättelimessä *i* ja $g'(in_j)$ päättelimen *j* aktivointifunktion derivaatan arvo sen sisääntulojen painotetulla summalla. Piilotettujen kerrosten painoille saadaan laskettua uusi arvo vastaavalla kaavalla korvaamalla termi Err_i vastavirta-algoritmin mukaisesti painotetulla summalla seuraavien kerrosten virheistä. [3, luku 2]

Kokonaisuudessaan neuroverkon opetusprosessi koostuu seuraavista vaiheista:

1. Alustetaan parametrit pienillä satunnaisluvuilla.
2. Syötetään opetusdata ja lasketaan häviöfunktion arvo.
3. Muutetaan verkon parametreja negatiivisen gradientin suuntaan oppimisasteen suhteessa kaavan 3 mukaisesti.
4. Toistetaan vaiheita 2–3 *N* kertaa.

Sama opetusdata näytetään verkolle siis useamman kerran, jotta häviöfunktion minimi löydetään. Yhtä opetusdatan näyttökertaa kutsutaan epookiksi (engl. epoch). Yllä olevassa opetusprosessissa voidaan siis sanoa olevan *N* epookkia.

Verkon parametrien määrä määrittää, kuinka paljon harjoitusdataa ja kuinka monta epookkia neuroverkon opetusprosessi vaatii. Näiden ominaisuuksien suhde on oleellisessa osassa verkon opetuksessa. Tavoitteena on opettaa neuroverkko niin, että se toimii hyvin harjoitusdataan kuulumattomalla testidatalla eli sen tulokset yleistyvät harjoitusdatan ulkopuolelle. Suuri epookkien määrä suhteessa verkon kokoon voi johtaa ylisovitukseen (engl. overfitting), jossa verkko alkaa muistaa harjoitusdatan ja suoriutuu sillä hyvällä tarkkuudella. Testidataa syötettäessä verkko ei kuitenkaan suoriudu hyvin. Vastaavasti jos verkon parametrimäärä on suuri verrattuna harjoitusdatan määrään, päädytään alisovitukseen (engl. underfitting) ja verkko suoriutuu huonosti sekä harjoitus- että testidatalla. [3, luku 1]

Ylisovituksen hallitsemiseen voidaan osaa harjoitusdatasta käyttää validointidatana eli testata jokaisen epookin jälkeen, miten verkko suoriutuu uudella datalla [10, s.119]. Tämä auttaa päättämään, milloin riittävä määrä epookkeja on saavutettu. Data lisäämisen metodit (engl. data augmentation) voivat auttaa alisovitukseen, koska niiden avulla opetusdatan määrää voidaan kasvattaa. Myös ylisovituksen todennäköisyys pienenee, sillä opetetussa datassa on enemmän variaatiota ja epookkeja tarvitaan vähemmän. [14] Pudotuskerrosten (engl. Dropout Layer) lisääminen vähentää ylisovituksen syntymistä. Nämä kerrokset konvoluutio- tai täysin kytkettyjen kerrosten välissä estävät satunnaisesti valittujen aktivointisignaalien kytkeytymisen seuraavaan kerrokseen [10, luku 7.12].

Myös opetusdatan laajuus ja laatu ovat erittäin olennaisia tekijöitä neuroverkon opetuksen onnistumisessa. Yleisesti ottaen dataa ei ole koskaan liikaa, kun verkon koko sovitetaan sopivaksi suhteessa opetusdatan määrään. Pattersonin et al. mukaan neuroverkon opetukseen tarvitaan vähintään 5000 luokiteltua esimerkkiä, mutta tarvittava määrä vaihtelee verkon rakenteen, arkkitehtuurin ja haluttujen tarkkuusvaatimusten mukaan. [3, luku 4]

Opetusdatan kerääminen, luokittelu, käsittely ja ylläpito on oleellinen osa neuroverkon opetusprosessia, mutta se voi olla aikaa vievää ja kallista [15]. Vapaasti käytettävissä olevia data-aineistoja eri tieteenaloilta on saatavilla runsaasti [16]. Esimerkiksi ImageNet data-aineisto sisältää yli 14 miljoonaa luokiteltua kuvaa ja sitä käytetään laajasti kuvan- ja hahmontunnistusneuroverkkojen opetukseen [17].

2.3 Neuroverkon käyttö

Opetetun neuroverkon käyttäminen päättelyyn (engl. inference) edellyttää siinä esiintyvien laskuoperaatioiden suorittamista. Täysin kytkettyjen kerrosten tapauksessa kaava

1 kuvaa yksittäisen päättelimen ulostulon laskemisen. Kaavaa voidaan laajentaa kuvaamaan yksittäisen kerroksen ulostulovektoria $\boldsymbol{a}$.

$$\boldsymbol{a} = g(\overline{\boldsymbol{W}}\boldsymbol{x} + \boldsymbol{b}), \tag{4}$$

jossa $\overline{\boldsymbol{W}}$ on painomatriisi, johon on koottu kaikki kerroksen painot, $\boldsymbol{x}$ kerroksen sisääntulovektori ja $\boldsymbol{b}$ poikkeamavektori. Oletuksena käytetään kerroksen jokaiselle päättelimelle samaa aktivointifunktiota g [3, luku 2]. Oleellisessa osassa on painomatriisin ja sisääntulon välinen matriisitulo. Kaava 4 toimii myös, mikäli sisääntulo $\boldsymbol{x}$ laajenee matriisiksi eli sisältää usean sisääntuloarvon. Kerros kerrokselta eteenpäin siirryttäessä kaava 4 ketjuuntuu eli edellisen kerroksen ulostulo $\boldsymbol{a}$ toimii seuravan kerroksen sisääntulona $\boldsymbol{x}$. Konvoluutiokerroksen tapauksessa on yksinkertaisesti suoritettava konvoluutio-operaatiot sisääntulon ja opetettujen painofunktioiden välillä.

Kuten luvussa 1.2 todettiin, päättely on osa neuroverkon opetusprosessia, kun lasketaan verkon antama tulos häviön laskemista varten. Opetusprosessi sisältää lisäksi verkon läpikäymisen vastavirta-algoritmin avulla sekä useita epookkeja, joten opetuksen voidaan todeta olevan laskennallisesti hyvin paljon raskaampi prosessi kuin päättelyn. Verkon syvyys ja leveys eli kerrosten lukumäärä sekä niissä olevien parametrien määrä vaikuttavat päättelyn aikana tehtävän laskennan määrään, sillä sekä kaavan 4 matriisien dimensiot että ketjujen pituus kasvavat.

Neuroverkon ulostulokerroksen rakennetta muuttamalla voidaan verkosta saada tuloksia erilaisiin tehtäviin. Luokittelutehtävässä tarkoituksena on tunnistaa, mihin opetettuun luokkaa sisääntulo kuuluu. Verkon viimeisenä kerroksena käytetään tällöin usein täysin kytkettyä kerrosta, jossa jokainen päättelin edustaa yhtä luokkaa. [3, luku 4]

Regressiotehtävässä verkko antaa ulostulonaan yhden tai useamman numeroarvon, joka kuvastaa sisääntulodataa. Esimerkiksi talon hinnasta tehty ennuste julkisivukuvan perusteella. Myös regressiossa verkon ulostulona on usein täysin kytketty kerros, jossa jokainen päättelin kertoo yhden numeroarvon. [3, luku 3]

Ulostulona voidaan saada ominaisuuskarttoja, kun viimeisenä kerroksena käytetään konvoluutiokerrosta. Generatiivisilla neuroverkoilla ulostulona on dataa, joka on muokattu sisääntulosta ennalta opittujen ominaisuuksien perusteella. Esimerkiksi yksinkertaisesta piirroksesta voidaan luoda aidon oloinen maisemakuva.

2.4 Neuroverkot sulautetuissa järjestelmissä

Sulautetulla järjestelmällä tarkoitetaan tiettyyn tarkoitukseen tehtyä prosessointijärjestelmää, joka on sulautettu osaksi laitetta tai kokonaisuutta. Koska erilaiset ympäristöt ja

tehtävät asettavat erilaisia vaatimuksia järjestelmille, on myös sulautettuja järjestelmiä monenlaisia. Yleisiä yhteispiirteitä sulautetuissa järjestelmissä ovat energia-, muisti- ja prosessointiresurssien tehokas käyttö, luotettavuus sekä reaaliaikaisuus. Myös mahdollisimman pieneksi optimoitu paino tai hinta ovat joidenkin sulautettujen järjestelmien ratkaisevia ominaisuuksia. Toisin kuin yleiskäyttöön suunnitellulla tietokoneella, sulautetulla järjestelmällä ei välttämättä ole mahdollista suorittaa kuin tietynlaisia tehtäviä erikoistuneen laitteiston, ohjelmiston tai käyttöliittymän takia. [18, luku 1]

Neuroverkot ovat mahdollistaneet esimerkiksi konenäön ja puheentunnistuksen alueilla uudenlaisia, paremman suorituskyvyn saavuttavia sovelluksia. Näiden käyttöön ottamiseksi kaupallisissa sovelluksissa, neuroverkkoihin liittyvän laskennan on oltava riittävän tarkkaa, nopeaa ja kustannustehokasta. Tavallisesti neuroverkkosovelluksia on voitu käyttää pilvipohjaisesti eli mitattu data on lähetetty tietoliikenneverkon yli keskustietokoneelle, jossa neuroverkkoihin perustuva päättely on tehty. Pilvipohjaisissa sovelluksissa ongelmia tuovat tietoliikenneyhteyksien rajallinen tiedonsiirtonopeus ja kapasiteetti sekä yhteyden saatavuus. [2]

Vaihtoehtona pilvipohjaisille sovelluksille, neuroverkkoja käyttäviä sovelluksia voidaan suorittaa paikallisesti, mahdollisesti juuri sovelluksen vaatimuksiin suunnitellulla sulautetulla järjestelmällä. Tätä lähestymistapaa kutsutaan reunalaskennaksi (engl. edge computing). Paikallisuus mahdollistaa ennustettavamman reaaliaikaisuuden ja luotettavuuden, sekä saattaa vähentää kokonaiskustannuksia, sillä keskustietokoneiden pääoma- ja käyttökustannukset ovat suuret. Heikentynyt prosessointi- ja muistikapasiteetti johtaa usein siihen, että paikallisesti suoritetuissa sovelluksissa neuroverkon syvyyttä joudutaan pienentämään, jolloin myös tuloksien tarkkuus alkaa heiketä. [2]

Kuten luvussa 2.3 todettiin, neuroverkon opetusprosessi on moninkertaisesti raskaampi kuin sovelluksessa käytettävä päättelyprosessi. Verkon rakenne ja parametrien arvot ovat kuitenkin sovelluskohtaisia, joka tarkoittaa, että ne voidaan laskea keskitetysti sovelluksen kehityksen osana. Sovellusta suorittavalla laitteella tarvitsee näin ollen suorittaa vain päättelyyn liittyvää laskentaa, kun opetettu neuroverkkomalli otetaan käyttöön osana sovellusta. [10, s 442–443] Muuttuvissa ympäristöissä sovellus voi hyödyntää jatkuvaa oppimista [19], johon liittyvä laskenta voidaan suorittaa joko paikallisesti tai keskitetysti varsinkin, jos muutokset ovat havaittavissa kaikilla sovelluksen ilmentymillä.

Erityyppisiä sovelluksia ja niihin liittyvää tutkimusta neuroverkkosovellusten käyttämisestä sulautetuissa järjestelmissä on saatavilla runsaasti eri aihealueilta [2, taulukko 7]. Esimerkiksi autonomisen ajamiseen liittyvät esteiden, kaistojen ja jalankulkijoiden tunnistustehtävät vaativat sovellukselta suurta luotettavuutta ja reaaliaikaisuutta [20][21].

Puheen- tai kasvojentunnistussovelluksissa reaaliaikaisuus ei ole kriittisen tärkeää, mutta paikallisella toteutuksella saavutetaan pienemmät latenssit ja käyttäjäkokemus paranee [23][24]. Lääketieteellistä dataa käsittelevissä sovelluksissa tietoturvatekijät ovat tärkeä syy paikallisen laskennan käyttöön [2]. Myös datan jalostus sensoritasolla sekä sensorifuusion toteuttaminen hyödyntävät neuroverkkomenetelmiä vähentääkseen kuljetettua datamäärää tietoverkossa ja keventääkseen keskustietokoneiden prosessointikuormaa [22].

3. NEUROVERKKOIHIN LIITTYVÄN LASKENNAN TEHOSTAMINEN

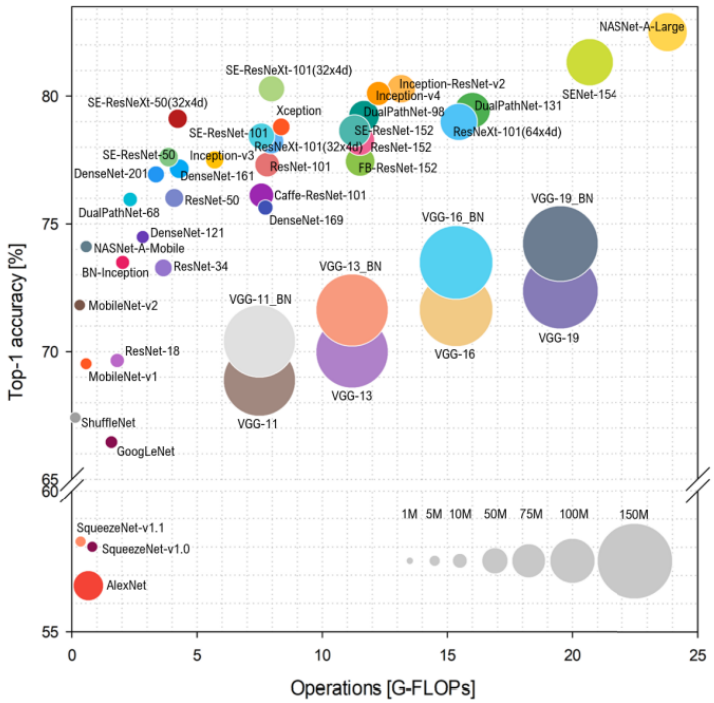
Sulautettujen järjestelmien rajoittunut laskentateho ja muistikapasiteetti sekä reaaliaikavaatimukset aiheuttavat rajoituksia paikallista laskentaa käytettävälle neuroverkolle. Tämän luvun tarkoituksena on tutkia erilaisia keinoja keventää, nopeuttaa ja optimoida neuroverkkoihin liittyvää laskentaa.

3.1 Neuroverkon arkkitehtuurin optimointi

Neuroverkon arkkitehtuurilla tarkoitetaan siinä esiintyvien kerrosten tyyppiä ja määrää ja kerroksissa esiintyvien päättelimien määrää. Samaan sovelluskohteeseen voidaan käyttää useaa erilaista arkkitehtuuria, joilla voi olla suuriakin eroja tarkkuudessa, suorituskyvyssä ja muistinkulutuksessa. Erona neuroverkon rakenteeseen on se, että arkkitehtuuri ei kuvaa kerrosten välisiä yhteyksiä kuten takaisinkytkentää.

Neuroverkkoon pohjautuvaa sovellusta suunniteltaessa on mahdollista rakentaa alusta asti oma neuroverkkoarkkitehtuuri tai käyttää valmista arkkitehtuuria. Ei ole olemassa yhtä tiettyä arkkitehtuuria, joka toimisi hyvin kaikissa sovelluksissa, mutta valmiille arkkitehtuureille on näyttöä siitä, että ne toimivat hyvin tietyissä sovelluksissa [10, luku 5.2.1]. Käytettävän neuroverkon arkkitehtuurin suunnittelua ohjaa sovelluksen tarkoitus ja tarkoitukseen hyväksi todetun rakenteen valinta. Esimerkiksi moniulotteista dataa käsiteltäessä kyseeseen tulee konvoluutiokerrosten käyttäminen ja aikasarjojen käsittelyn tapauksessa takaisinkytketty neuroverkko tai jokin näiden yhdistelmä. Myös halutut tarkkuus- ja suorituskykyvaatimukset on otettava huomioon neuroverkon arkkitehtuuria suunniteltaessa.

Kuva 6 esittää Biancon et al. [25] mittauksia eri arkkitehtuuristen kuvantunnistusneuroverkkojen tarkkuudesta, suoritusnopeudesta ja koosta tietokoneen muistissa. Kuten kuvasta 6 nähdään, enemmän parametreja sisältävä malli ei suoraan tarkoita tarkempia tuloksia, muttei myöskään heikoimpia suoritusnopeuksia. Kuvantunnistusverkot ovat hyvä vertailukohde, sillä niitä on saatavilla runsaasti erilaisilla arkkitehtuureilla ja niitä voidaan käyttää osana monimutkaisempia sovelluksia, kuten hahmontunnistusta toteutettaessa.



Kuva 6. *Yleisten neuroverkkoarkkitehtuurien vertailu. Pystyakselilla top-1 tarkkuus eli kuinka monta prosenttia verkon tuloksista osui täsmälleen oikein. Vaaka-akselilla yhden sisääntulon tulkitsemiseen tarvittava määrä laskentaoperaatioita. Pallon koko kuvaa mallin kokoa opetettavien parametrien määränä (miljoonaa parametria). [25]*

Bianco et al. [25] nostavat eri arkkitehtuurien testauksen tuloksista esiin saavutetun tarkkuuden epälineaarisen riippuvuuden myös suoritettujen operaatioiden määrästä. Esimerkiksi kuvassa 6 SE-ResNeXt-50-arkkitehtuuri saavuttaa yhden suurimmista tarkkuuksista vaatien kolmanneksen vähemmän operaatioita kuin samanlaisen tarkkuuden saavuttava DualPathNet-131-arkkitehtuuri. Toinen nosto mittauksista on, että arkkitehtuurin muistinkulutusta voidaan luotettavasti arvioida siinä käytettävien parametrien määrällä [25]. Lisäksi huomattiin, että kaikki arkkitehtuurit eivät pysty käyttämään suurta parametrimääräänsä tehokkaasti saavuttaakseen hyvän tarkkuuden [25]. Tämä näkyy etenkin VGG-arkkitehtuureissa, jotka ovat melko varhaisessa vaiheessa kehitetty arkkitehtuuriyritys, jonka tarkoituksena oli tutkia, kuinka paljon syvyyttä konvoluutioneuroverkkoon voidaan sisällyttää [26].

Parhaan tarkkuuden sulautetussa järjestelmässä Bianco et al. mittauksissa saavuttivat erilaisilla nopeus ja muistivaatimuksilla ResNet-50- tai MobileNet-arkkitehtuurien neuroverkot [25, taulukko 3]. ResNet-50 on 50 erilaista kerrosta sisältävä konvoluutioverkko, jossa uutena ideana oli helpottaa verkon opetusta käyttämällä jäljentäviä yhteyksiä konvoluutiokerrosten yli. Näin oli mahdollista tunnistaa, mitä erilaisten aktivointikarttojen tulokset esittävät. [27] MobileNet-arkkitehtuuri on kevyt mobiili ja sulautetuille järjestelmille tarkoitettu konvoluutioverkkoarkkitehtuuri, jossa verkon kokoa ja kompleksisuutta voi

mukauttaa vaaditun tarkkuuden perusteella kahdella yksinkertaisella parametrilla. Toinen on verkon sisääntulon dimensio ρ, joka on oletuksena 224 x 224 RGB-pikseliä ja toinen verkon leveys α, jolle 1 on koko leveys ja arvot 0–1 osuus koko leveydestä. [28]

Yhteenvetona neuroverkon arkkitehtuurin optimoinnista voidaan sanoa, että arkkitehtuurin valinnalla ja suunnittelulla voidaan vaikuttaa mallin suorituskykyyn ja muistinkulutukseen. Valmiita arkkitehtuureja käyttämällä voidaan saavuttaa pienemmällä vaivalla haluttu tarkkuus. Esimerkiksi TensorFlow tarjoaa kirjastossaan useita valmiita, tässä luvussakin esillä olleita, kuvantunnistusverkkoja helposti kehittäjien saataville [8].

3.2 Opetuksen tehostaminen

Kuten luvussa 2.4 todettiin, neuroverkkojen opetusprosessi on laskennallisesti raskas, eikä sitä yleensä suoriteta sulautetussa järjestelmässä. Opetus on neuroverkkoa käyttävän sovelluksen kehityksessä oleellinen osa, sillä se mahdollistaa yhdessä sopivan arkkitehtuurin kanssa sovelluskohtaisten tarkkojen tulosten saamisen. Opetusprosessin tehostamiseen on tutkittu monenlaisia tekniikoita, joista perusteellisimpia ja käytetympiä esitellään tässä luvussa.

Suurissa neuroverkkomalleissa ja paljon harjoitusdataa käytettäessä gradientin laskeminen koko harjoitusdatalle käy laskennallisesti hyvin työlääksi. Ratkaisuna tähän on esitetty stokastista, todennäköisyyteen perustuvaa tekniikkaa löytämään arvio gradientista (engl. Stochastic Gradient Descend, SGD). SGD:ssä harjoitusdatasta otetaan jokin osajoukko, jota kutsutaan eräksi (engl. batch). Erälle lasketaan gradientti, joka on arvio todellisesta koko harjoitusdatan gradientista. Mitä suurempaa eräkokoa käytetään, sitä tarkempi arvio gradientista saadaan, mutta toisaalta myös laskentakuorma kasvaa. Liian pieni eräkoko ei anna tarpeeksi tarkkaa arviota gradientista, eikä häviöfunktion minimiä löydetä. Koska harjoitusdataa jaetaan pienempiin eriin, on tarkan arvioin gradientista saamiseksi erityisen tärkeää valita erään satunnainen ja edustava otos koko harjoitusdatasta. [10, luvut 8.1.3, 8.3.1]

Luvussa 2.2 esitetty neuroverkon opetusalgoritmi muuttuu SGD:n käyttöönoton myötä niin, että vaiheita 2 ja 3 suoritetaan jokaisen epookin sisällä jokaiselle erälle. SGD:n erityisiä etuja ovat sen skaalautuvuus, eli yksittäisen erän laskemiseen käytetty aika tai muistinkäyttö ei kasva, vaikka opetusdatan määrä kasvaa [10, s. 292]. Tämä mahdollistaa erittäin suuren opetusdatan käyttämisen ilman, että kaikkea dataa tarvitsee kerralla tallentaa tietokoneen muistiin.

Syvien neuroverkkojen opetuksessa optimaalisten parametrien löytäminen saattaa opetuksen aikana olla vaikeaa katoavan tai nopeasti muuttuvan gradientin takia [10, luku

8.2]. Ongelman välttämiseksi on kehitetty uudenlaisia neuroverkkoarkkitehtuureja kuten ResNet [27] ja myös uusia opetusalgoritmeja. Esimerkiksi Adam ja RMSProp ovat SGD:n ohella suosittuja algoritmeja, joissa opetusastetta muutetaan dynaamisesti [10, luku 8.5].

Neuroverkon opetuksen laskennallisen raskauden ja suurten opetusdatamäärien käsittelemiksi mahdollisimman lyhyessä ajassa saadaan apua laskennan rinnakkaisuudesta. Tämä tarkoittaa usean laskentaoperaation suorittamista samaan aikaan erillisissä laskentayksiköissä. Rinnakkaisuus voidaan neuroverkkojen opetuksessa toteuttaa monella eri tasolla. Yleisiä lähestymistapoja ovat yhden erän datan jakaminen usealle yksikölle tai erien jakaminen usealle yksikölle. Lisäksi laskennassa esiintyvät vektori- tai matriisioperaatiot kuten tulo, konvoluutio ja summa ovat hyvin rinnakkaistettavissa olevia operaatioita. Todella suuria malleja tai opetusdatoja voidaan rinnakkaistaa usealle eri tietokoneelle laskennan mahdollistamiseksi. Parhaassa tapauksessa rinnakkaisuudella voidaan saavuttaa 40 kertaa lyhyempi suoritusaika konvoluutio- tai täysin kytketyn kerroksen tulosten laskemisessa. [29]

Jo melko yksityiskohtaista optimointia opetusprosessiin voidaan löytää erilaisista regularisointimenetelmistä. Luvussa 2.2 esitetyt alinäytteistys- sekä pudotuskerrokset ovat tärkeässä osassa ylisovituksen välttämisessä, mutta myös vähentävät opetuksen aikana suoritettavia laskentaoperaatioita. Eräs yleisesti käytetty regularisointimenetelmä on erän normalisointi, jolla tarkoitetaan kerroksien ulostulojen normalisointia, ennen kuin data syötetään seuraavaan kerrokseen. Normalisoinnilla poistetaan edellisen kerroksen muuttuvien parametrien haittavaikutus seuraavan kerroksen oppimiseen, normalisoimalla ulostulon arvot tietylle arvovälille tietyllä jakaumalla. [30] Normalisointi vähentää laskennan määrää, sillä harjoitusdataa ja epookkeja tarvitaan vähemmän. Lisäksi normalisointi auttaa hallitsemaan ylisovitusta.

Neuroverkon opetusprosessia ei ole tarpeen aloittaa aina alustamalla verkon painoja satunnaisluvuilla, vaan opetuksen alkuna voidaan käyttää jo valmiiksi löydettyjä painoja. Jo opetetun neuroverkon uudelleen opettamista suoriutumaan paremmin tehtävästään tai suoriutumaan uudenlaisesta tehtävästä kutsutaan siirto-opetukseksi (engl. Transfer Learning). Esiopetettua verkkoa voidaan käyttää osana uutta arkkitehtuuria ominaisuuksien erottelijana tai arkkitehtuuri voidaan säilyttää samana ja vain opettaa verkon painoja uudella datalla. Siirto-oppimista kannattaa hyödyntää, kun saatavilla on esiopetettu toimiva malli sekä jonkin verran uudenlaista harjoitusdataa. Vaatimuksen opetusdatan määrästä siirto-oppimista käytettäessä ovat huomattavasti pienemmät kuin mallia alusta asti opetettaessa. [3, luku 7] Siirto-oppimisen avulla opetukseen tarvittavan laskennan

määrä vähenee huomattavasti, sillä häviöfunktion arvo on jo opetuksen alussa pienempi kuin satunnaisista painoista aloitettaessa.

Esimerkiksi kuvia sisääntulona käyttävä neuroverkkosovellus voi käyttää ImageNet datalla esiopetettua sopivaa neuroverkkoarkkitehtuuria. Siirto-opetuksen avulla malli saadaan suoriutumaan uudesta tehtävästään kohtuullisella määrällä harjoitusdataa ja laskentatehoa. Edellisessä luvussa esitellyt yleiset kuvantunnistukseen tarkoitetut neuroverkkoarkkitehtuurit ovat usein saatavilla myös ImageNet-datalla opetetuilla painoilla [8].

3.3 Päättelyn tehostaminen

Neuroverkkosovelluksen käyttövaiheessa suoritetaan neuroverkolla päättelyä eli verkkoon syötetään dataa, josta lasketaan tulokset. Tehokas ja mahdollisimman vähän laskentatehoa ja energiaa kuluttava päättely on oleellista paitsi sulautetuissa järjestelmissä, myös keskitetyissä laskentajärjestelmissä kustannusten ja latenssin vähentämiseksi. Tässä luvussa tutkitaan yleisimpiä päättelyn optimointimenetelmiä, sekä esitellään optimointia varten kehitettyjä työkaluja.

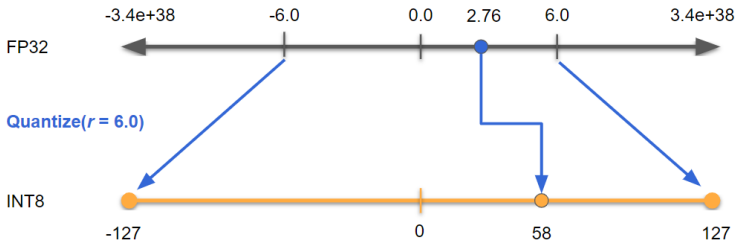
Kuten neuroverkon opetuksessa, päättelyä voidaan tehostaa laskennan rinnakkaisuudella. Usean laskentayksikön samanaikainen työskentely nopeuttaa vektori ja matriisitulojen laskentaa. Erityisen tehokkaisiin matriisituloihin päästään koostamalla sisääntulodataa eriin. Erona opetusvaiheen datan pilkkomisessa eriin, päättelyvaiheessa erät muodostetaan vain, koska niiden avulla saadaan suurempi etu rinnakkaislaskennasta. [31]

Eräkokoon päättelyvaiheessa vaikuttavat laitteiston ominaisuuksien lisäksi myös sovelluksen vaatimukset ja saadun datan muoto. Jos tulokset täytyy saada mahdollisimman nopeasti, peräkkäisiä datapisteitä ei ehditä koostamaan suuremmaksi eräksi, vaan laskenta on suoritettava heti käyttäen pientä eräkokoa. Jos dataa saadaan kerralla suurempi määrä, on se tehokasta prosessoida suurissa erissä. [10, luku 12]

Eräs tapa vähentää laskennan määrää ja näin tehostaa päättelyä on pienentää sisääntulon resoluutiota. Sabottke ja Spieler [32] esittävät lääketieteellisen datan tapauksessa, että resoluution laskiessa neuroverkon tulosten tarkkuus heikkenee, mutta päättelyn tehokkuus kasvaa. Ongelmana sisääntulon resoluution muuttamisessa on eteenpäin kytketyn neuroverkon arkkitehtuurin joustamattomuus. Uudelle resoluutiolle on suunniteltava ja opetettava uusi neuroverkko, jonka sisääntulokerroksen dimensio täsmää datan kanssa.

Datan käsittelyä tietokoneessa voidaan tehostaa käyttämällä vähemmän tallennustilaa vieviä datatyyppejä. Yleisesti käytetty datatyyppi esimerkiksi kuvapikselin arvoille voi olla

64- tai 32-bittinen liukuluku. Tämä voidaan muuttaa esimerkiksi puolet vähemmän muistia vieväksi 16-bittiseksi liukuluvuksi tai 8-bittiseksi kokonaisluvuksi kvantifioinnin avulla. Kuva 7 esittää kvantifiointia 32-bittisen liukuluvun ja 8-bittisen kokonaisluvun välillä. Datan symmetrisyyden säilyttämiseksi on liukulukuarvot ensin skaalattu -6–6:teen [33].



Kuva 7. *32-bittisen liukuluvun kvantifiointi 8-bittiseksi kokonaisluvuksi [33].*

Kvantifiointia suoritetaan sekä sisääntuloon että neuroverkon kerroksien painoihin. Neuroverkon arkkitehtuuria ei tarvitse muuttaa, joten kvantifiointi on käytettävissä helposti monelle erilaiselle mallille. Kvantifiointi vaikuttaa vain hieman neuroverkon tuloksien tarkkuuteen, mutta sillä saavutetaan tyypillisesti 2–3 kertaa nopeampi päättely sekä pienempi muistin ja energian kulutus. Mallin koko muistissa voi kvantifioinnin jälkeen tippua neljännekseen alkuperäisestä. Tarkkuuden lisäämiseksi voidaan neuroverkkoa opettaa jo valmiiksi kvantifioidulla datalla. [34]

Opetettua neuroverkkoa analysoimalla voidaan siitä karsia (engl. pruning) laskentaoperaatioita laskennan nopeuttamiseksi. Verkosta voidaan poistaa merkityksettömiä eli lähellä nollaa olevia painoja. Lisäksi, jos jokin päättelimen ulostulo jää jatkuvasti hyvin pieneksi, sillä on hyvin pieni vaikutus tulokseen ja se voidaan poistaa. Karsimista voidaan siis tehdä sovelluskohtaisesti, jos käytössä on jonkin verran dataa. Usein karsimista voidaan suorittaa iteratiivisesti eli jokaisen painon tai päättelimen poiston jälkeen tutkitaan ovatko tarkkuus ja laskennan tehokkuus halutulla tasolla. Karsiminen voi johtaa laskennassa käytettävien matriisien harvenemiseen (engl. sparse) eli matriiseissa on paljon nolla-arvoja. [35] Harvoille matriiseille on kehitetty erityisiä tehokkaita tallennus ja laskentatapoja, joissa nolla-arvot jätetään huomiotta. [36].

Useimmat neuroverkkopäättelyä tarjoavat ohjelmointikirjastot, kuten TensorFlow, kykenevät oletuksena käyttämään laitteistoa rinnakkaislaskentaan ja tarjoavat mahdollisuuden kontrolloida päättelyn eräkokoa [8]. Lisäoptimointien suorittamiseksi tarvitaan kuitenkin erillisiä optimointikirjastoja. Seuraavaksi esitellään Nvidian GPU-arkkitehtuuriin neuroverkkolaskentaa optimoivaa TensorRT-työkalua.

Neuroverkon päättelyä tehostava Nvidian TensorRT-työkalun optimointi koostuu kvantifioinnista sekä käytössä olevan laitteiston optimaalisesta käytöstä. Se tukee kvantifiointia

32-bittiseen liukulukuun, 16-bittiseen liukulukuun ja 8-bittiseen kokonaislukuun, sekä mahdollistaa myös kvantifioinnin huomioimisen opetuksessa. TensorRT valitsee käytettävistä olevista algoritmeista sopivimmat kyseiselle suoritusalustalle, sekä optimoi laitteen muistinkäyttöä käyttämällä CUDA-rinnakkaisohjelmointiviitekehystä. Lisäksi TensorRT optimoi usean sisääntulon käyttöä ja yhdistää aika-askelia takaisinkytketyissä neuroverkoissa [37]. Rajoitteena on TensorRT:n soveltuvuus ainoastaan Nvidian GPU:n sisältäviin järjestelmiin. Käytännön optimointi TensorRT:llä tapahtuu käyttäen korkean abstraktion C++, tai Python rajapintaa tai vaihtoehtoisesti muihin neuroverkkokirjastoihin sisällytettyä lisäosaa [37].

TensorRT tukee käytetyimpiä neuroverkkojen tallennusmuotoja, joista malli voidaan tuoda TensorRT-ympäristöön. Tuonnin jälkeen malli käännetään erillisenä vaiheena TensorRT-moottoreiksi. Moottori sisältää yhden tai useamman mallissa suoritettavan operaation tai operaatioiden jonon, joka on optimoitu käytössä olevalle laitteistolle TensorRT:n metodeilla. Tuettujen operaatioiden määrä on rajallinen eli kaikenlaisia neuroverkkorakenteita tai laskentakerroksia ei voi optimoida TensorRT:llä. [38]

TensorFlow:n sisäänrakennettu TensorRT-rajapinta (TF-TRT) on korkea abstraktion rajapinta TensorRT:n käyttämiseen, kun malli on tallennettu tai rakennettu TensorFlow:lla. Sen avulla TensorFlow-mallin voi kääntää TensorRT-moottoriksi, joka voidaan edelleen tallentaa tai ajaa TensorFlow:n rajapinnalla. Kerrokset tai operaatiot, joita TensorRT ei tue, suoritetaan normaalilla TensorFlow:lla. Kontrolloitavia parametreja muunnoksen tekemiseen ovat halutun TF-TRT -mallin käyttämä laskentatarkkuus ja käytettävissä oleva muistikapasiteetti. Lisäksi voidaan asettaa pienin lukumäärä kerroksille, joista TensorRT-moottori voidaan muodostaa ja maksimimäärä TensorRT-moottoreita, jotka tallennetaan eridimensioisiin sisääntuloihin varautuen. TF-TRT-malli voidaan myös kääntää valmiiksi, jolloin mallin käyttö voidaan aloittaa nopeammin. [39] TF-TRT saavuttaa Nvidian mukaan 8-kertaisen päättelynopeuden verrattuna TensorFlow:n päättelyyn [33].

TensorRT:n Python-rajapinnan avulla on mahdollista suorittaa mallin muunnos ja päättelyä tuetuista tallennusformaateista. Muunnoksen aikana käyttäjä voi kontrolloida samoja parametreja kuin TF-TRT:n tapauksessa, mutta rajapinta antaa käyttäjälle myös enemmän mahdollisuuksia kontrolloida muun muassa muistinhallintaa. Python-rajapinnan ominaisuudet ovat samat kuin TensorRT:n C++-rajapinnalla, joskin C++-rajapintaa käyttämällä voidaan saavuttaa parempi päättelynopeus. [38, luku 2]

3.4 Laitteisto

Neuroverkkosovellusta sulautettuun järjestelmään suunniteltaessa voidaan laitteiston ominaisuuksiin usein vaikuttaa. Tässä luvussa tutkitaan, minkälaisella laitteistolla edellisissä luvuissa esitettyjä optimointimenetelmiä voidaan toteuttaa käytännössä sekä millaisia erityisratkaisuja neuroverkkolaskentaan on tähän mennessä kehitetty.

Sekä päättely- että opetusvaiheen laskentaa voidaan nopeuttaa huomattavasti rinnakkaisuuden avulla. Laitteiston tasolla rinnakkaisuus tarkoittaa, että useampi laskentayksikkö suorittaa saman tehtävän osakokonaisuutta samaan aikaan. Nykyaikaiset tietokoneiden keskusprosessoriyksiköt (engl. Central Processing Unit, CPU) sisältävät usein muutamia laskentaytimiä, joilla laskentaa voi suorittaa hieman rinnakkain. CPU:t ovat kuitenkin suunniteltu etenkin suorittamaan nopeasti sarja peräkkäisiä operaatioita. Grafiikkaprosessoreissa (engl. Graphics Processing Unit, GPU) taas on satoja tai tuhansia laskentaytimiä, joissa yksittäisen operaation suorittaminen vie enemmän aikaa. GPU:t ovat alun perin suunniteltu grafiikan esittämiseen tietokoneen näytöllä, joka koostuu massiivisesti rinnakkaistettavista vektori- ja matriisioperaatioista. [40]

Pelkän GPU:n käyttäminen laskennassa ei onnistu, vaan laskennan jakamiseen ja kontrollointiin tarvitaan avuksi CPU:ta. Kahden eri prosessorityypin yhteistoimintaa kutsutaan heterogeeniseksi laskennaksi. Heterogeenisen laskennan ohjelmointiin on tarjolla useita viitekehyksiä, kuten Nvidian GPU-arkkitehtuuriin soveltuva CUDA ja yleinen rinnakkaisohjelmointiviitekehys OpenCL. [41]

GPU:n lisäksi laskentaa voidaan tehostaa nimenomaan neuroverkkolaskentaan ja päättelyyn tarkoitetuilla mikropiireillä sekä ohjelmoitavilla porttilogiikoilla (FPGA) [2, luku 4]. Esimerkiksi Google on kehittänyt erityisesti neuroverkkolaskentaan soveltuvia tensorisuorittimia (TPU), joissa laskentaa tehostetaan rinnakkaisuuden lisäksi käyttämällä vain kokonaislukuoperaatioita. Näin sama laskuoperaatio saadaan tehtyä pienemmällä määrällä transistoreita. [42] Google on kehittänyt tensorisuorittimesta myös vain päättelyyn soveltuvan hyvin pienikokoisen Edge TPU -version, joka tehostaa päättelynopeutta ollen silti energiatehokas ja edullinen [43].

Tehokkaan GPU:n sisällyttäminen sulautettuun järjestelmään voi olla mahdotonta, jos järjestelmän paino-, koko- tai energiankulutusvaatimukset ovat tiukat. Lisäksi päättelyyn käytettävän laitteen käyttömuistin (RAM) on oltava riittävä, jotta käytettävä malli ja käsiteltävä data mahtuu muistiin. Käyttömuistin koko, sekä rinnakkaisten suorittimien lukumäärä määrittää edellisessä luvussa esitellyn laitteistolle optimaalisen eräkoon [10, s. 276].

4. KUVANTUNNISTUS JETSON NANO -JÄRJESTELMÄLLÄ

Tässä luvussa esitellään työn yhteydessä tehty vertailu ja sen tulokset erilaisista neuroverkon päättelyn optimointitekniikoista. Aluksi esitellään työssä käytetty laitteisto ja ohjelmisto.

4.1 Nvidia Jetson Nano

Työn kokeellisessa osuudessa käytettiin Nvidian Jetson Nano -järjestelmää. Se kuuluu Nvidian Jetson-tuoteperheeseen, jonka tarjoaa pienikokoisia, mutta tehokkaita tietokoneita tekoälysovellusten käyttöön ja kehitykseen osana sulautettuja järjestelmiä. Laitteiston lisäksi Jetson koostuu Linux-käyttöjärjestelmään pohjautuvasta laitteille optimoidusta kehitysympäristö JetPackistä. [44]

Nano on Jetson-tuoteperheen perusmalli, jonka matala hinta yhdistettynä riittävään suorituskykyyn tarjoavat mahdollisuuksia kevyimpiin teköälysovelluksiin ja kehityksen opetteluun. Nanossa on 128-ytiminen Maxwell-sarjan grafiikkaprosessori yhdistettynä neliytimiseen ARM-arkkitehtuurin keskusprosessoriin. Keskusmuistia laitteessa on 4 gigatavua ja kiintolevynä toimii erillinen mikro SD-kortti. Järjestelmän prosessorit jakavat yhteisen keskusmuistin, joka poistaa ongelman datan siirtämisestä muistista toiseen. Laitteesta löytyy lisäksi neljä USB 3.0 porttia, HDMI- ja display-portit, gigabitin ethernetyhteys sekä MIPI- ja GPIO-väylät erilaisten laitteiden, kuten kameroiden liittämistä varten. Nanon virrankulutusta on mahdollista vaihtaa kahden eri tilan välillä. Pienempää 5 watin tehoa voidaan käyttää esimerkiksi, kun virtalähteenä käytetään akkua. Rajoittamattomassa tilassa laitteen virrankulutus on noin 10 wattia. [45] Laitteen pituus, leveys ja korkeus ovat noin 8, 10 ja 3 senttimetriä. Kuvassa 8 Jetson Nano ilman kotelointia. Laitteen päällä näkyy lämmönsiirrin, joka huolehtii prosessoreiden jäähdytyksestä, sillä erillistä tuuletinta laitteessa ei ole.

Kuva 8. *Nvidia Jetson Nano [45]*

Testit toteutettiin käyttäen JetPackin versiota 4.6 ja siihen valmiiksi asennettua Python 3.6.9:ä sekä Googlen avoimen lähdekoodin TensorFlow 2.5.0 -koneoppimiskirjastoa. Lisäksi osassa testejä käytettiin TensorRT:n Python-rajapintaa ja erillistä kirjastoa onnx-mallien testaukseen. Käyttömuistin kulutuksen seurantaan testien aikana käytettiin JetPackin `tegrastats`-komentoa. Testien aikana Nanoa hallinnoitiin etäyhteydellä WinSCP- ja Putty-sovellusten avulla Windows-tietokoneelta, jolla myös saadut tulokset analysoitiin.

4.2 Sovelluksen esittely ja toteutus

Työn kokeellisen osuuden tarkoituksena oli selvittää, miten neuroverkon päättelyä voidaan optimoida sulautetussa järjestelmässä. Luvun 3 perusteella kokeisiin valittiin eri arkkitehtuurien testaus, eräkoon vaikutuksen testaus, sekä valmiin optimointityökalun TensorRT:n käyttö. Lisäksi tutkittiin laitteiston rinnakkaisuuden ja laitteiston erilaisten tilojen vaikutusta päättelynopeuteen. Kokeita varten kehitettiin arkkitehtuurien, eräkoon ja optimointien vaikutuksen tutkimiseen testipenkki, jolla on mahdollista mitata mallin tarkkuutta ja nopeutta. Lisäksi tutkittiin mallien kokoa tallennettuna tietokoneen levylle, sekä niiden päättelyn aikana kuluttamaa käyttömuistia. Testipenkki toteutettiin kuvantunnistustehtävän avulla. Kuvantunnistuksessa neuroverkolle syötetään yksi kuva ja tuloksena saadaan todennäköisyys, esittääkö kuva kyseistä luokkaa, kullekin ennalta verkolle opetetulle luokalle. Työssä käytetyn testipenkin ja tarvittavien analysointityökalujen lähdekoodit ovat saatavilla GitHubissa [46].

Testeihin käytettiin ImageNet_v2-data-aineistoa, joka sisältää 10 000 erikokoista värikuvaa 1000:sta eri luokasta. Aineisto on kerätty noudattaen alkuperäisen ImageNet-aineiston protokollaa. [47] Neuroverkkoina käytettiin TensorFlow:ssa saatavilla olevia valmiita

ImageNet-kuvilla opetettuja malleja. Testeihin valittiin eri määrän parametreja sisältäviä malleja arkkitehtuurin optimoinnin testaamiseen, painottaen kuitenkin pienikokoisimpia malleja, sillä suurimmat arkkitehtuurit vaativat liikaa muistia ja prosessointitehoa käytettävältä laitteistolta. Kaikki käytössä olleet mallit käyttävät sisääntulonaan 224 x 224 x 3 kokoisia kuvia, joten data-aineiston kuvat rajattiin sen kokoisiksi. Muita kuvan koon muuttamisen metodeja ja niiden aiheuttamaa poikkeamaa tuloksiin ei tutkittu.

Mallien suorituskyvyn arviointiin jouduttiin käyttämään useaa eri rajapintaa, joiden tulokset eivät ole keskenään vertailukelpoisia. Arkkitehtuurin, eräkoon ja laitteiston suorituskykyvertailut toteutettiin TensorFlow:n korkean abstraktion Keras-rajapintaa hyödyntäen. TensorRT:n optimointeja varten toteutettiin suorituskykyarviointia myös puhtaalla TensroFlow:lla ja TensorRT:n Python-rajapintaa käyttäen. Lisäksi tutkittiin TensorRT:n C++-rajapintaa hyödyntävää JetPackin trtexec-työkalua [38, luku A.3.1] TensorRT:n optimoinnin suorittamiseen ja päättelynopeuden testaamiseen. Kvantifioinnissa käytettiin 32- ja 16-bittisiä liukulukuja, sillä Jetson Nanon GPU ei tue 8-bittistä kokonaislukulaskentaa [38].

Mitattuja suorituskykyarvoja ovat päättelyn tarkkuus ja nopeus. Tarkkuuden mittaamiseen käytettiin kahta mittaria: top-1 ja top-5 tarkkuus. Kuvantunnistuksessa neuroverkon tuloksena saadaan todennäköisyys kullekin luokalle, jotka voidaan järjestää suuruusjärjestykseen. Top-1 tarkkuus tarkoittaa, että tuloksen suurimman todennäköisyyden luokka on sama kuin testikuvan oikea luokka. Top-5 taas tarkoittaa, että oikea luokka löytyy viiden suurimman todennäköisyyden joukosta. Top-1 ja Top-5 tarkkuudet laskettiin testeissä koko 10 000 kuvaa sisältävälle data-aineistolle.

Päättelyn nopeutta mitattiin kuvina sekunnissa (engl. Frames Per Second, FPS), eli kuinka monta kuvaa sekunnissa neuroverkko kykenee luokittelemaan. Vertailukohtana voidaan pitää yleisesti videoiden esittämiseen käytettyä 24 FPS:n kuvataajuutta [48]. Jos neuroverkko saavuttaa yli 24 FPS:n nopeuden, voitaisiin sitä käyttää reaaliaikaisesti videon analysointiin. Päättelynopeutta mitattiin pienemmällä 320:n kuvan otoksella data-aineistosta, jolle laskettiin tulokset 20 kertaa. Nopeimmasta kierrosajasta laskettiin tulos FPS-yksikköön. Nopeimman suoritusajan valitsemiseen päädyttiin, sillä tuloksissa esiintyi usein joitakin todella hitaita kierroksia ja muut kierrokset olivat usein lähes yhtä nopeita keskenään. Näin nopeimman valitsemisen katsottiin olevan lähempänä todellista suoritusaikaan kuin keskiarvon laskemisen. Päättelynopeuden mittauksissa datan esiprosessointiin tai tuloksen käsittelyyn käytettyä aikaa ei laskettu suoritusaikoihin, vaan mitattiin vain neuroverkon päättelyn suorittamiseen käytetty aika.

Käyttömuistin varattua kapasiteettiä arvioitiin suorituskykymittausten aikana JetPackin `tegrastats`-komennolla viiden sekunnin välein. Näytteet koottiin tiedostoon, jota analysoitiin piirtämällä kuvaaja sekä selvittämällä minimi- ja maksimiarvot. Muistinkulutuksen mittauksissa huomattavaa epävarmuutta aiheuttaa tietämättömyys siitä, mitä keskusmuistissa on kullakin hetkellä tallennettuna.

Työn kokeellisessa osuudessa koettiin huomattavia vaikeuksia etenkin optimointityökalu TensorRT:n käytössä. Hankaluudet näkyvät tuloksissa useina erilaisina ei-vertailukelpoisina testijärjestelyinä. Lisäksi muistinkulutuksen vertailu todettiin haastavaksi, sillä korkean abstraktion rajapintoja käytettäessä on mahdoton arvioida, mitä dataa keskusmuistissa on tallennettuna. Huomattiin esimerkiksi, että pelkän testipenkin ja tarvittavien kirjastojen tuonti Pythonin työtilaan aiheutti pysyvän noin 1 GB:n nousun muistin käytössä.

Muita neuroverkkojen optimointikyökaluja, kuten TensorFlow Lite:ä [51], ei päästy kokeilemaan, sillä TensorFlow Lite ei tue Nvidian GPU:n käyttöä. Myöskään sisääntulodatan keventämistä ei tutkittu, sillä tutkituille neuroverkkoarkkitehtuureille ei löytynyt erikokoisia sisääntuloja tukevia vaihtoehtoja Keras-rajapinnasta. Neuroverkon aktivointeja ja painoja analysoimalla saatua optimointia eli harvennusta ei testeissä toteutettu.

4.3 Sovelluksen tulokset ja päätelmät

Eri neuroverkkoarkkitehtuurien suorituskyvyn testaamiseen käytettiin TensorFlow:n Keras-rajapintaa ja ImageNet_v2 data-aineistoa. Mallien koko levyllä mitattiin käyttäen Linux-komentoa `du -s <mallin_nimi>`. Top-1 ja top-5 tarkkuudet laskettiin koko 10 000 kuvaa sisältävälle data-aineistolle. Päättelynopeutta mitattiin pienemmällä 320:n kuvan otoksella, jolle laskettiin tulokset 20 kertaa ja nopeimmasta kierrosajasta laskettiin tulos FPS-yksikköön. Kaikissa Taulukon 1 mittauksissa käytettiin eräkokoa 32 ja Nanon normaalia 10 W:n tehotilaa. MobileNet-arkkitehtuurissa $\propto$ kuvaa kuinka paljon verkon kokoa on supistettu alkuperäisestä.

Taulukko 1. *Neuroverkkoarkkitehtuurien vertailu Jetson Nano -järjestelmällä.*

Mallin nimi ja julkaisu	koko levyllä (MB)	parametrit	top-1 (%)	top-5 (%)	nopeus (FPS)	muisti max. (GB/4GB)
MobileNet ($\propto = \mathbf{0,25}$) [28]	3,9	475 544	34,29	57,31	133	3,41
MobileNet ($\propto = \mathbf{0,5}$)	7,2	1 342 536	45,53	68,51	86	3,83
MobileNet ($\propto = \mathbf{0,75}$)	12,2	2 601 976	49,74	71,90	59	3,83
MobileNet ($\propto = \mathbf{1}$)	18,7	4 253 864	51,72	74,15	45	3,82
MobileNetV2 ($\propto = \mathbf{0,5}$) [49]	11,7	1 987 224	45,9	69,08	70	3,78
MobileNetV2 ($\propto = \mathbf{1}$)	17,8	3 538 984	52,31	75,07	42	3,84
ResNet-50 [27]	104,5	25 636 712	59,55	80,44	17	3,84
DenseNet-121 [50]	41,1	8 062 504	58,56	80,43	19	3,86

Taulukon 1 tuloksista on yleistettävissä, että mallin koon kasvaessa sen tarkkuus paranee, mutta suoritusnopeus hidastuu. Tulokset ovat pääpiirteittäin linjassa Keras-rajapinnan dokumentaatiosta löytyvän taulukon kanssa [52]. Dokumentaation taulukon tarkkuuslukemat on saatu ImageNet-aineiston validointidatalla, mistä ero tarkkuuslukemissa johtuu. Validointidataa on saatettu käyttää mallien opetusvaiheessa mittarina siitä, kuinka hyvin verkko suoriutuu, joka selittäisi eron tarkkuusluvuissa. Toisaalta ImageNet_v2 aineisto voi yksinkertaisesti sisältää neuroverkoille ”haastavampia” tapauksia. Tässä työssä tehdyissä mittauksissa DenseNet121-arkkitehtuuri saavutti suuremman päättelynopeuden sekä heikomman tarkkuuden kuin ResNet50. Dokumentaation taulukossa tulokset ovat päinvastaiset eikä syytä tähän osattu selvittää. Käyttömuistin kulutuksen mittaus on hyvin epävarma ja siihen vaikuttaa useita muitakin tekijöitä päättelyn aikana kuin neuroverkossa suoritettava laskenta. Tuloksista on kuitenkin nähtävissä, että mallin parametrimäärän kasvaessa myös muistinkäyttö kasvaa.

MobileNet-arkkitehtuurin $\propto$-parametri vaikutti mittauksissa huomattavasti mallien kokoon ja parametrimäärään. Erot suorituskyvyssä näyttävät korostuvan $\propto$:n pienetessä. Huomionarvoista on myös, kuinka MobileNetV2 arkkitehtuuri kykenee käyttämään parametrimääräänsä tehokkaammin kuin MobileNet. Tarkkuutta on pystytty V2-versioon kasvattamaan hieman, mutta tehokkaampi parametrien käyttö näkyy heikompana päättelynopeutena.

Seuraava tutkittu optimointimenetelmä on eräkoon vaikutus päättelynopeuteen. Eräkoko ei vaikuta päättelyn tarkkuuteen, joten tarkkuuslukemia ei mitattu. Taulukossa 2 on esitetty eräkoon vaikutus päättelyn nopeuteen käyttäen MobileNet-arkkitehtuuria ($\propto = 1$) ja Jetson Nanon normaalia 10 W:n tehotilaa.

Taulukko 2. *Vertailu eräkoon vaikutuksesta päättelynopeuteen Mobilenet-arkkitehtuurilla* ($\propto = 1$).

eräkoko	nopeus (FPS)	muisti max. (GB)
1	32,3	3,77
2	37,2	3,85
4	40,9	3,76
8	43,7	3,84
16	44,8	3,82
32	45,0	3,82
64	43,9	3,82
128	41,0	3,85

Taulukon 2 tuloksista huomataan, että eräkoon kasvattaminen nopeuttaa päättelyä, kunnes saavutetaan laitteiston muistirajoitteet. Eräkoon kasvaessa, kerralla keskusmuistiin tallennettavan datan määrä lisääntyy. Eräkokoa 64 käytettäessä päättely hidastuu hieman verrattuna edelliseen eräkokoon johtuen muistin osittaisesta täyttymisestä ja 128 eräkoko aiheuttaa muistiin jo niin suurta kuormitusta, että päättely hidastuu huomattavasti. Tätä suurempia eräkokoja ei onnistuttu kokeilemaan, sillä muisti täyttyi ja päättely keskeytyi. Jälleen korkean abstraktion rajapintoja käytettäessä muistin hallintaan ei pystytä vaikuttamaan, eikä tutkimusta tehdessä ole tietoa, miten muistia hallinnoidaan päättelyn aikana.

Tulokset ovat Biancon et al. [25] taulukkoon 1 verrattuna samankaltaiset, lukuun ottamatta eräkoon 64 hidastumista verrattuna eräkokoon 32 näissä mittauksissa. Käytössä olevat laitteet ovat ominaisuuksiltaan hyvin samankaltaiset, joten ero saattaa johtua päättelyyn käytetystä ohjelmistosta. Muistin mittaukset taulukossa 2 ovat yleisesti nousevia eräkoon kasvaessa. Mittauksiin aiheutuu kuitenkin häiriötä, koska muistinhallintaan ei voitu kokeissa vaikuttaa.

Jetson Nano -järjestelmää on mahdollista käyttää normaalin 10 W:n tehotilan lisäksi 5 W:n tilassa, jos sovelluksen energiavaatimukset sitä vaativat. Lisäksi Nanon CPU:n ja GPU:n laskentataajuuksia, eli kuinka monta operaatiota yhdessä laskentayksikössä suoritetaan sekunnissa, voidaan optimoida käyttäen JetPackin `jetson_clocks`-komentoa [53]. Taulukoon 3 on koottu tulokset Jetson Nanon eri tilojen vaikutuksesta päättelyn nopeuteen käyttäen MobileNet-arkkitehtuuria ($\propto = 1$) ja eräkokoa 32. Perf. mode -tilalla viitataan tilaan, jossa kellotaajuudet ovat optimoitu `jetson_clocks`-komennolla. Lisäksi mitattiin suoritusnopeus vain laitteen keskusprosessoria eli CPU:ta käyttäen. Päättelyn tarkkuutta tai muistinkulutusta ei tilojen vertailujen yhteydessä mitattu, sillä niissä ei ole muutoksia.

Taulukko 3. *Jetson Nanon eri tilojen vaikutus päättelynopeuteen Mobilenet-arkkitehtuurilla* ($\propto = 1$).

Nanon tila	nopeus (FPS)
5 W	32,8
10 W	45,0
perf. mode	45,2
vain CPU	7,4

Taulukon 3 tuloksista huomataan, että GPU:n käyttäminen nopeuttaa päättelyä huomattavasti. Syynä tähän on laskennan massiivinen rinnakkaisuus GPU:lla, johon pelkkä CPU ei kykene. Kellotaajuuksia optimoivan perf. moden käyttäminen ei näytä vaikuttaneen päättelyn nopeuteen juurikaan. Komento asettaa dynaamisesti laskentakuorman mukaan muuttuvat kellotaajuudet maksimiarvoonsa, jonka ne joka tapauksessa saavuttavat laskennan aikana. Komennosta voisi olla hyötyä siis tilanteessa, jossa laitteistolta vaaditaan nopeaa reaktiivisuutta herätteeseen.

5 W:n tehotilaa käytettäessä päättelyn nopeus hidastuu, muttei puolitu verrattuna normaaliin 10 W:n tehotilaan. 5 W:n tilassa Jetson Nano kykenee siis suorittamaan enemmän laskentaoperaatioita wattia kohden. Esimerkiksi akkuja virtalähteenä käyttävän sovelluksen tapauksessa 5 W:n tila voi olla hyvin käyttökelpoinen.

Optimointityökalu TensorRT:tä käytettiin kokeissa kolmella eri tavalla, joiden tulokset eivät ole vertailtavissa keskenään. Ensimmäiseksi testattiin TensorFlow:n sisäänrakennettua TensorRT-rajapintaa TF-TRT. Toiseksi tutkittiin TensorRT:n Python-rajapintaa ja JetPackin trtexec-työkalua.

Taulukko 4. *TF-TRT optimointi Mobilenet-arkkitehtuurilla* ($\propto = 1$).

optimointi	top-1 (%)	top-5(%)	nopeus (FPS)	muisti max. (GB)
TensorFlow	51,72	74,15	64,0	3,4
TF-TRT 32-bittinen liukuluku	51,72	74,15	64,1	3,6
TF-TRT 16-bittinen liukuluku	51,72	74,15	66,0	3,5

Taulukkoon 4 on koottu tulokset TF-TRT optimoinnilla saadut tulokset käyttäen eräkokoa 32 ja Jetson Nanon normaalia tehotilaa. Tulokset ovat tarkkuuden ja päättelynopeuden osalta lähes identtiset, joka saattaa tarkoittaa, että TensorRT-moottoreiden luonti epäonnistui ja käytössä on ollut TensorFlow:n päättely. Huomataan kuitenkin, että TensorFlow:lla suoritettu päättely vaikuttaisi mittausten perusteella olevan selvästi nopeampaa kuin Keras-rajapinnan käyttö.

TensorRT:n Python-rajapinnalla ja Trtexec-työkalulla tehdyissä testeissä jouduttiin käyttämään eri tallennusformaattia neuroverkkomallille ohjelmiston rajoituksista johtuen.

Käyttöön otettiin standardoitu ONNX-malliformaatti ja ResNet-50 kuvantunnistusarkkitehtuuri ONNX Model Zoo -kirjastosta [54]. Koska käytettiin eri lähteestä tulevaa mallia, tulosten vertailukelpoisuus täytyy kyseenalaistaa. Sama parametrimäärä ja lähellä olevat tarkkuuslukemat sekä dokumentaatiossa että tämän työn testeissä antavat olettaa, että mallit ovat vertailukelpoisia. Trtexec-työkalulla voidaan suorittaa vain mallien päättelynopeuden vertailua, joten ONNX-mallin ja TensorRT-mallien tarkkuuksien tutkiminen toteutettiin työkalun ulkopuolella TensorRT:n Python-rajapinnalla.

Taulukko 5. *TensorRT-optimointi käyttäen ResNet-50-arkkitehtuuria ja eräkokoa 1.*

optimointi	top-1 (%)	top-5 (%)	nopeus (FPS)	muisti max. (GB)
TensorFlow	59,55	80,44	6,8	3,4
ONNX Resnet-50	57,97	79,14	-	-
TensorRT 32-bittinen liukuluku	57,97	79,14	20,4	3,4
TensorRT 16-bittinen liukuluku	57,98	79,18	37,2	3,4

Taulukkoon 5 on koottu TensorRT:n avulla optimoitua päättelyä Resnet-50 arkkitehtuurilla käyttäen ONNX-mallin rajoitteista johtuen eräkokoa 1. TensorRT-malleille tarkkuudet on laskettu käyttäen TensorRT:n Python-rajapintaa ja päättelynopeudet käyttäen trtexec-työkalua. Taulukon 5 TensorFlow-rivi käyttää taulukossa 1 esitettyä Keras-rajapinnan Resnet-50-mallia, joka saavuttaa hieman paremman tarkkuuden tällä data-aineistolla verrattuna ONNX-malliin. TensorFlow-mallille on mitattu päättelynopeus eräkokoa 1 ja TensorFlow:ta käyttäen.

TensorRT:llä optimoitujen mallien tarkkuus on sama tai jopa suurempi kuin alkuperäisen ONNX-mallin tarkkuus. Kvantifiointi ei siis tässä tapauksessa ole vaikuttanut tarkkuuteen negatiivisesti. TensorRT-mallien päättelynopeus on huomattavasti korkeampi kuin vastaavan ResNet-50-arkkitehtuurin muut mittaukset. Työssä saatu päättelynopeuden tulos TensorRT:n 16-bittistä liukulukukvantifiointia käytettäessä on lähes sama kuin Nvidian ilmoittama lukema Jetson Nano -järjestelmälle [55].

TensorRT:n avulla saavutettiin huomattava päättelynopeuden kasvu tarkkuuden pysyessä samana. Sen avulla ResNet-50:nen tapaista melko suurta ja korkean tarkkuuden saavuttavaa kuvantunnistusneuroverkkoa kyetään käyttämään Jetson Nanon kaltaisessa sulautetussa järjestelmässä esimerkiksi reaaliaikaisen videosyötteen käsittelyyn.

5. YHTEENVETO

Yhä useammat tahot ottavat neuroverkkoja käyttöön osana omia sovelluksiaan, joka asettaa uusia vaatimuksia myös neuroverkoille esimerkiksi laskennan tehokkuuden kannalta. Neuroverkkolaskentaa varten kehitetty erityinen laitteisto ja optimointimenetelmät mahdollistavat neuroverkkojen käytön myös sulautetuissa järjestelmissä tehokkaasti ja nopeasti.

Erilaiset neuroverkkorakenteet mahdollistavat neuroverkkojen käytön moneen erilaiseen tarkoitukseen ja rakenteen arkkitehtuurin muutokset tuovat joustavuutta tarvittavan laskentatehon ja vaaditun tarkkuuden välille. Opetuksessa neuroverkon miljoonien parametrien optimointia varten tarvitaan usein suuri määrä opetusdataa, jotta neuroverkon tulokset yleistyvät myös opetusdataan kuulumattomalle datalle. Sulautettujen järjestelmien vaatimukset esimerkiksi reaaliaikaisuudessa, laskentakapasiteetissä tai energiankulutuksessa asettavat rajoitteita neuroverkkojen käyttöön. Laskennallisesti raskas opetus voidaan usein suorittaa paljon laskentatehoa omaavassa järjestelmässä kehityksen yhteydessä, sillä neuroverkon painot ovat sovelluskohtaisia. Päättelyn suorittamiseksi nopeasti, järjestelmän on kyettävät suorittamaan suuri määrä matriisi- ja vektorilaskentaa, sekä mahdollisesi käsittelemään suurta määrää sisääntulodataa.

Työssä tutkittiin neuroverkon päättelyvaiheen laskennan optimointia sulautetussa järjestelmissä Nvidian Jetson Nano -järjestelmällä, joka on tarkoitettu koneoppimissovellusten toteuttamiseen keveyttä ja rajattua energiankulutusta vaativiin järjestelmiin. Jetson Nanon matalamman tehotilan käyttäminen todettiin työn kokeissa tehokkaaksi tavaksi rajoittaa järjestelmän energiankulutusta päättelyn hidastuessa vain noin 30 %.

Tutkimuksessa huomattiin, että neuroverkon arkkitehtuurin valinnalla kyetään vaikuttamaan laajalla skaalalla tulosten tarkkuuteen, sekä vastapainoisesti päättelynopeuteen. Optimaalinen eräkoko kyettiin määrittämään kokeilemalla laitteistolle, jolloin rinnakkaisuudesta saatiin parannusta päättelynopeuteen noin 30 %. Ohjelmallisten optimointien avulla saavutettiin kokeissa jopa 5-kertainen päättelynopeus.

Työn kokeellisen osuuden toteutuksessa jouduttiin laitteiston ja ohjelmistojen rajoitteiden johdosta käyttämään useaa eri rajapintaa mittausten suorittamiseen. Tämä heikentää tulosten vertailukelposuutta ja vaikeuttaa tuloksista päätelmien tekemistä tuloksista. Lisäksi muistinkulutuksen mittaukset osoittautuivat hyvin epävarmoiksi, sillä korkean abstraktion ohjelmistoja käytettäessä muistinkulutukseen ei voida vaikuttaa.

LÄHTEET

[1] S. Russell, P. Norvig, Artificial intelligence a modern approach, 3rd edition, Pearson, Boston, 2016.

[2] S. Mittal, A Survey on optimized implementation of deep learning models on the NVIDIA Jetson platform, Journal of Systems Architecture, Vol.97, 2019, pp. 428–442, Saatavissa: https://doi.org/10.1016/j.sysarc.2019.01.011.

[3] J. Patterson, A. Gibson, Deep learning: a practitioner's approach, 1st edition, O'Reilly Media, Beijing, China, 2017.

[4] W. S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The bulletin of mathematical biophysics Vol.5, 1943, pp. 115–133.

[5] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, Advances in Neural Information Processing Systems, Vol.2, 2012, pp.1097–1105.

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, Research Highlights: ImageNet Classification with Deep Convolutional Neural Networks, Communications of the ACM, Vol.60, No. 6, 2017, pp. 84–90.

[7] R. M. Balabin, E. I. Lomakina, Neural network approach to quantum-chemistry data: Accurate prediction of density functional theory energies, The Journal of Chemical Physics, Vol.131 No.7, 2009, pp. 074104–8, Saatavissa: https://doi.org/10.1063/1.3206326.

[8] TensorFlow, Why TensrorFlow, verkkosivu, Saatavilla: https://www.tensorflow.org/about, Haettu 4.11.2021.

[9] B. Ramsundar, RB. Zadeh, TensorFlow for deep learning : from linear regression to reinforcement learning, 1st edition, Beijing, China, O'Reilly Media, 2018.

[10] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016, Saatavilla: http://www.deeplearningbook.org

[11] P. Ratan, What is the Convolutional Neural Network Architecture?, verkkosivu, Analytics Vidhya, 28.10.2020, Saatavissa: https://www.analyticsvidhya.com/blog/2020/10/what-is-the-convolutional-neural-network-architecture/, Haettu 5.11.2021.

[12] P. Radhakrishnan, Image Captioning in Deep Learning, verkkosivu, towards data science, 29.9.2017, Saatavissa: https://towardsdatascience.com/image-captioning-in-deep-learning-9cd23fb4d8d2, Haettu 5.11.2021.

[13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative Adversarial Nets, Universite de Montreal, 2014.

[14] C. Shorten, T.M. Khoshgoftaar, A survey on Image Data Augmentation for Deep Learning, Journal of Big Data, Vol.6, 2019, pp. 1–48, Saatavissa: https://doi.org/10.1186/s40537-019-0197-0.

[15] G.M. Weiss, F. Provost, Learning When Training Data are Costly: The Effect of Class Distribution on Tree Induction, The Journal of artificial intelligence research, Vol. 19, 2003, pp.315–54.

[16] Wikipedia, List of datasets for machine-learning research, verkkosivu, 30.11.2021, Saatavissa: https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research, Haettu 1.12.2021.

[17] ImageNet, About ImageNet, 11.3.2021, verkkosivu, Saatavissa: https://image-net.org/about.php, Haettu 1.12.2021.

[18] P. Marwedel, Embedded System Design, 1st edition, New York, NY, Springer US, 2006.

[19] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, S. Wermter, Continual lifelong learning with neural networks: A review, Neural Networks, Vol. 113, 2019, pp. 54–71, Saatavissa: https://doi.org/10.1016/j.neunet.2019.01.012.

[20] N. Deepika, V. V. Sajith Variyar, Obstacle classification and detection for vision-based navigation for autonomous driving, International Conference on Advances in Computing, Communications and Informatics (ICACCI), 2017, pp. 2092-2097.

[21] D. Tomè, F. Monti, L. Baroffio, L. Bondi, M. Tagliasacchi, S. Tubaro, Deep Convolutional Neural Networks for pedestrian detection, Signal Processing: Image Communication, Vol. 47, 2016, pp. 482–489, Saatavissa: https://doi.org/10.1016/j.image.2016.05.007.

[22] R. Krishnamurthi, A. Kumar, D. Gopinathan, A. Nayyar, B. Qureshi, An Overview of IoT Sensor Data Processing, Fusion, and Analysis Techniques, Sensors, Vol. 20, No. 21, 2020, Saatavissa: https://doi.org/10.3390/s20216076.

[23] P. Ghazi, A. P. Happonen, J. Boutellier, H. Huttunen, Embedded Implementation of a Deep Learning Smile Detector, 2018.

[24] R. Yazdani, A. Segura, J. Arnau, A. Gonzalez, Low-Power Automatic Speech Recognition Through a Mobile GPU and a Viterbi Accelerator, IEEE Micro, Vol. 37, No. 1, 2017, pp. 22–29.

[25] S. Bianco, R. Cadene, L. Celona, P. Napoletano, Benchmark Analysis of Representative Deep Neural Network Architectures, IEEE access, Vol. 6, 2018, pp. 64270–64277.

[26] K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, University of Oxford, ICLR, 2015.

[27] K. He, X. Zhang, S. Ren, J. Sun, Deep Residual Learning for Image Recognition, Microsoft Research, 2015.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, Google, 2017

[29] V. Hegde, S. Usmani, Parallel and Distributed Deep Learning, Stanford University, 2016

[30] S. Ioffe, C. Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Google, 2015

[31] Semiconductor Engineering, Do Large Batches Always Improve Neural Network Throughput?, verkkosivu, G. Tate, 6.6.2019, Saatavilla: https://semiengineering.com/do-large-batches-always-improve-neural-network-throughput/, Haettu 17.12.2021.

[32] C.F. Sabottke, B. M. Spieler, The Effect of Image Resolution on Deep Learning in Radiography, Radiology Artificial intelligence, Vol. 2, No. 1, 2020, pp. 190015–.

[33] Tensorflow, High performance inference with TensorRT Integration, verkkosivu, 13.6.2019, Saatavilla: https://blog.tensorflow.org/2019/06/high-performance-inference-with-TensorRT.html. Haettu 5.12.2021.

[34] R. Krishnamoorthi, Quantizing deep convolutional networks for efficient inference: A whitepaper, 2018, Saatavilla: https://arxiv.org/abs/1806.08342.

[35] Towards data science, R. Bandaru Pruning Neural Networks, verkkosivu, 1.9.2020, Saatavilla: https://towardsdatascience.com/pruning-neural-networks-1bb3ab5791f9, Haettu 5.12.2021.

[36] Machine Learning Mastery, J. Brownlee, A Gentle Introduction to Sparse Matrices for Machine Learning, 13.3.2018, Saatavilla: https://machinelearningmastery.com/sparse-matrices-for-machine-learning/, Haettu 5.12.2021.

[37] Nvidia, NVIDIA TensorRT, Saatavilla: https://developer.nvidia.com/tensorrt#features, Haettu 5.12.2021.

[38] Nvidia, TensorRT Documentation, verkkosivu, 22.11.2021, Saatavilla: https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/index.html, Haettu: 7.12.2021.

[39] Nvidia, Accelerating Inference In TF-TRT User Guide, verkkosivu, 12.11.2021, Saatavilla:, Haettu 7.12.2021.

[40] Nvidia, What's the Difference Between a CPU and a GPU?, verkkosivu, 15.12.2009, Saatavilla: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/. Haettu 29.11.2021.

[41] W. Yang, K Li, K Li, A hybrid computing method of SpMV on CPU–GPU heterogeneous computing systems, Journal of parallel and distributed computing, Vol. 104, 2017, pp. 49–60.

[42] Google, Google supercharges machine learning tasks with TPU custom chip, verkkosivu, 27.6.2017, Saatavilla: https://cloud.google.com/blog/products/ai-machine-learning/google-supercharges-machine-learning-tasks-with-custom-chip, Haettu: 29.11.2021.

[43] Google, Edge TPU, verkkosivu Saatavilla: https://cloud.google.com/edge-tpu, Haettu 29.11.2021.

[44] Nvidia, Embedded Systems with Jetson, verkkosivu, 2021. Saatavissa: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/, Haettu 3.11.2021.

[45] Nvidia, Jetson Nano Developer Kit, verkkosivu, 2021. Saatavissa: https://developer.nvidia.com/embedded/jetson-nano-developer-kit, Haettu 3.11.2021.

[46] Työn lähdekoodit, Saatavilla: https://github.com/teerol/kandi.

[47] TensorFlow imagenet_v2, verkkosivu, 2.12.2021, Saatavilla: https://www.tensorflow.org/datasets/catalog/imagenet_v2, Haettu 6.12.2021.

[48] TechSmith, Frame Rate: A Beginner's Guide, verkkosivu, Saatavilla: https://www.techsmith.com/blog/frame-rate-beginners-guide/, Haettu: 7.12.2021.

[49] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L.-C. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2019

[50] G. Huang, Z. Liu, L. van der Maaten, K. Q. Weinberger, Densely Connected Convolutional Networks, 2018.

[51] TensorFlow, Deploy machine learning models on mobile and IoT devices, verkkosivu, Saatavilla: https://www.tensorflow.org/lite, Haettu 7.12.2021.

[52] Keras, Keras Applications, verkkosivu, Saatavilla: https://keras.io/api/applications/, Haettu 7.12.2021.

[53] Nvidia, Power Management for Jetson Nano and Jetson TX1 Devices, verkkosivu, 3.8.2021, Saatavilla: https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_nano.html#wwpID0E06P0HA, Haettu 7.12.2021.

[54] Gitlub, ONNX Model Zoo, verkkosivu, 23.11.2021, Saatavilla: https://github.com/onnx/models/tree/master/vision/classification/resnet, Haettu 10.12.2021.

[55] Nvidia, Jetson Nano: Deep Learning Inference Benchmarks, verkkosivu, Saatavilla: https://developer.nvidia.com/embedded/jetson-nano-dl-inference-benchmarks, Haettu 10.12.2021.