Tampereen yliopisto

Kai Jylkkä

NEUROVERKKOPOHJAISEN AJONEUVOTUNNISTIMEN TOTEUTUS SOC-ALUSTALLA

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Marraskuu 2021

TIIVISTELMÄ

Kai Jylkkä: Neuroverkkopohjaisen ajoneuvotunnistimen toteutus SoC-alustalla
Diplomityö
Tampereen yliopisto
Tietotekniikka, DI
Marraskuu 2021

Konenäkösovellusten kehitys on johtanut yhä monimutkaisempien ja syvempien neuroverkkosovellusten kehitykseen. Konvoluutioneuroverkkosovellukset, kuten luokittelijat ja kohteenilmaisimet vaativat paljon laskentakykyä ja muistinkäyttöä. Tämä asettaa suorituskykyvaatimuksia myös suorittavalle laitteistolle.

Usein ei ole käytännöllistä suorittaa neuroverkon laskentaa pilvipalveluissa tai tehokkailla tietokoneilla, vaan neuroverkkosovellusta on tarve käyttää sulautetuissa laitteissa. Nykypäivän mobiililaitteissa onkin tehokkaita suorittimia, kuten GPU ja DSP, jotka mahdollistavat laskennan rinnakkaistamisen ja siten soveltuvat hyvin neuroverkkolaskentaan.

Tässä työssä mitattiin ja arvioitiin mobiililaitteen soveltuvuutta neuroverkkopohjaisen ajoneuvoluokittelijan laitteessa tapahtuvaan suorittamiseen. Mittauksissa mitattiin erityisesti laskentanopeuksia ja tehonkulutusta eri prosessoreilla ja eri neuroverkko-ohjelmakirjastojen toteutuksilla. Laskentanopeuksien osalta mitattiin neuroverkon suorittamiseen kuluvaa aikaa. Tehonkulutusta mitattiin sovelluksen jatkuvan suorittamisen aikana. Mittauksia suoritettiin prosessorin eri kiihdytinyksiköillä ja ohjelmakirjastoilla. Neuroverkkosovelluksena mittauksissa käytettiin MobileNet SSD-kohteenilmaisinta ja laitteena Snapdragon 835-prosessorilla varustettua kehitysalustaa.

Parhaaksi suoritusnopeudeksi mittauksissa saatiin 21,4 inferenssiä sekunnissa GPU:lla. DSP:n ja CPU:n suoritusnopeudet olivat 13,1 ja 3,2 inferenssiä sekunnissa. Energiatehokkain laskentayksikkö oli DSP noin 560 mW kulutuksella. CPU:lla Tensoflow ohjelmakirjasto oli sekä nopein ja energiatehokkain, mutta silti huomattavast muita laskentayksiköitä hitaampi ja suurikulutuksisempi. Muilla laskentayksiköillä Qualcommin oman Snapdragon Processing Engine -kirjaston (SNPE) toteutukset olivat nopeimmat ja energiatehokkaimmat. GPU:lla SNPEn suorituskykyprofiilien suoritusnopeudet sijoittuvat välille 9,2-21,4 inferenssiä sekunnissa ja tehonkulutukset välille 657-1707 mW. DSP:llä vastaavat arvot olivat 9,1-13,1 inferenssiä sekunnissa ja 559-726 mW.

Mittaustulokset osoittavat, että käytettäessä prosessoriarkkitehtuuria tukevaa ohjelmakirjastoa mobiililaitteella on mahdollista suorittaa neuroverkkosovellusta reaaliajassa akkukäyttöisesti useiden tuntien ajan. Näin saadaan huomattava parannus sekä suorituskyvyssä että tehokulutuksessa GPU:lla ja DSP:llä verrattuna CPU:hun ja yleiskäyttöisiä neuroverkkokirjastoja käyttäviin toteutuksiin. Työn tuloksia voi soveltaa myös muissa sulautetuissa alustoissa.

Avainsanat: neuroverkko, konvoluutioneuroverkko, luokittelija, kohteenilmaisin, konenäkö, TensorFlow, Snapdragon neural processing engine, Android

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck -ohjelmalla.

ABSTRACT

Kai Jylkkä: Implementation of a neural network-based vehicle classifier on a SoC platform
Master's thesis
Tampere University
Computer Sciences
November 2021

The evolution of computer vision has led to more complex and deeper neural network applications. Convolutional neural network applications, such as image classifiers and object detectors, are very computation and memory intensive. This also sets high standards for hardware to meet the performance requirements.

It is not always practical to accelerate the computation of the neural network in cloud services or use high performance desktop computers, but there is a need to use computer vision applications in embedded systems. Modern mobile devices have often powerful processors, such as GPU or DSP, that are capable of massively parallel processing and therefore suitable for accelerating neural network computation.

In this thesis we measured and evaluated suitability of mobile device to perform neural network-based vehicle classifier locally in the device. Measurements focus mainly on inference times, throughput and energy efficiency in different processors of the device and implementations of different neural network software frameworks. As a vehicle classifier we used MobileNet SSD object detector and hardware unit was Snapdragon 835 powered development board.

The best execution speed, 21,4 fps, was measured with GPU. Execution speed for DSP and CPU were 13,1 fps and 3,2 fps. Most power efficient processing unit was DSP with 560 mW consumption. With CPU Tensorflow framework was the fastest and the most power efficient, but still considerably slower and consumed more power than other processing units. For other processing units Qualcomm's Snapdragon Processing Engine was the fastest and the most power efficient. When using GPU with SNPE the execution speed for different performance profiles were between 9.21-21.4 fps and for power consumption 657-1707 mW and for DSP same values were between 9.1-13.1 fps and 559-726 mW.

As a conclusion, it is possible to get good real time performance and battery life, if the application uses software framework that supports processor architecture. The performance and energy efficiency were considerably better with GPU and DSP compared to CPU and generic neural network frameworks. Results of this thesis can be applied to other embedded platforms.

Keywords: neural network, convolutional neural network, classifier, object detector, computer vision, TensorFlow, Snapdragon neural processing engine, Android

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

ALKUSANAT

Tämä opinnäytetyö on tehty Tampereen yliopiston tietokonetekniikan diplomi-insinöörin tutkintoa varten. Työhön liittyvät mittaukset tein keväällä 2019 mutta työn kirjoitus olikin sitten minulle haastavampi ja pidempi prosessi ja sen tein vuosien 2019 - 2021 aikana.

Haluaisin kiittää ohjaajiani Tenure track -tutkija Jani Boutellieria, jonka ohjauksessa aloitin tämän opinnäytetyön, ja professori Timo Hämäläistä, jonka ohjauksessa työn saatoin loppuun. Kiitokset myös tarkastajille professori Heikki Huttuselle ja yliopisto-opettaja Matti Haavistolle. Kiitos tohtorikoulutettava Mir Khanille MobileNet SSD -kohteenilmaisimen OPenCL toteutuksesta. Haluaisin kiittää tutkimusrahoituksesta myös Suomen Akatemiaa, joka on osittain rahoittanut työtäni (hanke 309903 'CoEfNet').

Tampereella, 25. marraskuuta 2021

Kai Jylkkä

SISÄLLYSLUETTELO

Kuvaluettelo . . . vi
Taulukkoluettelo . . . vii
Ohjelma- ja algoritmiluettelo . . . viii
Lyhenteet ja merkinnät . . . ix
1 Johdanto . . . 1
2 Konvoluutioneuroverkot . . . 2
2.1 Yleistä neuroverkoista . . . 2
2.2 Konvoluutioneuroverkot . . . 3
2.3 Luokittelijat . . . 4
2.3.1 MobileNet . . . 5
2.4 Kohteenilmaisimet . . . 5
2.4.1 Single Shot Detector . . . 7
3 Kehitystyökalut neuroverkoille . . . 8
3.1 Yleiskäyttöiset kehitystyökalut . . . 8
3.1.1 TensorFlow . . . 9
3.1.2 PyTorch . . . 9
3.1.3 Keras . . . 10
3.2 Neuroverkkokirjastot Androidille . . . 11
3.2.1 TensorFlow . . . 11
3.2.2 Snapdragon Neural Processing Engine . . . 12
4 Mittaukset . . . 14
4.1 Käytetty laitealusta . . . 14
4.2 Käytetyt ohjelmistot ja algoritmit . . . 16
4.2.1 Snapdragon Profiler . . . 16
4.2.2 Trepn Power Profiler . . . 17
4.2.3 COCO - Common Objects in Context . . . 18
4.2.4 MobileNet SSD . . . 18
4.3 Mittausjärjestelyt . . . 19
4.3.1 Tarkkuuden mittaukset . . . 20
4.3.2 Laskentanopeuden mittaukset . . . 21
4.3.3 Tehonkulutuksen mittaukset . . . 22
4.3.4 Lämpötilanmittaus . . . 24
5 Mittaustulokset . . . 25
5.1 Tarkkuudet . . . 25
5.2 Laskentanopeudet . . . 26
5.3 Tehonkulutukset . . . 31

5.4 Lämpötilanmittaukset 35
6 Yhteenveto 37
Lähdeluettelo 39
Liite A Liite: Konvoluutio-operaatioiden laskenta-ajat 42

KUVALUETTELO

2.1 Neuroverkko koostuu yleensä sisäänmeno-, ulostulo- ja yhdestä tai useammasta piilokerroksesta. 3
2.2 Konvoluutio-operaation suodatin, jonka avulla tunnistetaan kuvasta tai edellisen kerroksen piirrekartoista piirteitä. 4
2.3 MobileNetin konvoluutiokerros on jaettu kahteen erilliseen osaan: depthwise- ja pointwise-operaatioihin. 6
2.4 IoU tarkoittaa neuroverkolta saadun tunnistuksen (A) ja annotaation (B) koordinaattien muodostamien suorakulmioiden leikkauksen pinta-alan suhdetta niiden unionin pinta-alaan. 6
2.5 SSD kohteenilmaisimen perusarkkitehtuuri. 7

3.1 TensorFlow-graafi koostuu solmuista, jotka kuvaavat erilaisia operaatioita ja muuttujia. Kaarilla kuvataan datansiirtoja. [1]. 10
3.2 SNPE:n työvuo. 12

4.1 Open-Q 835 HDK laitealusta. Alustaan kytketty kameramoduuli oikealla. . 14
4.2 Kuvaesimerkit helposta (vasen) ja vaikeasta (oikea) kuvasarjoista. Ylempiin kuviin piirretty suorakulmiot annotaatioissa määritellyille kohteille ja alapuolisiin TensorFlow:n verkosta saaduille tunnistuksille. 21

5.1 Toteutuksien laskentanopeudet pylväsdiagrammina. 27
5.2 OpenCL-kernelien laskenta-aikojen suhteelliset osuudet. 30
5.3 Energiankulutukset eri toteutuksille. 32
5.4 Suoritusnopeuden suhde inferenssin suorittamiseen kuluvaan energiaan. . 33
5.5 Taulukon 5.6 akun kestoajat pylväsdiagrammina. 34
5.6 Keskusprosessorin lämpötilan ja laskentanopeuksien mittaustulokset yhtäjaksoisen inferenssin suorittamisen ajalta. Lämpötilat on esitetty yhtäjaksoisella viivalla ja laskentanopeudet katkoviivalla. 35

TAULUKKOLUETTELO

4.1 Mittauksiin käytetyt ohjelmistot. 17

5.1 Eri neuroverkkototeutuksien tarkkuudet 25
5.2 Eri neuroverkkototeutuksien kokonaislaskentanopeudet ja -ajat. 26
5.3 Laskentanopeuksien mittaustulokset eri luokkien ja tunnistusten määrille. Vasemmalla tulokset 1 ja 90 luokan verkoille ja oikealla kuville, joissa tunnistuksia 1 ja 27 kappaletta. 28
5.4 Konvoluutio-operaatioiden yhteenlasketut laskenta-ajat. 29
5.5 Tehon- ja energian kulutukset eri toteutuksille. 31
5.6 Akunkestoajat järjestelmän kokonaistehonkulutuksella *h* ja pelkästään inferenssin tehonkulutuksilla *h'*. 33

6.1 MobileNet SSD -kohteenilmaisimen eri toteutuksien ja suorituskykyprofiilien laskentanopeudet (fps) ja tehonkulutukset (mW). 37

A.1 MobileNet SSD -neuroverkon konvoluutio-operaatioiden OpenCL-kernelien laskenta-aikoja SNPE- ja OpenCL-toteutuksille. 42

OHJELMA- JA ALGORITMILUETTELO

4.1 Laskenta-aikojen mittauksessa käytetyn *nanoTime()* -metodin resoluution määritykseen käytetty menetelmä. 23

LYHENTEET JA MERKINNÄT

AP	tarkkuuden keskiarvo (engl. average precision), yleinen neuroverkon tarkkuuden ilmaisemiseen käytetty metriikka
API	ohjelmointirajapinta (engl. application programming interface)
COCO	avoin kuvasarja kohteenilmaisimille (Common Objects in Context)
CPU	keskusprosessori (engl. central processing unit)
DSP	digitaalinen signaaliprosessori (engl. digital signal processor)
GPU	grafiikkaprosessori (engl. graphics processing unit)
HVX	digitaalisen signaaliprosessorin vektorilaskentayksikkö (engl. Hexagon Vector eXtensions)
IoU	leikkauksen suhde unioniin (engl. intersection over union)
MAC	kertolasku summaus -operaatio (engl. multiply-accumulate)
ReLU	neuroverkoissa käytettävä aktivaatio-operaatio (engl. rectified linear unit)
SNPE	neuroverkkokirjasto Snapdragon-alustoille (Snapdragon Neural Processing Engine)
VLIW	suoritin arkkitehtuuri, joka on suunniteltu suorittamaan useita käskyjä rinnakkain (engl. very long instruction word)

1 JOHDANTO

Konenäkösovellusten kehitys on ollut nopeaa viime vuosina ja niiden kehitystä on ajanut lähinnä tarve kehittää tarkempia ja ominaisuuksiltaan parempia algoritmeja. Tämä kehitys on tullut kuitenkin suorituskyvyn kustannuksella. Erityisesti visuaaliset neuroverkkosovellukset, kuten luokittelijat ja kohteenilmaisijat, vaativat paljon laskentakykyä ja muistinkäyttöä. Neuroverkkosovellusten suoritus laitteessa vaatii paljon laskentakykyä, ja ne ovatkin keskusprosessoreilla suoritettuna usein hyvin hitaita.

Ratkaisuina laskentakyvyn tarpeeseen on tarjolla erilaisia pilvipalveluita, joihin neuroverkon laskenta voidaan siirtää. Lisäksi neuroverkon laskentaa voidaan kiihdyttää esimerkiksi grafiikkaprosessoreilla. Ne soveltuvat hyvin tähän tehtävään, sillä neuroverkot koostuvat lähinnä matriisilaskennasta, jota on helppo rinnakkaistaa. Aina laskennan siirtäminen pilveen ei kuitenkaan ole mahdollista, eikä tehokkaiden tietokoneiden käyttäminen kaikissa käyttötilanteissa ole järkevää. Nykypäivän älylaitteet, kuten mobiilipuhelimet, ovat laskentakyvyltään tehokkaita ja yleensä varustettu myös grafiikkaprosessoreilla. Lisäksi uusista prosessoriyksiköistä löytyy usein myös muita prosessoreja ja yksiköitä, kuten esimerkiksi digitaalinen signaaliprosessori.

Tämän opinnäytetyön tarkoituksena on tutkia, soveltuuko mobiililaiteen kaltainen sulautettu laite ajoneuvontunnistussovelluksen suorittamiseen paikallisesti laitteessa. Tämä kysymys voidaan jakaa useampaan tarkentavaan alakysymykseen. Riittääkö mobiililaitteen suorituskyky monimutkaisen neuroverkon laskentaan? Soveltuvatko mobiililaitteen laskennan kiihdytinyksiköt, kuten grafiikka- ja digitaalinen signaaliprosessori, neuroverkon kiihdyttimiksi? Mahdollistaako eri suoritinyksikköjen tehonkulutukset riittävän pitkän akunkeston mobiilikäyttöön? Laitteen soveltuvuuden lisäksi voidaan myös pohtia, mikä kohteenilmaisinarkkitehtuuri olisi sopivin tällaiseen käyttöön.

Luvussa 2 esitellään ensin yleisellä tasolla neuroverkkojen ja konvoluutioneuroverkkojen rakennetta. Lisäksi käydään läpi työn kannalta oleellisia neuroverkkosovelluksia. Seuraavaksi luvussa 3 tutustutaan yleiskäyttöisiin neuroverkkojen kehitystyökaluihin. Lopuksi esitellään neuroverkkokirjastoja Android-mobiilikäyttöjärjestelmälle.

Luvussa 4 esitellään työhön liittyvät mittausjärjestelyt. Luvun alussa käydään läpi työssä käytetty laitealusta ja työhön liittyvät ohjelmistot ja algoritmit. Seuraavaksi esitetään itse mittausjärjestelyt ja mittausten toteutus. Mittausjärjestelyjen jälkeen luvussa 5 esitetään mittaustulokset. Lopuksi luvussa 6 tehdään yhteenveto mittauksista ja pohditaan mittaustuloksia tutkimuskysymysten näkökulmasta.

2 KONVOLUUTIONEUROVERKOT

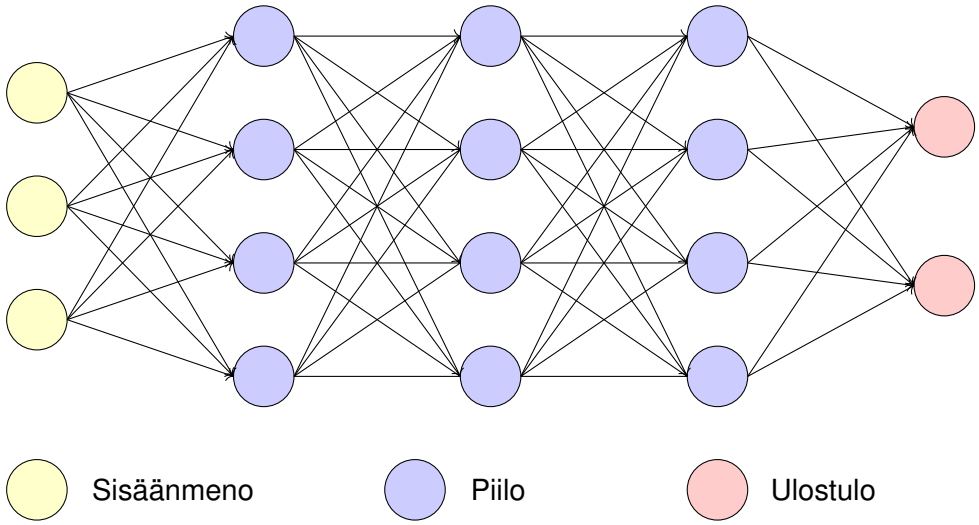
2.1 Yleistä neuroverkoista

Keinotekoisten neuroverkkojen (engl. artificial neural networks, ANN) rakenne pyrkii jäljittelemään aivojen rakennetta. Aivot koostuvat hermosoluista eli neuroneista ja hermosolujen välisistä yhteyksistä eli synapseista. Neuronit muodostavat monimutkaisen verkon, jossa yksittäisellä neuronilla voi olla jopa tuhansia synapseja muihin neuroneihin. Keinotekoisten neuroverkkojen rakenteessa ei kuitenkaan pyritä samanlaiseen monimutkaisuuteen, vaan ne koostuvat yleensä useasta neuronikerroksesta ja niiden välisistä yhteyksistä. Ensimmäisen aivojen toimintaan perustuvan laskennallisen mallin neuroverkoista loivat McCulloch ja Pitts[21] jo vuonna 1943. Neuroverkot ovat yksi koneoppimisen menetelmä, jossa algoritmia ei ohjelmoida suorittamaan mitään tehtävää vaan oppimaan sisään syötetystä datasta haluttu toiminnallisuus. Neuroverkon toiminnallisuutta ei siis hallita ohjelmakäskyillä vaan datalla, jolla neuroverkko koulutetaan.

Neuroverkko koostuu yleensä sisääntulo- ja ulostulokerroksesta sekä niiden välisestä yhdestä tai useammasta piilokerroksesta (Kuva 2.1). Neuroverkkoa, jossa on useampia piilokerroksia, kutsutaan syväksi neuroverkoksi (engl. deep neural network, DNN).

Neuroverkon kerrokset koostuvat neuroneista ja kerrosten välissä kulkevat neuronien väliset kytkennät eli synapsit. Jokaiselle synapsille on määritelty painokerroin. Neuronissa lasketaan sisääntulokytkentöjen ja niiden painokertoimien avulla neuronin ulostuloarvo, joka välitetään seuraavalle neuronille. Käytännössä tämä tarkoittaa yleensä matriisikertolaskua, jossa sisääntulo- ja painokerroinmatriisit kerrotaan keskenään. Matriisikertolaskun jälkeen tulokselle suoritetaan vielä aktivaatiofunktio, jolla saadaan lopullinen ulostuloarvo. Yksinkertaisimmillaan aktivaatiofunktio voisi olla funktio, jolla kynnysarvon ylittävät arvot saavat arvon 1, joka tarkoittaisi, että neuroni on aktiivinen, ja vastaavasti alle jäävät arvot saavat arvon 0, jolloin neuroni ei ole aktiivinen. Yleensä aktivaatiofunktio on kuitenkin monimutkaisempi epälineaarinen funktio, kuten *Sigmoid*- tai *ReLU*-funktio. Aktivaatiofunktiolla on tärkeä tehtävä neuroverkossa. Ilman sitä neuronin laskenta olisi vain lineaarifunktio. Epälineaarinen aktivaatiofunktio lisää verkon kompleksisuutta ja täten parantaa verkon koulutuksen tehokkuutta ja stabiilisuutta.

Yleensä tällaisia matriisikertolaskuun perustuvia kerroksia kutsutaan täysin kytketyiksi kerroksiksi (engl. *fully connected* tai *dense*). Tällaisiin kerroksiin perustuvat neuroverkot ovat hyvin monikäyttöisiä ja soveltuvat monenlaisiin erilaisiin tehtäviin. Näissä verkoissa



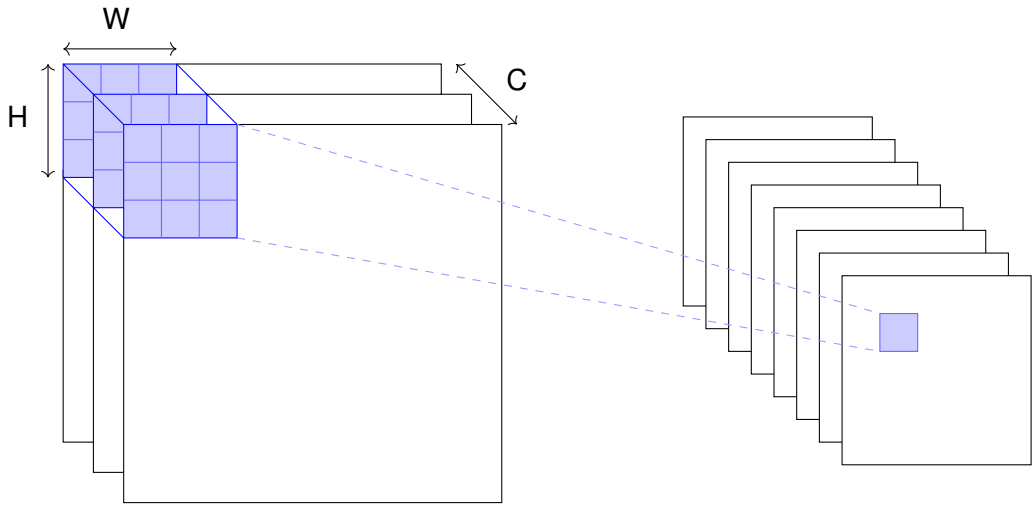
Kuva 2.1. *Neuroverkko koostuu yleensä sisäänmeno-, ulostulo- ja yhdestä tai useammasta piilokerroksesta.*

ongelma on se, että verkon syvyyttä kasvatettaessa parametrien ja laskentatoimitusten lukumäärä kasvaa nopeasti hyvin suureksi, minkä takia niiden suorittamiseen tarvitaan hyvin tehokkaita laskentayksikköjä. Edellä mainitun lisäksi on olemassa erilaisia neuroverkkoja kuten konvoluutioneuroverkko (engl. convolutional neural network, cnn) ja *recurrent*-neuroverkot (rnn). *Rnn* soveltuu hyvin esimerkiksi tekstin tai puheen tunnistukseen ja *cnn* visuaaliseen, kuten kuvasta tai videosta tapahtuvaan tunnistukseen. Yhteistä *rnn*- ja *cnn*-toteutuksille on, että niillä saadaan vähennettyä parametrien ja laskutoimitusten määrää täysin kytkettyihin kerroksiin perustuvaan neuroverkkoon nähden.

2.2 Konvoluutioneuroverkot

Konvoluutioneuroverkossa kerroksen matriisilaskenta on korvattu konvoluutio-operaatiolla. Konvoluutiokerroksessa kuva käydään läpi useilla suodattimilla. Suodatin (engl. kernel tai filter) on korkeus(H)- ja leveys(W)-dimensioilla määritelty osa kuvasta, jolle konvoluutio-operaatio suoritetaan (Kuva 2.2). Käytännössä suodatin pitää sisällään konvoluutio- operaatiossa käytettävät painokertoimet. Syvyys suuntaan suodattimessa on aina yhtä monta kanavaa (engl. channel, C) kuin on käsiteltävässä kuvassa. Suodattimien avulla kuvasta tai edellisen kerroksen piirrekartoista pyritään tunnistamaan erilaisia piirteitä, kuten esimerkiksi muotoja tai värejä. Yleensä suodattimia on useita erilaisia erilaisten piirteiden tunnistamista varten. Suodattimia kuljetetaan kuvan yli, jolloin sama piirre saadaan tunnistettua kuvan eri kohdista. Tällainen painokertoimien jakaminen vähentää niiden määrää merkittävästi verrattuna täysin kytkettyyn neuroverkkoon. Kun koko kuva on käyty läpi, saadaan piirrekartta kuvasta. Ja lopulta kun konvoluutio-operaatio on suoritettu kaikilla suodattimilla, saadaan kokoelma piirrekarttoja, joka välitetään seuraavalle kerrokselle. Seuraavissa konvoluutiokerroksissa piirrekartoista pyritään vastaavalla tavalla tunnistamaan monimutkaisempia piirteitä ja näin edettäessä neuroverkossa eteenpäin tunniste-

taan aina monimutkaisempia piirteitä ja lopulta haluttuja kohteita kokonaisuudessaan.



Kuva 2.2. *Konvoluutio-operaation suodatin, jonka avulla tunnistetaan kuvasta tai edellisen kerroksen piirrekartoista piirteitä.*

Kuten täysin kytketyissä neuroverkoissa myös konvoluutio-operaatioon yleensä liittyy aktivaatio ennen tuloksen välittämistä seuraavalle kerrokselle. Muita konvoluutioneuroverkoissa yleisesti käytettäviä operaatioita ovat esimerkiksi *pooling*- ja täysin kytketyt kerrokset. *Pooling*-operaation tarkoituksena on pienentää piirrekarttojen korkeus ja leveys dimensioita. Dimensioiden pienentäminen vähentää konvoluutioneuroverkon parametrien määrää ja samalla verkon suorittamiseen tarvittavien laskentaoperaatioiden määrää. Parametrien vähentäminen samalla pienentää myös ylikoulutuksen mahdollisuutta koulutuksen aikana. Täysikytkettyä kerrosta käytetään yleensä neuroverkon lopussa piirrekarttojen muokkaamiseen haluttuun muotoon ulostulokerrokseen.

2.3 Luokittelijat

Yleisesti luokittelija on algoritmi, joka luokittelee tunnistamiaan kohteita etukäteen määrättyihin luokkiin. Visuaaliset luokittelijat ovat yleensä konvoluutioneuroverkkoja. Tunnettuja neuroverkkoihin perustuvia luokittelijoita, ovat esimerkiksi Alexnet [18], Inception[37], MobileNet[14] ja ResNet[12]. Neuroverkkoon perustuvat luokittelijat koulutetaan tunnistamaan määrättyjä luokkia. Luokittelija saa sisääntulona esimerkiksi kuvan, josta se tunnistaa kohteita. Ulostulona luokittelija antaa tunnistetun kohteen luokan ja tunnistuksen pistemäärän eli kuinka varma tunnistus oli.

Yleensä luokittelijoiden toiminnallisuuden mittaamiseen käytettävä mittari on tarkkuus (engl. precision). Lisäksi mittareina voidaan käyttää suoritusnopeutta, parametrien määrää yms. Tarkkuus määritellään positiivisten tunnistusten suhteena kaikkiin tunnistuksiin, eli kuinka suuri osa tunnistuksista oli oikeita. Vertailtaessa neuroverkkoarkkitehtuureja toisiinsa käytetään yleensä tunnettuja datasettejä kuten ImageNet [8]. Nämä datasetit sisältävät verkon koulutukseen käytetyt koulutus- ja validointidatasarjat ja tarkkuuden mää-

ritykseen käytetyn testisarjan. Käytettäessä yleisesti tunnettuja datasettejä saadaan vertailukelpoisia tuloksia, joiden avulla neuroverkkoja voidaan verrata keskenään.

2.3.1 MobileNet

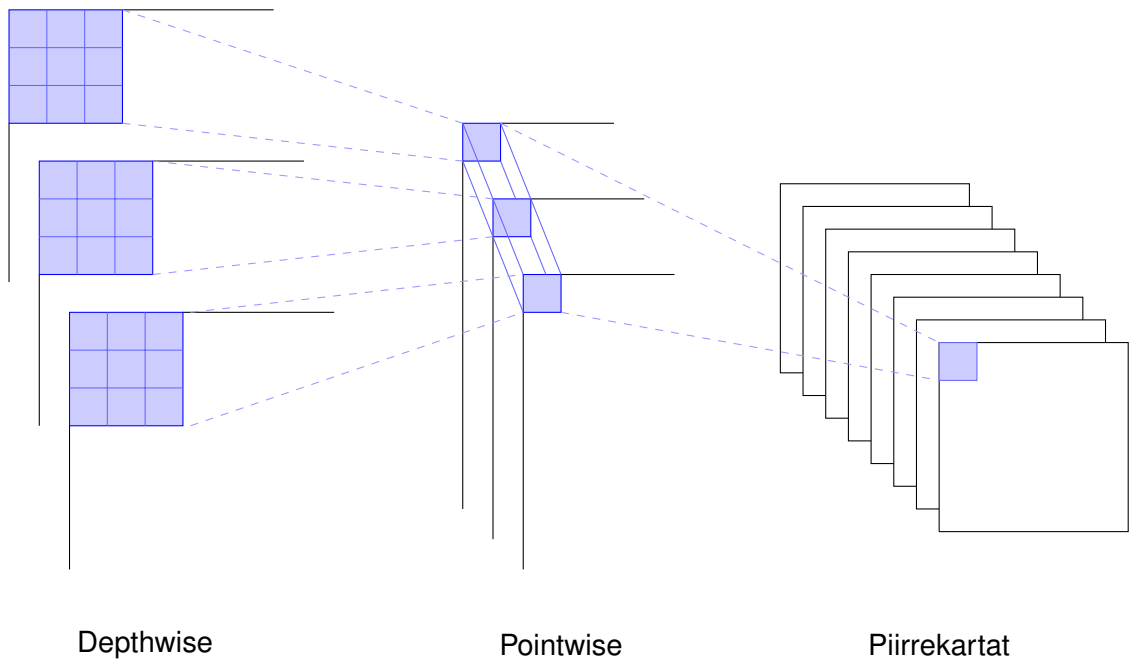
MobileNet on konvoluutioneuroverkkomalli, joka on erityisesti suunnattu mobiili- ja sulautetuille laitteille. MobileNetin arkkitehtuurissa konvoluutiokerrokset on jaettu kahteen erilliseen operaatioon, joka vähentää kerrosten parametrien ja laskentatoimitusten määrää. Tämä parantaa verkon suorituskykyä erityisesti matalan suorituskyvyn laitteissa kuten mobiili- ja sulautetuissa laitteissa.

MobileNetissä [14] konvoluutiokerros on jaettu kahteen osaan, *depthwise*- ja *pointwise*-operaatioihin (Kuva 2.3). *Depthwise*-operaatiossa konvoluutio suoritetaan jokaiselle kanavalle erikseen. Tämän jälkeen suoritetaan *pointwise*-operaatio, jossa edellisen vaiheen *depthwise*-operaation tulokset yhdistetään kanavien yli piirrekartaksi. Kummankin operaation jälkeen tehdään *ReLU*-aktivaatio-operaatio. MobileNetissä *depthwise*-operaatio on kuten normaali 3x3 konvoluutio-operaatio, mutta kanavien konvoluutio-operaatioita ei summata yhteen. *Pointwise*-operaatio on 1x1 konvoluutio-operaatio ja se yhdistää *depthwise*-operaation konvoluutiot yhdeksi piirteeksi. Tällainen konvoluutio-operaation jako kahteen erilliseen operaatioon vähentää tarvittavien parametrien ja laskentakuorman määrää. Howard et al. [14] mukaan jako *depthwise*- ja *pointwise*-operaatioihin vähentää, normaaleilla konvoluutiokerroksilla toteutettuun MobileNetiin verrattuna, parametrien määrää 29,3 miljoonasta 4,2 miljoonaan ja kertolaskusummaus laskutoimitusten määrää 4866 miljoonasta 569 miljoonaan. Laskutoimitusten ja parametrien vähenemisestä huolimatta verkon tarkkuus säilyi silti hyvänä, laskien 71.7 prosentista 70,6 prosenttiin.

2.4 Kohteenilmaisimet

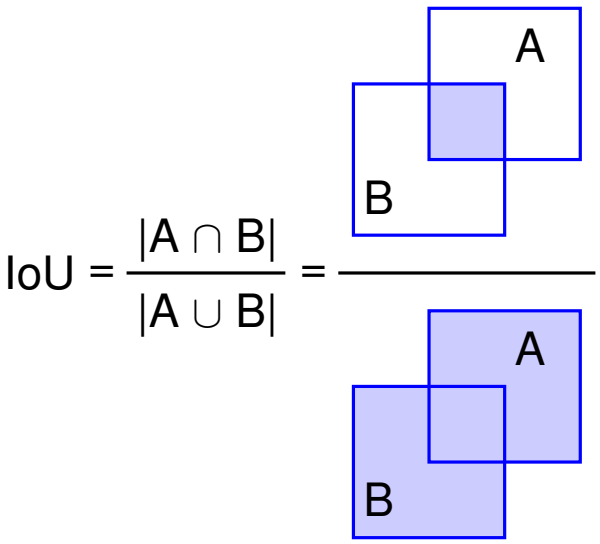
Kohteenilmaisin (engl. object detector) on yleensä konvoluutioneuroverkkoon perustuva algoritmi, joka ilmaisee kohteen luokan lisäksi kohteen sijainnin kuvassa. Yleensä kohteenilmaisijat perustuvat kohteen luokittelijaan, joka luokittelee tunnistetut kohteet. Kohteiden luokituksen jälkeen algoritmi määrittää kohteiden sijainnin kuvassa. Kuvassa 4.2 nähdään esimerkki, jossa kuviin on kohteenilmaisimesta saadun sijaintitiedon perusteella kohteiden ympärille piirretty suorakulmiot. Tunnettuja kohteenilmaisimia ovat SSD [20] , YOLO [30] ja R-CNN:n eri versiot [11]. Näissä kohteenilmaisimissa voidaan usein käyttää erilaisia luokittelijoita käyttökohteesta ja sovelluksesta riippuen. Näin saadaan käyttötarkoituksen parhaiten sopiva ilmaisin. Luokittelijan valinnassa voidaan painottaa esimerkiksi tarkkuutta tai suorituskykyä.

Kohteenilmaisimien analysointiin yleensä käytettävät suureet ovat tarkkuus(engl. average precision, AP), suoritusaika ja suoritusnopeus. Tarkkuuden määritys suoritetaan koh-



Kuva 2.3. *MobileNetin konvoluutiokerros on jaettu kahteen erilliseen osaan: depthwise- ja pointwise-operaatioihin.*

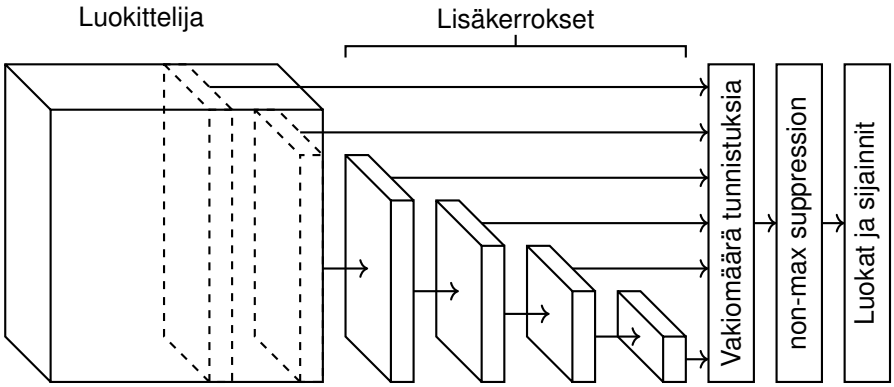
teenilmaisimille eri tavalla kuin luokittelijoille. Kohteenilmaisimien tarkkuus määritetään IoU:n (Intersection over Union) avulla. IoU tässä tapauksessa tarkoittaa ilmaisimelta saadun ja kohteen todellisen sijainnin muodostamien suorakulmioiden pinta-alojen leikkauksen suhdetta niiden unioniin (Kuva 2.4). IoU:lle määritetään kynnysarvo, jonka ylittäneet kohteet luokitellaan oikeiksi eli positiivisiksi tunnistuksiksi. Tarkkuus määritetään sitten näiden positiivisten tunnistusten suhteena kaikkiin tunnistuksiin.



Kuva 2.4. *IoU tarkoittaa neuroverkolta saadun tunnistuksen (A) ja annotaation (B) koordinaattien muodostamien suorakulmioiden leikkauksen pinta-alan suhdetta niiden unionin pinta-alaan.*

2.4.1 Single Shot Detector

Single Shot Detector(SSD) on neuroverkko, joka nimensä mukaisesti yhdellä suorituskerralla määrittää kuvasta luokiteltavat kohteet, luokituksen pistemäärän ja kohteen sijainnin kuvassa. SSD käyttää perusrakenteena luokittelijaa, jonka loppupäähän on lisätty konvoluutiokerroksia määrittämään kohteiden luokat ja sijainnit. Käytettävä luokittelija voidaan valita käyttötarpeen mukaan, esimerkiksi painottaen kohteen tunnistuksen tarkkuutta tai suoritusajan minimointia.



Kuva 2.5. *SSD kohteenilmaisimen perusarkkitehtuuri.*

Kuvassa 2.5 nähdään SSD kohteenilmaisimen perusarkkitehtuuri. SSD:n alkupään muodostaa luokittelijan arkkitehtuuri, josta on poistettu luokituskerrokset. Tähän typistettyyn luokittelijaan on lisätty konvoluutiokerroksia tuottamaan kohteen luokitus ja sijainti. Lisäkerrosten piirrekarttojen koko pienenee loppua kohti edettäessä, mikä mahdollistaa kohteiden tunnistamisen eri mittakaavoissa. Eri mittakaavat mahdollistavat eri kokoisten kohteiden tunnistuksen. Jokainen lisäkerros tuottaa vakiomäärän tunnistuksia. Tarkemmin tämä tarkoittaa, että jokaiselle piirrekerroksen ulostulolle suoritetaan vielä kaksi *pointwise*-operaatiota, joissa toisessa muodostetaan kohteen sijainti ja toisessa luokkien pisteytys eli tunnistuksen varmuus. Jokaiselle vakiotunnistukselle lasketaan varmuus kaikille luokille. Sijaintioperaatioista saadaan 4 kappaletta x- ja y-koordinaatteja kohteen sijainnin määritystä varten. Nämä 4 koordinaattia ovat minimi ja maksimi koordinaatit molemmille sekä x- että y-koordinaateille. Tämän jälkeen koordinaateille lasketaan vielä *offset*, jolla nämä eri mittakaavojen piirrekartoista saadut koordinaatit saadaan vastaamaan verkon sisääntulodimensioita. Eri mittakaavoissa suoritettujen tunnistuksien takia verkko tuottaa useita päällekkäisiä tunnistuksia samasta kohteesta. *Non-maximum Suppression*-operaatio valikoi päällekkäisistä tunnistuksista parhaan. Lopuksi ulostulona saadaan jokaiselle kohteelle luokka, varmuus ja sijainti.

3 KEHITYSTYÖKALUT NEUROVERKOILLE

3.1 Yleiskäyttöiset kehitystyökalut

Yleiskäyttöisillä kehitystyökaluilla tarkoitetaan tässä yhteydessä työkaluja, joita käytetään neuroverkkojen tutkimukseen, kehittämiseen, testaukseen, kouluttamiseen ja suorittamiseen. Tällaisia työkaluja ovat muun muassa TensorFlow [1], Caffe [17], PyTorch [28], Theano [31], MXNet [4] ja CNTK [22]. Näiden lisäksi on olemassa lukuisia erilaisia kehitystyökaluja ja kirjastoja koneoppimiseen ja neuroverkkojen kehitykseen, mutta nämä edellä mainitut ovat eniten käytettyjä työkaluja syväoppivien neuroverkkojen kehitykseen. Työkalujen suosiota on arvioitu esimerkiksi Nguyen et al.[23] vertailussa, jossa suosituimmuutta on analysoitu muun muassa *Github* versionhallinnan erilaisten käyttäjäyhteisön käyttödata-analyysien, kuten *committien*, käyttäjien antamien tähtien ja uusien kehityshaarojen, perusteella. Vertailun mukaan TensorFlow on selvästi suosituin työkalu omassa kategoriassaan.

Edellä mainittujen lisäksi maininnan arvoisia työkaluja ovat *Keras* [6] ja ONNX (engl. Open Neural Network Exchange) [26]. Keras on Python-ohjelmointikielinen korkean tason neuroverkko API, joka toimii TensorFlow:n, CNTK:n tai Theanon päällä. ONNX taas on avoin neuroverkkojen esitysformaatti ja se tukee useimpia neuroverkkojen kehitystyökaluja ja kirjastoja. ONNX:n avulla voi siirtää neuroverkkoja kehitystyökalulta toiselle ja se mahdollistaa esimerkiksi neuroverkon kouluttamisen yhdellä työkalulla ja siirtämisen toiselle verkon suorittamista varten.

Vaikka nämä edellä mainitut työkalut ovat yleiskäyttöisiä ja niillä voidaan viedä neuroverkon tutkimus ja kehitys alusta loppuun valmiiseen sovellukseen asti, ei voida sanoa, että jokin työkalu olisi kaikissa kehityskaaren osavaiheissa paras, vaan työkalut ovat hyviä eri osa-alueilla ja työkalujen kehitys on painottunut hiukan eri asioihin. Esimerkiksi PyTorch on hyvä neuroverkkojen nopeaan tutkimukseen ja kehittämiseen ja testaukseen, mutta siitä puuttuu tuki mobiililaitteille. Caffe taas soveltuu erityisesti konvoluutioneuroverkkojen kehittämiseen. TensorFlow:lla suosituimpana työkaluna on takanaan laaja yhteisö ja sillä on hyvä tuki heterogeeniseen laskentaan ja mobiililaitteille.

Ohjelmointirajapintana näissä työkaluissa yleensä ovat Python ja C/C++ -kielet. Näiden lisäksi jotkin työkalut tukevat muita ohjelmointikieliä kuten Matlab ja Java. Heterogeeniseen laiteympäristöön yleisin rajapinta on CUDA [24], joistakin työkaluista löytyy tuki OpenCL-ohjelmointikehykselle [34]. CUDA:n suosiota selittää cuDNN-neuroverkkokirjasto

[5], joka tarjoaa valmiit komponentit syväoppivien neuroverkkojen toteuttamiseen.

3.1.1 TensorFlow

TensorFlow on Googlen kehittämä avoimen lähdekoodin koneoppimisalusta, joka tarjoaa kehitystyökalut koneoppimiseen tutkimus- ja kehitysvaiheesta aina valmiiseen tuotteeseen asti. Se julkaistiin vuoden 2015 marraskuussa ja se on jatkoa aikaisemmalle DistBelief -projektille. TensorFlow tukee useita ohjelmointikieliä. Python ja C/C++ -kielien lisäksi se tukee esimerkiksi R, Java, JavaScript ja Go-ohjelmointikieliä. TensorFlow tukee myös neuroverkkojen suoritusta heterogeenisessä laiteympäristössä Cuda -ohjelmointirajapinnan kautta.

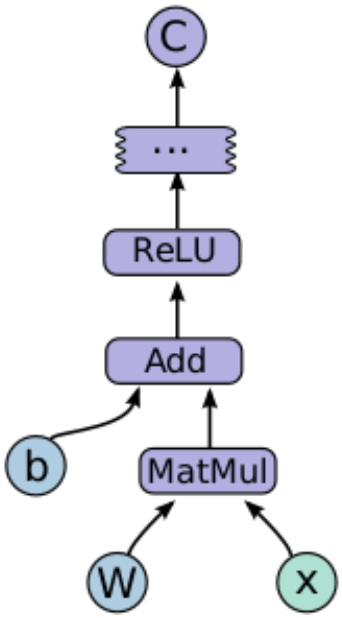
TensorFlow sisältää myös TensorBoard sovelluksen, jonka avulla TensorFlow-graafeja voi tarkastella visuaalisesti. Graafin lisäksi sillä voi tarkastella erilaisia suorituksen ja koulutuksen aikaisia mittareita esimerkiksi kaavioina. Visuaalinen tarkastelu helpottaa graafien optimointia ja debuggausta ja neuroverkon koulutuksen edistymisen seuraamista.

TensorFlow käyttää datavuograafeja algoritmien esittämiseen. TensorFlow-graafi on suunnattu graafi, joka sisältää solmuja. Graafi toteuttaa laskennan datavuokaaviona, jossa solmut kuvaavat erilaisia laskentaoperaatioita, muuttujia ja lisäksi silmukoiden ja haarojen kontrollointiin liittyviä operaatioita (kuva 3.1). Solmulla voi olla nolla tai enemmän sisääntuloja ja ulostuloja. Solmujen välillä data liikkuu kaaria pitkin *tensoreina*. Tensori on moniulotteinen taulukko, joka tukee useita datatyyppejä ja -kokoja. Datan liikkumista ilmaisevien kaarien lisäksi on olemassa kontrollikaaria, joilla ilmaistaan datapolkujen kontrollointiin liittyviä riippuvuuksia.

Operaatiot graafissa ovat yleensä erilaisia laskentaoperaatioita. Operaatiolla voi olla myös attribuutteja. Attribuutteja käytetään graafin ominaisuuksien määrittämiseen graafin luonnin yhteydessä. Yksi operaation toteutustapa on *kernel*, jolla operaation suoritus voidaan ohjata halutulle laskentayksikölle, esimerkiksi GPU:lle. Muuttujat ovat operaatioita, joilla tensoreita voidaan säilyttää graafin yksittäisten suorituskertojen välillä. Muuttujat ovat tarpeellisia, koska tensorit eivät säily suoritusten välillä ja tensorien säilyttäminen voi olla joissakin tapauksissa tarpeen. Muuttujat voivat olla vakioita tai niitä voidaan päivittää erillisillä operaatioilla.

3.1.2 PyTorch

PyTorch on Python-kielinen ohjelmointirajapinta syväoppivien neuroverkkojen kehitykseen. Se pohjaa Torch-kirjastoon [7] ja käyttää samoja optimoituja C-kielisiä kirjastoja kuin Torch. PyTorch on Facebookin tekoälyn tutkimusryhmän (engl. Facebook AI research) projekti ja se on perustettu vuonna 2016. Sen suosio on kasvanut nopeasti ja se on erityisesti koneoppimisen parissa työskentelevien tutkijoiden suosiossa.



Kuva 3.1. *TensorFlow-graafi koostuu solmuista, jotka kuvaavat erilaisia operaatioita ja muuttujia. Kaarilla kuvataan datansiirtoja. [1].*

PyTorch käyttää neuroverkon suorittamiseen dynaamista graafia. Dynaaminen graafi tarkoittaa, että graafin rakenne määritetään verkon suorituksen aikana. Tämä tarkoittaa, että graafin rakenne voi muuttua suorituskertojen välillä. Tämän dynaamisen graafin osalta PyTorch eroaa muista suosituista kehitystyökaluista, jotka käyttävät staattista graafia. Staattisen graafin rakenne määritetään ennen verkon suorittamista ja se pysyy määrityksen jälkeen samanlaisena jokaiselle suorituskerralle. Dynaaminen on staattista joustavampi ja voi olla käytännöllisempi monien ongelmien ratkaisemisessa, kuten esimerkiksi tekstin tunnistuksessa, jossa lauseiden pituudet vaihtelevat.

Selkeä puute PyTorchissa on mobiililaitetuen puuttuminen. PyTorch tukee kuitenkin ONNX-formaattia, joka mahdollistaa PyTorchilla kehitetyn neuroverkon siirtämisen toiselle kehitystyökalulle mobiiliympäristössä suorittamista varten.

3.1.3 Keras

Keras on neuroverkkojen kehitykseen tarkoitettu korkean tason ohjelmointirajapinta. Keras on toteutettu Python -ohjelmointikielellä ja se toimii muiden ohjelmakirjastojen, kuten TensorFlow, CNTK ja Theano, päällä ja abstrahoi näiden *backend*-kirjastojen monimutkaisuutta. Sen kehityksen fokuksena on ollut neuroverkkojen nopea kokeilu ja testaus. Koska Keras toimii *backend*-kirjastojen päällä, mikä mahdollistaa neuroverkkomallin siirtämisen kirjastolta toiselle. Mahdollisuus siirtää malli kirjastolta toiselle laajentaa samalla mallin luomisessa ja koulutuksessa käytettävää ja käyttökohteen laitevalikoimaa.

Keraksen neuroverkkomalli koostuu moduuleista, joita voidaan liittää toisiinsa vähillä ra-

joitteilla. Neuroverkon eri komponentit, kuten neurokerrokset, häviöfunktiot, optimointimetodit, aktivaatiofunktiot ja regularisointimetodit ovat kaikki itsenäisiä moduuleja, joita yhdistelemällä voidaan luoda uusia neuroverkkomalleja. Uusia moduuleja voi myös lisätä jo luotuihin malleihin, mikä tekee verkon laajentamisesta helppoa. Tällainen modulaarisuus tekee erilaisten neuroverkkokokoonpanojen kokeilemisesta ja testauksesta nopeaa ja helppoa.

3.2 Neuroverkkokirjastot Androidille

Useimmilla yleiskäyttöisillä neuroverkkojen kehitystyökaluilla on myös omat ohjelmakirjastot neuroverkkojen suorittamiseen mobiililaitteissa. Nämä kirjastot sisältävät usein geneerisen toteutuksen neuroverkko-operaatioista, eivätkä siten osaa hyödyntää mobiililaitteiden erilaisia prosessoriarkkitehtuureja optimaalisesti.

Edellä mainitusta syystä useimmilla prosessorivalmistajilla onkin nykyään tarjota työkaluja ja kirjastoja neuroverkkojen optimointiin ja suorittamiseen niiden omilla prosessoriarkkitehtuureilla. Tällaisia ovat esimerkiksi Qualcommin Snapdragon Neural Processing Engine [33] , Nvidian TensorRT [25] ja Arm NN [3] .

Jos tarkoitus on maksimoida suorituskyky neuroverkon laskennassa, on usein tarpeen käyttää prosessorivalmistajan tarjoamia ohjelmakirjastoja verkkojen optimointiin ja suoritukseen. Näiden kirjastojen toteutukset on optimoitu hyödyntämään kutakin arkkitehtuuria optimaalisesti. Prosessoriarkkitehtuurit sisältävät nykyään usein myös joko neuroverkkojen suorittamiseen tarkoitettuja yksiköitä, kuten Arm Machine Learning Processor [2] , tai yksiköitä, kuten Hexagon Vector Extensions, jonka suunnittelussa on otettu huomioon myös neuroverkkojen laskennan tarpeet. Näiden yksiköiden hyödyntämiseen neuroverkkojen laskennassa on usein käytettävä valmistajan omaa kirjastoa.

3.2.1 TensorFlow

TensorFlow sisältää kaksi Android mobiililaitteille tarkoitettua kirjastoa, TensorFlow Mobilen ja TensorFlow Liten. TensorFlow Mobile-kirjasto on ollut osa TensorFlow:ta sen alusta asti. Lite-versio taas on TensorFlow:n uudempi projekti, jonka tarkoituksena tehdä TensorFlow:n neuroverkkomalleista suorituskykyisempiä ja kevyempiä mobiililaitteissa.

TensorFlow Mobile tarjoaa rajapinnan TensorFlow:n neuroverkkomallien käyttämiseen mobiilisovelluksissa. Kirjasto on toteutettu C++-kielellä, mutta kirjaston käyttöön on olemassa myös Java-kielinen rajapinta, joka käyttää samaa C++ -kielistä toteutusta.

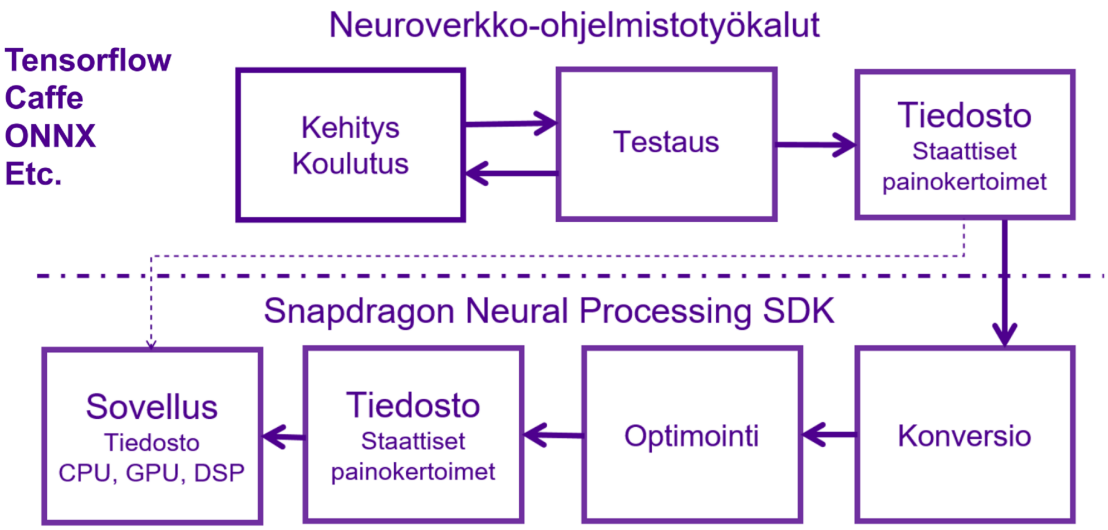
TensorFlow Lite vaatii TensorFlow:n neuroverkkomallin muuntamista Lite yhteensopivaan muotoon. Muuntaminen täytyy suorittaa ennen mobiilisovelluskäyttöä ja se tehdään TensorFlow:n omalla konversiosovelluksella. Konversio pienentää mallin tiedostokokoa ja mahdollistaa samalla myös erilaisten optimointimenetelmien käytön vaikuttamatta kuiten-

kaan neuroverkon tarkkuuteen. Konversion jälkeen malli on myös mahdollista kvantisoida 8-bittiseen kiintolukumuotoon. Kvantisointi voi pienentää entisestään mallin tiedostokokoa ja lyhentää suoritusaikaa, mutta se myös mahdollisesti huonontaa verkon tarkkuutta. TensorFlow Lite on kuitenkin vielä kehitysvaiheessa, mikä tarkoittaa, ettei se vielä tue kaikkia TensorFlow:n neuroverkko-operaatioita. Tämä rajoittaa joidenkin neuroverkkojen käyttöä TensorFlow Lite -kirjastolla, mutta tulevaisuudessa Lite tulee todennäköisesti korvaamaan TensorFlow Mobilen mobiilisovelluskäytössä.

3.2.2 Snapdragon Neural Processing Engine

Snapdragon Neural Processing Engine (SNPE) on Qualcommin neuroverkkokirjasto mobiililaitteille, jotka on varustettu Snapdragon-piirillä [33]. SNPE sisältää työkalut neuroverkkojen konversioon, analysointiin, suorituskyvyn testaukseen ja Java ja C++ APIt verkkojen suorittamiseen sovelluksissa. SNPE:ssä ei siis ole omia työkaluja neuroverkkojen kehitykseen tai koulutukseen, vaan nämä vaiheet täytyy ensin suorittaa jollakin toisella kehitystyökalulla kuten Caffe tai TensorFlow.

Kuva 3.2 esittää konvoluutioneuroverkkosovelluksen perustyövuon SNPE:n ohjelmakirjastolle. Neuroverkon kehitys, koulutus ja testaus toteutetaan jollakin yleiskäyttöisellä ohjelmistotyökalulla, kuten TensorFlow tai Caffe. Valmis malli muunnetaan tämän jälkeen SNPE:n konversiosovelluksella SNPE:n tukemaan tiedostomuotoon, jota voidaan käyttää mobiililaitesovelluksessa.



Kuva 3.2. *SNPE:n työvuo.*

Konversiotyökaluilla neuroverkkomallin voi muuntaa Caffe, Caffe 2, TensorFlow tai ONNX tiedostomuodoista SNPE:n omaan *deep learning container (dlc)* tiedostomuotoon. SNPE tukee vain konvoluutioneuroverkkoja, joka rajoittaa myös tuettujen neuroverkko-operaatioiden määrää. Lisäksi jotkin operaatiot voidaan suorittaa vain CPU:lla. Esimerkiksi SSD

neuroverkon *non-max suppression* -operaation suorittamiseksi operaation laskenta siirretään muilta laskentayksiköiltä CPU:lle. Tämä laskennan palautus CPU:lle voi – ylimääräisistä datansiirtojen takia – kasvattaa verkon suoritusaikaa [33].

Neuroverkkojen analysointi- ja diagnostiikkatyökalut tarjoavat erilaisia tapoja tarkastella verkon rakennetta. Nämä työkalut mahdollistavat esimerkiksi verkon rakenteen visuaalisen tarkastelun tai tekstipohjaista informaatiota verkon rakenteesta ja eri kerrosten ja operaatioiden parametreista. Benchmarking-työkalu kerää suorituskykyyn liittyvää mittausdataa neuroverkon suorittamisen ajalta. Mittausdataa kerätään verkon ja sen eri kerrosten suoritusajoista ja muistinkäytöstä.

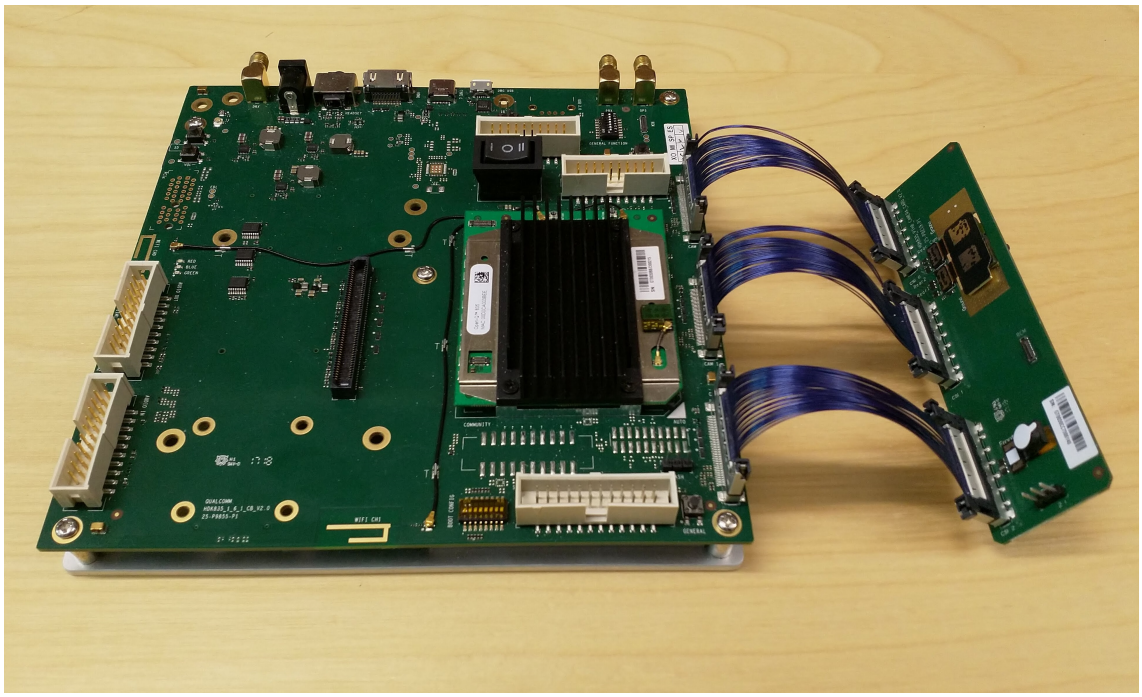
Java ja C++ APIt tarjoavat ohjelmointirajapinnan, jonka kautta neuroverkkoa voidaan ajaa kohdelaitteessa Android sovelluksessa. Rajapinnat käyttävät *dlc* tiedostoa verkon alustamiseen. Alustuksen yhteydessä voidaan antaa erilaisia alustusparametrejä. Parametreillä määritetään muun muassa käytettävä laskentaympäristö ja suorituskykyprofiili. Käytettävissä olevia laskentaympäristöprofiileja on useita, mutta niiden käyttöä rajoittaa käytettävä kohdelaite. Myös suorituskykyprofiileja on useita ja sellaisia ovat esimerkiksi *default*, *power saver* ja *high performance* -profiilit. *Power saver*-profiili pyrkii minimoimaan verkon suorituksen aikaisen tehonkulutuksen ja *high performance* -profiili taas maksimoimaan suorituskyvyn, joka tässä tarkoittaa suoritusajan minimointia.

4 MITTAUKSET

4.1 Käytetty laitealusta

Tässä luvussa esitellään laiteympäristö, jossa tähän työhön liittyvät mittaukset on tehty. Esittelyssä ei käydä läpi kaikkia laiteympäristöön liittyviä komponentteja, vaan ainoastaan työn kannalta oleelliset komponentit.

Kaikki mittaukset tehtiin Open-Q 835 HDK Development Kitillä [27], joka on Intrinsycin [16] valmistama kehitysalusta. Intrinsyc on keskittynyt tuottamaan erilaisia sulautettuja- ja Internet of Things -tuotteita ja -palveluja.



Kuva 4.1. *Open-Q 835 HDK laitealusta. Alustaan kytketty kameramoduuli oikealla.*

Open-Q 835 HDK -kehitysalusta on muun muassa suunnattu Android/Linux sovelluskehittäjille, laitevalmistajille, videovalvontaan, robotiikkaan, Qualcommin Snapdragon 835 -sarjan teknologiaa hyödyntävien sovellusten arviointiin, optimointiin, testaukseen ja kehittämiseen. Kehitysalusta pitää sisällään seuraavat komponentit:

- Open-Q 835 -piirikortti,
- komponenttialusta I/O-liitimille ja ulkoisten oheislaitteiden kytkemiseen,

- GNNS (Global Navigation Satelite System)-piirikortti,
- näyttösovitinkortti LCD-kosketusnäytölle,
- 4.35 V 3000 mAh akku ja
- 12 V AC adapteri.

Käyttöjännitteenä alusta käyttää 12 V tasavirtaa, joka saadaan AC adapterilta. Tämä jännitelähde toimii koko alustan ja siihen liitettyjen oheiskorttien jännitelähteenä. Ennen käyttöjännitteen syöttämistä oheiskorteille jännite muunnetaan oheiskortille sopivaksi. Lisäksi jännitelähteenä voidaan käyttää USB-C-porttia ja akkua, mutta näistä jännitelähteistä saadaan käyttöjännite vain Open-Q 835 -piirikortille, jota voidaan siis käyttää itsenäisesti ilman muita oheiskortteja.

Komponenttialustasta löytyvät seuraavat I/O-liitynnät: USB3.1, USB to UART-silta ja HDMI-portti ulkoiselle näytölle. Lisäksi alustalta löytyvät JTAG-portti testaukseen ja debuggaukseen, liittimet kameralle sekä erinäisiä sensori-, GPS- ja audio-liityntöjä.

Open-Q 835 -piirikortti

Open-Q 835 -piiri on kehitysalustan varsinainen laskenta-alusta. Piiri pitää sisällään muun muassa seuraavat komponentit:

- Snapdragon 835 -pääprosessointiyksikön,
- LPDDR4 1866 MHz 4 GB RAM muistin,
- Wi-Fi + BT + FM-yhdistelmäpiirin ja
- 128 GB UFS 2.1 -keskusmuistia.

Snapdragon 835 -piiri pitää sisällään 3 prosessoriyksikköä, joita voidaan käyttää neuroverkkojen laskentaan, keskusprosessorin (CPU), grafiikkaprosessorin (GPU) ja digitaalisen signaaliprosessorin (DSP). Piiri perustuu 10 nm FINFET piiriteknologiaan.

Keskusprosessorina toimii Qualcommin Kryo 280 -piiri, joka perustuu ARM:n Cortex A73-teknologiaan. Kryo 280-prosessorissa on 8 ydintä ja se noudattaa *big.LITTLE* -arkkitehtuuria. Tässä tapauksessa *big.LITTLE*-arkkitehtuuri tarkoittaa, että ytimet on jaettu kahteen ryhmään, joista toinen on optimoitu maksimoimaan suorituskykyä ja toinen tehonkulutuksen minimointiin. Molemmissa ryhmissä on 4 ydintä. Korkean suorituskyvyn *big*-ryhmän ytimien kellotaajuus on 2,45 GHz ja alhaisen tehonkulutuksen *LITTLE*-ryhmän ytimien 1,9 GHz. Tarkoituksena on tarjota samassa ytimessä sekä korkeaa suorituskykyä että alhaista tehonkulutusta, joita voidaan käyttää eri tarkoituksiin tarpeen mukaan. Tämä voi olla hyödyllistä esimerkiksi mobiililaitteissa, joissa alhainen tehonkulutus on tärkeää.

Snapdragon 835 -piirissä on Qualcommin Adreno-tuotesarjaan kuuluva Adreno 540 -grafiikkaprosessori. Adreno 540 -prosessorin kellotaajuus on 710 MHz, prosessori sisältää 256 ALUa ja sisäistä muistia siinä on 1024 kB. Prosessorilla on täysi OpenCL 2.0 tuki. Lisäksi piiristä löytyy tuki Vulkan 1.0, OpenGL ES 3.2 ja Direct 3D 12- ohjelmistorajapinnoille.

Piiristä löytyvä digitaalinen signaaliprosessori on Qualcommin Hexagon 682 -prosessori. DSP sisältää kontrolli- ja yleisrekisterit, paikallista muistia, 4 suoritusyksikköä ja vektorilaskentayksikön (engl. Hexagon Vector eXtension lyh. HVX). Hexagon DSP on VLIW-suoritinarkkitehtuuri, joka mahdollistaa jopa neljän käskyn rinnakkaisen suorittamisen yhden kellojakson aikana. HVX-laajennus on suunniteltu tukemaan korkeaa suorituskykyä vaativia kuvantamis- ja konenäkösovelluksia. HVX-laajennuksen tuki konenäkösovelluksiin tekee DSP:stä kiinnostavan tähän työhön liittyvien neuroverkkojen laskennan kannalta. Laajennus koostuu neljästä 512-bitin kiintolukuvektorista, jotka voidaan yhdistää kahdeksi 1024-bitin vektoriksi. Vektori voi suorittaa yhden operaation useille data-alkiolle rinnakkain. Esimerkiksi 8-bitin kiintolukuja mahtuu 512-bitin vektoriin 64 kappaletta. Rinnakkaisia operaatioita 512-bitin vektoreilla voidaan suorittaa 4 ja 1024-bitin vektoreilla 2 kappaletta.

4.2 Käytetyt ohjelmistot ja algoritmit

Mittauksissa käytetyt ohjelmistot on esitetty taulukossa 4.1. Ohjelmistojen tarkemmat kuvaukset löytyvät seuraavista aliluvuista.

4.2.1 Snapdragon Profiler

Snapdragon Profiler on Snapdragon-prosessoreiden profilointiin tarkoitettu sovellus. Sovellusta voidaan käyttää Windows, Mac ja Linux ympäristöissä ja sitä käytetään Android mobiililaitteiden profilointiin USB-väylän kautta. Sovelluksella on mahdollista analysoida CPU, GPU, DSP-yksikköjä, laitteen muistin ja energian käyttöä, sekä lämpötilaa ja verkon käyttöä.

Snapdragon Profilerissa on neljä erilaista käyttömoodia: *realtime*, *trace capture*-, *snapshot capture*- ja *sampling capture*-moodit [29]. *Realtime*-moodissa käyttäjä voi sovelluksen käydessä reaaliajassa tarkastella erilaisia suorituskykyyn liittyviä mittareita. Profilerilla voidaan tarkastella jopa 150 eri laitteiston suorituskykymittaria 22 kategoriassa. Mittareita voi tarkastella sovelluksen käyttöliittymästä reaaliaikaisina viiva- tai pylväsgraafeina. Reaaliaikaisen mittausdatan voi myös tallentaa tiedostoon myöhempää tarkastelua ja analysointia varten.

Trace capture kaappaa ajon aikaisia järjestelmän ja sovellusten toimintaan liittyvien tapahtumien ja laitteiston suorituskykymittareiden dataa. Kaappauksen maksimi kesto on 10 sekuntia. *Trace capture* mahdollistaa esimerkiksi säikeiden aikataulutuksen tarkastelun CPU-ytimillä tai OpenCL-kernelien suorituksen yksityiskohtaisen tarkastelun.

Snapshot capture mahdollistaa yksittäisen kehyksen kaappaamisen grafiikkasovelluksesta. Kaappaus näyttää yksityiskohtia kehyksen renderöinnistä. *Sampling capture* taas näyttää funktio-kutsugraafin suoritettavasta sovelluksesta.

Taulukko 4.1. *Mittauksiin käytetyt ohjelmistot.*

Mittaus	Ohjelmistot	Kohteenilmaisin	Toteutus
Tarkkuus	COCO API	MobileNet SSD 90 luokkaa	TensorFlow OpenCL SNPE CPU SNPE GPU SNPE DSP
Laskentanopeus	Java nanoTime()-metodi	MobileNet SSD 90 luokkaa	TensorFlow OpenCL SNPE CPU SNPE GPU SNPE DSP
Luokkien määrän vaikutus laskentanopeuteen	Java nanoTime()-metodi	MobileNet SSD 90 luokkaa ja MobileNet SSD 1 luokka	TensorFlow SNPE GPU
Tunnistuksien määrän vaikutus laskentanopeuteen	Java nanoTime()-metodi	MobileNet SSD 90 luokkaa	TensorFlow SNPE GPU
OpenCL-kernelien laskentanopeudet	Snapdragon Profiler, trace capture	MobileNet SSD 90 luokkaa	OpenCL SNPE GPU
Tehonkulutus	Trepn Power Profiler	MobileNet SSD 90 luokkaa	TensorFlow OpenCL SNPE GPU SNPE DSP
Lämpötila	Snapdragon Profiler, realtime	MobileNet SSD 90 luokkaa	TensorFlow

4.2.2 Trepn Power Profiler

Trepn Power Profiler on Android-mobiililaitteessa suoritettava sovellus, jolla voidaan profiloida Android-sovelluksen suorituskykyä ja tehonkulutusta. Trepn on Qualcommin kehittämä sovellus ja on tarkoitettu erityisesti Snapdragon-piirillä varustettujen Android laitteiden profilointiin. Trepn Profiler mahdollistaa reaaliaikaisen suorituskykymittareiden tarkastelun sovelluksen ajon aikana. Reaaliaikainen data näkyy sekä graafina että numeroarvona laitteen näytöllä. Lisäksi mittausdata voidaan tallentaa tiedostoon myöhempää analysointia varten.

Tehonkulutuksen mittaukset Trepn Profilerilla suoritetaan akkukäytöllä, jolloin laite ei saa olla kytkettynä laturiin tai USB-väylän kautta toisiin laitteisiin. Tarkkoja mittaustuloksia tehonkulutuksista saadaan mitattaessa koko järjestelmän tehonkulutusta. Joillakin laitteilla on myös mahdollista saada tarkkoja mittaustuloksia eri laitekomponenttien tehonkulutuksista, kuten esimerkiksi CPU, GPU, WiFi-komponenteista. Sovelluskohtaiset mittaukset ovat arvioituja tehonkulutuksia, missä arvot annetaan suhteessa *Delta*-arvoon.

Delta-arvo on vertailuarvo, jonka sovellus määrittää ennen varsinaista mittausta. Hoque et al.[13] tekemässä mobiililaitteiden energiankulutuksen profilointityökalujen vertailussa Trepn Profiler todettiin Nokia Energy Profilerin ohella tarkimmiksi omassa kategoriassaan. Molemmille profilointisovellukselle vertailussa saatiin 99 % tarkkuus tehonkulutuksen mittauksissa.

Trepn Profilerilla on myös mahdollista kaapata Android *Intent*tejä muilta sovelluksilta. Android Intent on viestien välitykseen tarkoitettu objekti, jolla voi muun muassa lähettää pyyntöjä käyttöjärjestelmälle tai toisille sovelluksille. Näillä *Intent*-pyynnöillä voi sovelluksesta käsin hallinnoida Trepn Profileria. Tällä tavalla voidaan tarkasti rajata profilointi halutun sovellustapahtuman tai toiminnallisuuden tarkasteluun.

4.2.3 COCO - Common Objects in Context

Microsoft Common Objects in COntext (COCO) on datasetti, joka sisältää kuvasarjat neuroverkkojen kouluttamiseen, kuvasarjoihin liittyvät annotaatiot ja COCO API ohjelmointirajapinnan [19]. MS COCO-datasetissä on 3 kuvasarjaa, jotka on julkaistu vuosina 2014, 2015, ja 2017. Kuvasarjoissa on yhteensä noin 330 tuhatta kuvaa ja 1,5 miljoonaa kohdetta. Luokkia kuvasarjoissa on 91 kappaletta. Kukin kuvasarja on jaettu kolmeen osaan: koulutus-, validointi- ja testikuvasarjoihin. Ideana MS COCO:ssa on ollut luoda kuvasarja, jossa kohteet ovat niiden luonnollisessa ympäristössä. Tällä tavoin se poikkeaa aikaisemmista kuvasarjoista, kuten Pascal VOC[9] ja ImageNet, joissa kuvat usein esittivät itse kohdetta. Lin et al. osoittivatkin, että MS COCO kuvasarjoilla koulutetut verkot suoriutuivat paremmin kuin aikaisemmilla kuvasarjoilla koulutetut tunnistustehtävistä, joissa kohteet ovat osa jokapäiväisiä tilanteita, taustalla tai osittain näkyvissä.

Kuvasarjojen annotaatiot sisältävät tietoa kuvissa olevista kohteista. Annotaatioihin on määritelty jokaisen kohteen luokka ja sijainti. Sijainti on määritelty kolmella eri tavalla: suorakulmiona (*bounding box*), kiintopisteinä (*keypoints*) ja segmenttinä.

COCO API sisältää ohjelmointirajapinnan Python, Matlab ja Lua ohjelmointikielille. Se sisältää työkalut muun muassa annotaatioiden jäsentämiseen ja neuroverkon tarkkuuden arvioimiseen.

4.2.4 MobileNet SSD

Kohteenilmaisijana tässä työssä käytettiin MobileNet SSD-neuroverkkoa, koska se on suunniteltu erityisesti käytettäväksi sulautetuissa- ja mobiilisovelluksissa ja soveltuu siten hyvin tässä työssä käytettävälle laitteelle. Eli ilmaisimena käytetään SSD-arkkitehtuuria (Luku 2.4.1), jonka selkärankana toimii MobileNet v1-luokittelija (Luku 2.3.1). MobileNet-luokittelijan konvoluutiokerrokset on jaettu *depthwise*- ja *pointwise*-operaatioihin. Tämä jako vähentää parametrien määrää merkittävästi ja samalla vähentää laskentakuormaa,

minkä takia se soveltuu hyvin sulautettuihin- ja mobiililaitteisiin.

Huang et al.[15] vertailivat tutkimuksessaan joidenkin tunnetuimpien kohteenilmaisimien suoritusnopeuksia ja tarkkuuksia. Tutkimuksessa vertailtiin ilmaisimia käyttäen useita eri luokittelijoita. Vertailussa SSD-ilmaisin oli nopein kaikilla luokittelijoilla ja MobileNet- luokittelijalla koko vertailun nopein. Kokonaismuistinkäytössä MobileNet SSD oli myös vertailun tehokkain käyttäen vähiten muistia. Tarkkuuden osalta MobileNet SSD pärjäsi vertailussa myös hyvin. Ainoastaan pienten kohteiden osalta MobileNet SSD:n tarkkuus oli muita ilmaisimia huonompi. Yleisesti ottaen voidaan sanoa, että MobileNet SSD -ilmaisimella saadaan hyvä kompromissi nopeuden ja tarkkuuden suhteen. Edellä mainittujen seikkojen perusteella MobileNet SSD -kohteenilmaisin soveltuu hyvin käytettäväksi ajoneuvoluokittelijana mobiililaitteessa.

4.3 Mittausjärjestelyt

Kaikki mittaukset suoritettiin luvussa 4.1 esitellyllä Open-Q 835 HDK -kehitysalustalla. Kohteenilmaisimena mittauksissa käytettiin MobileNet SSD -neuroverkkoa. Pääosa mittauksista suoritettiin käyttäen COCO-datasetillä koulutettua verkkoa, joka tunnistaa 90 luokkaa, mukaan lukien useita eri ajoneuvoluokkia. Lisäksi joissakin mittauksissa käytettiin kasvontunnistukseen koulutettua verkkoa, joka tunnistaa vain yhden luokan. Molemmat verkot oli toteutettu ja koulutettu TensorFlow:n työkaluilla. TensorFlow:n toteutuksien lisäksi verkoista tehtiin myös SNPE:n ja OpenCL toteutukset. SNPE:n toteutukseen käytettiin SNPE:n konversiotyökalua, joka muuntaa TensorFlow:n tiedoston SNPE:n omaan tiedostomuotoon. OpenCL-toteutus taas on Tampereen yliopistolla itse tehty C-kielinen toteutus, joka käyttää itse toteutettuja OpenCL-kerneleitä konvoluutiokerrosten suorittamiseen.

Mittauksissa mitattiin pelkästään neuroverkon suorituskykyä laitealustalla ja sen eri kiihdytinyksiköillä. Mittauksiin käytetyn sovelluksen suorittamia muita toimintoja, kuten kuvien esikäsittely, ei mitattu eikä analysoitu. Kohteenilmaisimina käytetyt neuroverkot valikoitiin siten, että niillä saatiin mahdollisimman kattavasti suoritettua mittauksia eri ajoympäristöissä. Verkkojen valikointia rajoitti mittausajankohtana lähinnä SNPE:n konversiotyökalujen rajoitteet. Myös kohteenilmaisimen koulutus rajattiin tämän työn ulkopuolelle, minkä takia ajoneuvoluokkien tunnistukseen käytetty verkko ei ollut varsinaisesti koulutettu siihen tehtävään. Koulutus tiettyyn tehtävään ei kuitenkaan vaikuta verkon suorituskykyyn tai tehonkulutukseen, joten se ei ole olennainen seikka tässä työssä, eikä vaikuta mittausten lopputulokseen.

Mittaukset suoritettiin Java-ohjelmointikielellä tehdyllä sovelluksella. Mittaussovelluksen ohjelmointiin käytettiin Android Studiota, joka on Android-käyttöjärjestelmän virallinen ohjelmointiympäristö. Käytetty Android SDK versio oli 26.1.1 ja Android NDK versio 18.1. Mittauksissa käytetyssä sovelluksessa käytettiin TensorFlow:n ohjelmakirjaston versiota 1.13 ja SNPE:n versiota 1.22. Kehitysalustan käyttöjärjestelmä oli Android 7.1.2.

TensorFlow:n verkko käyttää mittauksissa ajoympäristönä CPU:ta ja OpenCL -verkko GPU:ta. SNPE:llä suoritetuissa mittauksissa käytetyt ajoympäristöt ja profiilit on esitetty jokaisen mittauksen kuvauksen yhteydessä seuraavissa aliluvuissa. DSP:llä suoritettavia mittauksia varten jouduttiin käyttämään omaa *non-max suppression* -operaation toteutusta, koska kyseinen operaatio ei toiminut oikein SNPE:n toteutuksessa. Mittauksissa, joissa käytettiin itse toteutettua *non-max suppression*-operaatiota, verkon ulostuloina käytettiin *non-max suppression* -operaation sisääntulodataa ja kokonaislaskentanopeuksiin mitattiin myös itse toteutetun operaation suorittamiseen kuluva aika.

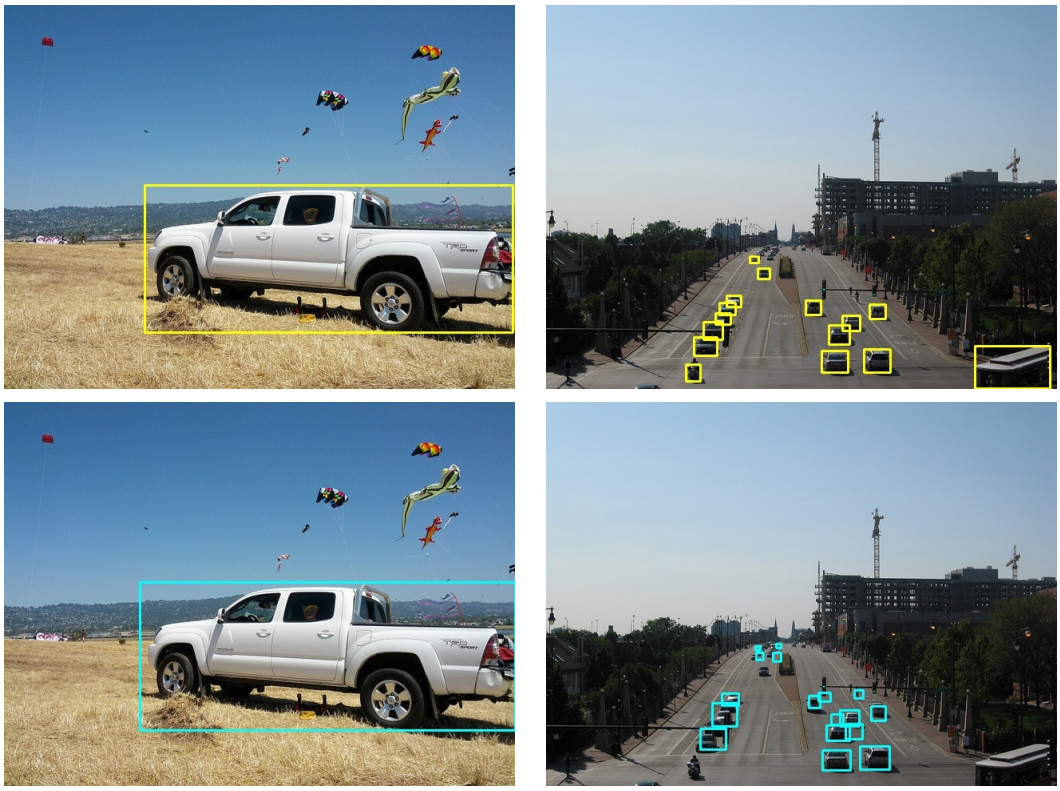
4.3.1 Tarkkuuden mittaukset

Tarkkuuden mittauksilla tässä työssä on tarkoitus osoittaa, että konversiot eri sovelluskehysten välillä toimivat halutulla tavalla. Lisäksi mitattiin tarkkuuksia SNPE:n eri laskentayksiköille, jotta voidaan vertailla tarkkuuksia myös niiden välillä, ja itse toteutetun *non-max suppression*-operaation oikean toiminnallisuuden varmistamiseksi.

Mittauksiin käytettiin kahta kuvasarjaa. Ensimmäiseen kuvasarjaan valittiin helppoja kohteita tunnistettavaksi ja tunnistettavien kohteiden määrä kuvissa oli pieni (1-3 kohdetta). Tähän ensimmäiseen kuvasarjaan viitattaessa käytetään jatkossa termiä *helppo-kuvasarja*. Toiseen kuvasarjaan valittiin kuvia, joissa oli vaikeampia tunnistuksia ja tunnistuksia oli määrällisesti paljon (2 - 15 kohdetta). Tähän jälkimmäiseen kuvasarjaan viitataan jatkossa termillä *vaikea-kuvasarja*. Molemmissa kuvasarjoissa oli 10 kuvaa. Kuvat valikoitiin COCO-datasetin vuoden 2017 validointikuvasarjasta ja tunnistettavat kohteet olivat ajoneuvoja. Kuvista saatujen tulosten analysointiin käytettiin COCO API:n Matlab-työkalua. Kuvassa 4.2 on molemmista kuvasarjoista esimerkkikuvat, jotka havainnollistavat kuvasarjojen eroja. Ylärivin kuviin on piirretty kohteet yksilöivät suorakulmiot koulutusdatasetin määritelmän mukaan ja alariviin TensorFlow:n toteutukselta saatujen tunnistuksien mukaan.

Mittaukset suoritettiin TensorFlow:n, SNPE:n ja geneerisen OpenCL:llä toteutetulla MobileNet SSD -verkolla. SNPE:n osalta mittauksia suoritettiin CPU-, GPU- ja DSP -yksiköillä ja *default*-, *high performance*- ja *power saver* -suorituskykyprofiileilla. Mitattiin myös verkon tarkkuus käytettäessä itsetehtyä *non-max suppression* -operaatiota, jotta voitiin osoittaa sen toimivan halutulla tavalla. *Non-max suppression* -operaation toteutettiin Java-ohjelmointikielellä ja sen laskenta tapahtuu CPU:lla.

COCO API -työkalu vertaa neuroverkosta saatuja tunnistuksia datasetin annotaatioissa määritettyihin tunnistettaviin kohteisiin. Vertailu suoritetaan laskemalla ensin tunnistuksen ja annotaation koordinaateista IoU (Intersection over Union). IoU tarkoittaa leikkauksen pinta-alan suhdetta unionin eli yhdisteen pinta-alaan (kuva 2.4). COCO-datasetin metriikassa tunnistukseksi lasketaan 0,5 ja sitä suuremmat IoU:n arvot. Lisäksi 0,5 ylittävä IoU-arvoalue on jaettu 10 kynnysarvoon eli kynnysarvojen väli on 0,05 pistettä. Jokainen tunnistus pisteytetään sen ylittämien kynnysarvojen mukaan. Tämä tarkoittaa, että



Kuva 4.2. *Kuvaesimerkit helposta (vasen) ja vaikeasta (oikea) kuvasarjoista. Ylempiin kuviin piirretty suorakulmiot annotaatioissa määritellyille kohteille ja alapuolisiin TensorFlow:n verkosta saaduille tunnistuksille.*

esimerkiksi IoU-arvo 0,71 antaa pisteen jokaiselle kynnysarvolle välillä 0,5-0,7. Tunnistuksen luokalle saatua pisteytystä ei huomioida tässä IoU:hun perustuvassa tarkkuuden analyysissä, vaan jokainen tunnistuksen pistemäärän kynnysarvon ylittänyt tunnistus on samanarvoinen. Näin pisteytetään koko testikuvasarja. Jokaiselle kynnysarvolle lasketaan tarkkuus vertaamalla todelliseen tunnistettavien kohteiden määrään, ja lopuksi lasketaan kynnysarvojen tarkkuuksien keskiarvo (engl. Average Precision lyh. AP), jota tässä työssä käytetään ilmaisemaan neuroverkon tarkkuutta. Tämä COCO-datasetin metriikan käyttämä menetelmä korostaa tunnistuksen lokalisoinnin tarkkuutta ja eroaa tällä tavoin perinteisestä menetelmästä, jossa käytetään vain yhtä kynnysarvoa (0,5). Yhtä kynnysarvoa käytetään esimerkiksi VOC-datasetin analysointimetriikassa.

4.3.2 Laskentanopeuden mittaukset

Laskentanopeuden määrittämiseksi mitattiin inferenssiin kuluvaa aikaa, eli mitattiin kuinka kauan kestää yhden kuvan ajo verkon läpi. Mittauksiin käytettiin MobileNet SSD -kohteenilmaisinta. Mittaukset suoritettiin TensorFlow:n, SNPE:n ja OpenCL toteutuksilla. SNPE:llä mittaukset tehtiin kaikille käytettävissä oleville ajoympäristöille ja *default*, *high performance* ja *power saver* -suorituskykyprofiileille.

Kuten luvussa 4.3 mainittiin SNPE DSP:llä tehdyissä mittauksissa käytettiin itsetoteu-

tettua *non-max suppression* -operaatiota. Jotta voitaisiin tutkia itsetoteutetun *non-max suppression*-operaation vaikutusta kohteenilmaisimen suoritusnopeuteen, suoritettiin vertaileva mittaus SNPE GPU:lla, jolla saatiin mitattua molempien sekä SNPE:n että itsetehdyn toteutuksen suoritusnopeudet.

Mittauksiin käytetyn kuvasarjan koko oli 437 kuvaa. Kuvat valittiin sattumanvaraisesti COCO-datasetin vuoden 2017 validointikuvasarjasta, koska käytetty kohteenilmaisin on koulutettu kyseisellä datasetillä. Tällä tavalla valikoidulla kuvasarjalla varmistetaan, että verkon suorittamisen aikana tapahtuu kohteiden tunnistuksia ja niihin liittyviä laskusuorituksia.

Jokaisessa mittauksessa ajettiin koko käytetty kuvasarja verkon läpi. Jokaiselle kuvalle mitataan verkon suorittamaan inferenssiin käytetty laskenta-aika. Yksittäisistä mitatuista laskenta-ajoista lasketaan lopuksi keskiarvo laskenta-ajaksi. Laskenta-ajoista laskettiin myös verkon laskentanopeus, joka tässä tapauksessa tarkoittaa maksimimäärää inferenssejä sekunnissa. Laskentanopeus saadaan laskenta-ajan käänteislukuna.

Inferenssin laskenta-aikojen mittauksiin käytettiin Java-ohjelmointikielen *System*-luokan *nanoTime()* -metodia[35]. Tämän metodin tarkkuuden eli resoluution määrittämiseen käytetty menetelmä on kuvattu Ohjelma 4.1. Menetelmässä haetaan käytetyn järjestelmän metodille tarjoama ajallisesti pienin askel. Askel haetaan 100 kertaa ja lopuksi lasketaan askelten keskiarvo, joka kertoo käytetyn metodin keskimääräisen resoluution. Tätä *nanoTime()* -metodia käytettiin kaikissa laskentanopeuksien mittauksissa.

Haluttiin myös tutkia kohteenilmaisimen tunnistamien luokkien ja tunnistettujen kohteiden määrän vaikutusta laskentanopeuteen. Molempien mittaukset tehtiin TensorFlow:n ja SNPE:n toteutuksille. Luokkien määrän vaikutuksen mittauksiin käytettiin kahta eri tarkoitukseen koulutettua MobileNet SSD -verkkoa: COCO-datasetillä koulutettu verkko, joka tunnistaa 90 eri luokkaa, ja Face Detection Data set and Benchmark (FDDB) -datasetillä[10] kasvojen tunnistukseen koulutettu verkko, joka tunnistaa 1 luokan. Molempien verkkojen laskentanopeuksien mittauksiin valittiin kuva, josta verkko tunnisti yhden kohteen. Kuvat valittiin COCO-datasetin vuoden 2017 validointikuvasarjasta. Laskentanopeuden mittaus samalle kuvalle toistettiin 100 kertaa, joista laskettiin keskiarvo. Tunnistuksien määrän vaikutus laskentanopeuteen toteutettiin kuten luokkien määrän vaikutus laskentanopeuteen. Erona oli, että käytettiin ainoastaan yhtä verkkoa ja käytettyjä kuvia oli kaksi. Käytetty verkko oli COCO-datasetillä koulutettu MobileNet SSD. Kuvat valittiin tässäkin tapauksessa COCO-datasetin vuoden 2017 validointikuvasarjasta siten, että toisessa kuvassa tunnistettavia kohteita oli 1 kappale ja toisessa 27 kappaletta.

4.3.3 Tehonkulutuksen mittaukset

Tehonkulutuksen mittaukset suoritettiin Trepn Power Profiler -sovelluksella. Mittauksiin käytettiin MobileNet SSD v1 -neuroverkkoa. Verkkoa ajettiin TensorFlow:n, SNPE:n ja OpenCL toteutuksilla. SNPE:llä mittauksia tehtiin GPU ja DSP ajoympäristöissä ja *de-*

```
public class timerPrecision {


    private static long total = 0;
    private static int nRepeat = 100;

    public static void getTimerPrecision() {
        long start;
        long stop;

        for (int i = 0; i < nRepeat; i++) {

            start = System.nanoTime();
            stop = System.nanoTime();

            while (stop == start)
                stop = System.nanoTime();

            total += (stop - start);

        }

        Log.d(LOG_TAG, String.format("nanoTime(): %f ms.", total
                                / (float)nRepeat / 1000000.0));
    }
}
```

Ohjelma 4.1. *Laskenta-aikojen mittauksessa käytetyn nanoTime() -metodin resoluution määritykseen käytetty menetelmä.*

fault, *high performance* ja *energy saver* -profiileilla.

Trepn Power Profiler mittaa tehonkulutusta akkukäytössä eli mittausten aikana kehitysalusta oli kytkettynä vain näyttöön HDMI-portin ja hiireen usb-portin kautta, jotta mittaussovellus saatiin käynnistettyä Android-käyttöliittymän kautta. Mittaukset tehtiin laitteen kokonaiskulutuksena, josta vähennettiin laitteen tyhjäkäynnin tehonkulutus.

Tehonkulutuksen mittaukset suoritettiin mittaussovelluksen suorittaessa verkon inferenssiä. Tämä suoritettiin siten, että inferenssiä suoritettiin ohjelmasilmukassa samalla kuvalla halutun ajan. Näin saadaan mitattua pelkästään inferenssin suorittamisen tehonkulutus. Tehonkulutuksen mittaukset suorittavan Trepn Power Profiler -sovelluksen mittausintervalliksi valittiin 100 ms, joka oli lyhin sovelluksen tarjoama intervalli. Trepn Power Profiler -sovellus tallentaa jokaisen yksittäisen mittauksen tuloksen *.csv* -tiedostoon. Näiden lisäksi sovellus laskee mittauksen ajalta tehonkulutuksen keskiarvon. Trepn Power Profiler -sovellus ja mittaus käynnistettiin ja lopetettiin mittaussovelluksesta käsin. Itse mittaus käynnistettiin juuri ennen inferenssiä suorittavaa ohjelmasilmukkaa ja lopetettiin heti silmukasta poistumisen jälkeen, jotta saatiin mitattua vain inferenssin tehonkulutus. Mittausten kestot olivat noin 5 - 8 minuuttia.

Koska Trepn Power Profiler mittaa järjestelmän kokonaistehonkulutusta mitattiin myös järjestelmän tyhjäkäyntitehonkulutus. Tämä suoritettiin käyttämällä *System.sleep()*- metodia[36], jolla mittaussovellus tauotettiin 5 minuutin ajaksi. Tauon ajalta mitattiin tehonkulutus, jolloin saadaan taustalla käyvän käyttöjärjestelmän itse laitteen keskimääräinen tyhjäkäyntitehonkulutus. Tämä menetelmä mittaa samalla myös tehonkulutusta mittaavan Trepn Power Profiler -sovelluksen tehonkulutuksen, jolloin se saadaan vähennettyä varsinaisesta neuroverkon tehonkulutuksesta.

Vähentämällä mitatuista kokonaiskulutuksista järjestelmän tyhjäkäynnin tehonkulutus saadaan inferenssin suorittamiseen kuluva keskimääräinen tehonkulutus. Lisäksi tehonkulutuksista laskettiin inferenssin energiankulutus. Inferenssin energian kulutus saadaan kertomalla tehonkulutus inferenssin suorittamiseen kuluvalla ajalla.

Akunkesto saadaan laskettua tehonkulutuksen (W) ja akun tehokapasiteetin (Wh) avulla. Akun tehokapasiteetti saadaan laskettua tehon yhtälöllä $P = UI$, jossa U on jännite ja I virta. Sijoittamalla akun käyttöjännite ja kapasiteetti ampeeritunteina(Ah) yhtälöön saadaan akun tehokapasiteetti wattitunteina(Wh). Jakamalla akun tehokapasiteetti tehonkulutuksella saadaan akunkesto tunteina.

4.3.4 Lämpötilanmittaus

Lämpötilanmittaukset suoritettiin Snapdragon Profilerilla, jolla kehitysalustasta saadaan keskusprosessorin ytimen ja grafiikkaprosessorin lämpötilasensorien mittaamat lämpötilat. Mittauksissa käytettiin COCO-kuvasarjalla koulutettua MobileNet SSD -neuroverkkoa. Keskusprosessorin lämpötilan mittauksiin käytettiin TensorFlow:n ja grafiikkaprosessorin SNPE:n verkkoa, koska TensorFlow:n verkko maksimoi keskusprosessorin ja SNPE grafiikkaprosessorin kuormituksen ajon aikana. SNPE:llä mittaukset suoritettiin *high performance*-profiililla. Lämpötilanmittaukset suoritettiin, koska Seppälä [32] oli opinnäytetyössään havainnut keskusprosessorin ylikuumenemista mobiilipuhelimessa, joka käyttää samaa Snapdragon-prosessoria.

Jokaisen mittauksen kesto oli noin 40 minuuttia, missä suoritettiin jatkuvaa inferenssiä yhdellä kuvalla. Käyttämällä yhtä kuvaa ja ajamalla sitä ohjelmasilmukassa saadaan maksimoitua laskentayksikön kuormitus. Mittauksien välissä kehitysalusta sammutettiin ja annettiin jäähtyä vähintään 15 minuuttia. Sekä keskusprosessorille että grafiikkaprosessorille suoritettiin kaksi mittausta, avoimena ja koteloituna. Kotelona käytettiin umpinaista pahvilaatikkoa, jonka ulkomitat olivat noin 5 cm * 20 cm * 30 cm. Huoneen lämpötila mittauksien aikana oli 22-24 $^{\circ}C$. Mittauksien aikana mitattiin myös inferenssiin kulunut aika, jotta saataisiin tieto mahdollisesta ylikuumenemisen vaikutuksesta laskentanopeuksiin.

5 MITTAUSTULOKSET

5.1 Tarkkuudet

Kuten luvussa 4.3.1 kuvattiin, tarkkuuden mittauksilla on tarkoitus osoittaa, että mittauksissa käytetyt toteutukset kohteenilmaisimesta toimivat oikein. Mittauksiin käytettiin kahta eri kuvasarjaa. Mittaustulokset esitetty taulukossa 5.1. Tarkkuudet taulukossa esitetty COCO- ja VOC-datasettien metriikoilla.

Taulukko 5.1. *Eri neuroverkkototeutuksien tarkkuudet*

Kuvasarja	helppo		vaikea	
	COCO	VOC	COCO	VOC
TensorFlow	0,506	0,804	0,209	0,296
OpenCL	0,506	0,804	0,209	0,296
SNPE CPU	0,506	0,804	0,209	0,296
SNPE GPU	0,506	0,804	0,207	0,299
SNPE GPU nms	0,506	0.804	0,207	0,299
SNPE DSP	0,482	0,697	0,220	0,317

Tarkkuuden mittaukset osoittavat, että konversiot sovelluskehysten välillä toimivat hyvin, eivätkä tarkkuudet niiden välillä eroa juurikaan. Ainoastaan SNPE:n DSP ajoympäristössä tarkkuudet ovat jonkin verran huonommat, mikä oli myös odotettavissa, ajoympäristössä käytettävästä 8-bittisestä kiintolukuformaatista johtuen.

Verrattaessa itsetoteutettua *non-max suppression* -operaatiota käyttävää verkkoa (SNPE GPU nms) vastaavaan SNPE toteutukseen (SNPE GPU) nähdään, että tarkkuuksissa näiden toteutuksien välillä ei ole eroa. Tulokset osoittavat, että tämän työn mittauksissa käytetty itsetehty toteutus *non-max suppression*-operaatiosta toimii kuten sen pitääkin ja mittaukset ovat siltä osin vertailukelpoisia.

Yksi mahdollinen virhelähde suoritetuissa tarkkuuden mittauksissa on pieni kuvasarjojen koko. Kuvasarjan pienen koon takia konversion tai laskennan epätarkkuus voi aiheuttaa virhettä tarkkuuden analyysin tulokseen. Tällainen virhe nähdään DSP:lle tehdyssä vaikean kuvasarjan tuloksessa, jonka mukaan DSP-ajoympäristö olisi jopa tarkempi vaikeiden kohteiden tunnistuksissa, mikä ei tietenkään ole mahdollista. Tämän vääristymän voi aiheuttaa laskennan epätarkkuudesta johtuva ero, joka aiheuttaa laskennan loppu-

tuloksen siirtymisen lähemmäksi todellisen annotaatioissa määritetyn kohteen kanssa. Kuvasarjan pienen koon takia jo yhdessä tunnistuksessa tapahtuva tuloksen siirtyminen seuraavalle kynnysarvolle voi aiheuttaa tällaisen vääristymän. DSP:llä saatu helpon kuvasarjan tulos onkin todennäköisesti lähempänä oikeaa. Voidaan olettaa, että suurella otoskoolla pienempi laskentatarkkuus huonontaa laskennan lopputulosten keskimääräistä tarkkuutta. Tässä tapauksessa kuitenkin kuvasarjojen koot ovat riittävät konversioiden toimivuuden osoittamiseen.

Joka tapauksessa nämä tarkkuuksien mittausten ja analyysin tulokset osoittavat, että konversiot sovelluskehysten välillä toimivat hyvin, eikä merkittävää eroa tarkkuuksien välillä havaittu. Ainoastaan DSP:llä dataformaatin takia tarkkuus on huonompi muihin verrattuna. Lisäksi MobileNet SSD:n tarkkuus verrattuna muihin kohteenilmaisimiin oli Huang et al. vertailussa todettu hyväksi (Luku 4.2.4). Kaikki mitatut toteutukset soveltuvat siis tarkkuuden puolesta käytettäväksi kohteenilmaisimena ajoneuvoluokittelijassa. Toki käytetty kohteenilmaisin vaatii tapauskohtaisen koulutuksen, jotta ilmaisimen tarkkuus saadaan optimoitua käyttökohteeseen ja -tarkoitukseen sopivaksi.

5.2 Laskentanopeudet

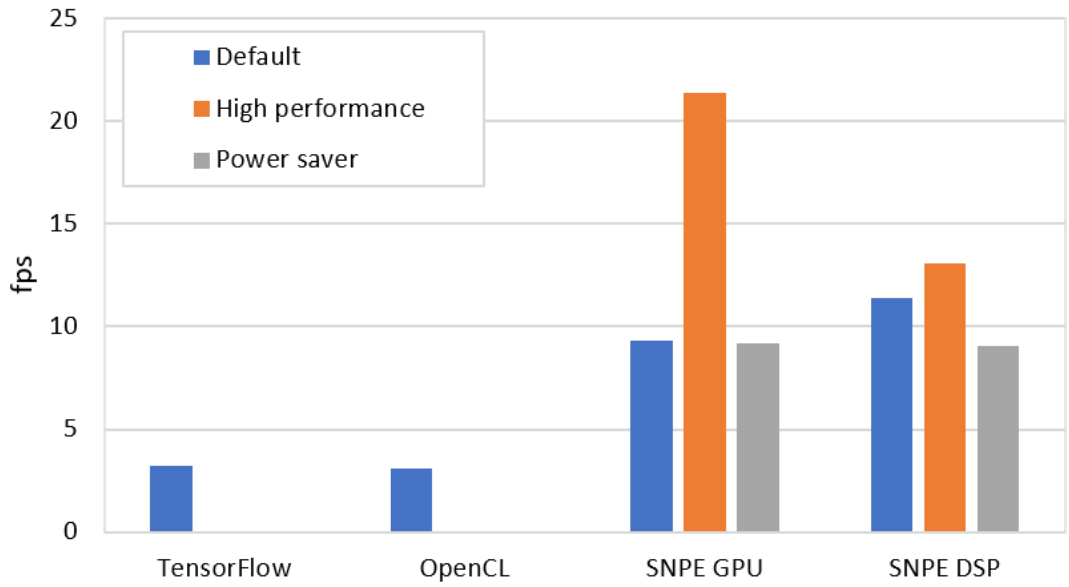
Laskentanopeuksien mittauksiin käytettiin MobileNet SSD -kohteenilmaisinta. Ilmaisimen kokonaislaskentanopeudet ja -ajat on esitetty taulukossa 5.2. Taulukossa laskenta-ajat millisekunteina ja laskentanopeudet inferenssiä per sekunti (fps). Laskentanopeudet on esitetty myös diagrammina kuvassa 5.1. Tarkempi kuvaus mittausjärjestelystä on esitetty luvussa 4.3.2.

Taulukko 5.2. *Eri neuroverkkototeutuksien kokonaislaskentanopeudet ja -ajat.*

	default		high performance		energy saver	
	ms	fps	ms	fps	ms	fps
TensorFlow	312	3,20				
OpenCL	326	3,07				
SNPE CPU	1400	0,72	809	1,24	1550	0,65
SNPE GPU	107	9,32	47	21,37	109	9,21
SNPE GPU nms	172	5,82	85	11,81	178	5,62
SNPE DSP	88	11,42	76	13,10	110	9,06

Mittauksissa käytetyn *nanoTime()*-metodin tarkkuudeksi saatiin 0,0017 ms. Tämä tarkkuus riittää hyvin laskenta-aikojen mittaamiseen, koska laskenta-ajat ovat pienimmillään kymmenien millisekuntien luokkaa ja muut mittauksien virhelähteet ovat huomattavasti suurempia.

Mittauksissa saatiin selkeitä eroja eri toteutusten laskentanopeuksissa. Hitaimmat nopeudet mitattiin SNPE:n CPU toteutuksella, joka oli *high performance* -profiililla noin 2,6



Kuva 5.1. *Toteutuksien laskentanopeudet pylväsdiagrammina.*

kertaa hitaampi kuin TensorFlow:n toteutus. SNPE:n CPU toteutuksen hitauden selittää se, että toteutusta ei ole optimoitu CPU:lle, mikä oli myös todettu SNPE:n dokumentaatiossa. Nopeimmat laskentanopeudet saatiin SNPE:n GPU- ja DSP-ajoympäristöissä. TensorFlow oli myös hieman nopeampi kuin OpenCL-toteutus.

Nopein toteutus mitatuista oli SNPE:n GPU toteutus *high performance* -profiililla, jonka kokonaislaskentanopeus oli 21,37 fps. GPU:n muilla profiileilla laskentanopeudet olivat lähes samat hieman yli 9 fps. DSP-yksiköllä eri profiilien välillä ei ollut yhtä suuria eroja kuin GPU:lla. DSP:n laskentanopeudet sijoittuivat noin 9-13,2 fps välille, *high performance* -profiilin ollessa nopein. OpenCL-toteutuksen laskentanopeus oli jopa hieman hitaampi kuin TensorFlow:n. OpenCL:n tulokset kertovat, ettei geneerisellä OpenCL-toteutuksella tässä tapauksessa saada etua suorituskyvyssä CPU:hun nähden. OpenCL-toteutuksen suorituskyvyn parantaminen vaatisi todennäköisesti sen optimointia laitteen GPU-arkkitehtuurille.

Koska DSP:n mittauksissa jouduttiin käyttämään itse toteutettua *non-max suppression* -operaatiota, suoritettiin mittaukset myös vastaavalla GPU toteutuksella (SNPE GPU nms), jotta nähtiin millainen vaikutus sillä on suorituskykyyn. Mittauksissa havaittiinkin selkeä ero itsetoteutetun *non-max suppression* -operaation ja SNPE:n toteutuksen välillä. Ero oli tässä tapauksessa *high performance* -profiililla 38 ms ja *power saver* 69 ms. Itsetoteutetun *non-max suppression* -operaation laskentanopeudeksi mitattiin noin 1 ms, joten tämä ei selitä eroa. Yksi selittävä tekijä tähän eroon voi olla ulostulokerroksien datansiirtomäärien ero. SNPE:n toteutuksen ulostulokerroksen solmujen data-alkioiden yhteenlaskettu määrä on 600 alkiota ja itse toteutetun 180198 alkiota. Datansiirtomäärä itse tehdyssä toteutuksessa on siis noin 300-kertainen verrattuna SNPE:n toteutukseen. Datansiirto-

määrien erot eivät kuitenkaan todennäköisesti voi kokonaan selittää eroa, koska ero ei ole sama eri profiilien välillä, vaikka datansiirtomäärät ovat. Ja koska SNPE:n kirjastot eivät ole julkisia, on mahdotonta sanoa, miten eri operaatiot on toteutettu ja miten tällainen muutos ulostulokerroksissa vaikuttaa itse verkon toimintaan. Joten mitään varmaa selitystä näiden toteutusten väliselle laskentanopeuksien erolle ei voida antaa. Edellisten havaintojen perusteella on mahdollista, että myös DSP:llä tapahtuu itsetoteutettua *non-max suppression* -operaatiota käytettäessä samanlaista hidastumista, mutta tässäkään tapauksessa näiden mittauksien perusteella ei mitään varmaa voida sanoa.

Luokkien ja tunnistuksien määrän vaikutukset laskentanopeuteen on esitetty kuvassa 5.3. Tuloksista nähdään, että luokkien tai tunnistuksien määrällä ei ole merkittävää vaikutusta laskentanopeuteen. GPU:lla tehdyissä mittauksissa laskentanopeudet olivat lähes yhtä suuret. CPU:lla TensorFlow:n verkolla ajettaessa saatiin molemmissa mittauksissa 10 ms erot siten, että pienemmällä luokkien ja tunnistuksien määrällä laskentanopeus oli hitaampi.

Laskentanopeudet (ms)		
Luokkia (kpl)	1	90
TensorFlow	314	304
SNPE GPU	88,5	88,6

Laskentanopeudet (ms)		
Tunnistuksia (kpl)	1	27
TensorFlow	304	294
SNPE GPU	88,6	90,1

Taulukko 5.3. *Laskentanopeuksien mittaustulokset eri luokkien ja tunnistusten määrille. Vasemmalla tulokset 1 ja 90 luokan verkoille ja oikealla kuville, joissa tunnistuksia 1 ja 27 kappaletta.*

Tarkasteltaessa käytetyn neuroverkon rakennetta nähdään, että esimerkiksi luokkien määrä vaikuttaa neuroverkon loppupään konvoluutio-operaatioiden dimensioihin. Vertailtaessa mittauksissa käytettyjä verkkoja, joissa luokkien määrät olivat 1 ja 90 kappaletta, ensimmäisen verkon parametrien kokonaismäärä on noin $5{,}5 \cdot 10^6$ ja suoritettujen MAC (Multiply Accumulate) -operaatioiden $1{,}093 \cdot 10^9$ ja jälkimmäisessä verkossa vastaavat määrät ovat $6{,}8 \cdot 10^6$ ja $1{,}205 \cdot 10^9$ Vertailussa 90 luokan verkon parametrien määrä oli noin 24 % suurempi kuin 1 luokan verkossa ja MAC-operaatioita oli noin 10 % enemmän.

Nämä parametrien ja laskentaoperaatioiden erot rajoittuvat kuuteen konvoluutio-operaatioon. Kyseisten operaatioiden (*Pointwise_X_L*), joihin luokkien määrä vaikuttaa, laskenta-ajat nähtävillä liitteessä A. Liitteen mittaukset on suoritettu 90 luokan verkolla. Esimerkiksi SNPE GPU:n *default*-profiililla näiden kuuden operaation yhteenlaskettu laskenta-aika on 1,44 ms, joka on vain noin 1,3 prosenttia verkon kokonaislaskenta-ajasta. Luokkien määrän vaikutus verkon kokonaislaskenta-aikaan on niin pieni, että se hukkuu esimerkiksi käyttöjärjestelmän keskeytyksistä aiheutuvaan laskenta-aikojen variaatioon.

Android-käyttöjärjestelmän keskeytyksien ajankohtaa tai sen varaamien resurssien määrää on mahdoton ennakoida, mutta tällaisilla moniajoon kykenevillä käyttöjärjestelmillä niitä tapahtuu, kun käyttöjärjestelmä suorittaa sille kuuluvia taustaprosesseja. Pääasiassa Android-käyttöjärjestelmä käyttää kuitenkin näihin toimintoihin CPU:ta ja tällöin verkon

laskentaan käytettävissä olevien ytimien määrä vaihtelee. Tämä vaihtelu näkyykin selkeämmin CPU:lla laskennan suorittavassa TensorFlow:n verkolla tehdyissä mittauksissa. CPU:lla pisin mitattu laskenta-aika oli 475,8 ms ja lyhin 239,3ms, joten laskenta-aikojen vaihteluväli tässä mittauksessa oli 236,5 ms. SNPE:n GPU:lla vastaavat arvot olivat 80,6 ms, 35,7 ms, 44,9 ms ja DSP:llä 128,7 ms, 50,7 ms ja 78,0 ms. SNPE:n GPU:lla ja DSP:llä tehdyissä mittauksissa vaihtelu on huomattavasti pienempää, koska tällöin CPU:ta käytetään laskentaan vain verkon jälkiprosessointiosion joissakin osissa. Myös tunnistuksien määrän nähdään, verkon rakennetta tarkasteltaessa, vaikuttavan hieman verkon laskentaan. Tämä kuitenkin rajoittuu lähinnä *non-max suppression* -operaatiossa tapahtuviin IoU vertailuihin ollen siten vaikutuksiltaan edellistä huomattavasti vähäisempi.

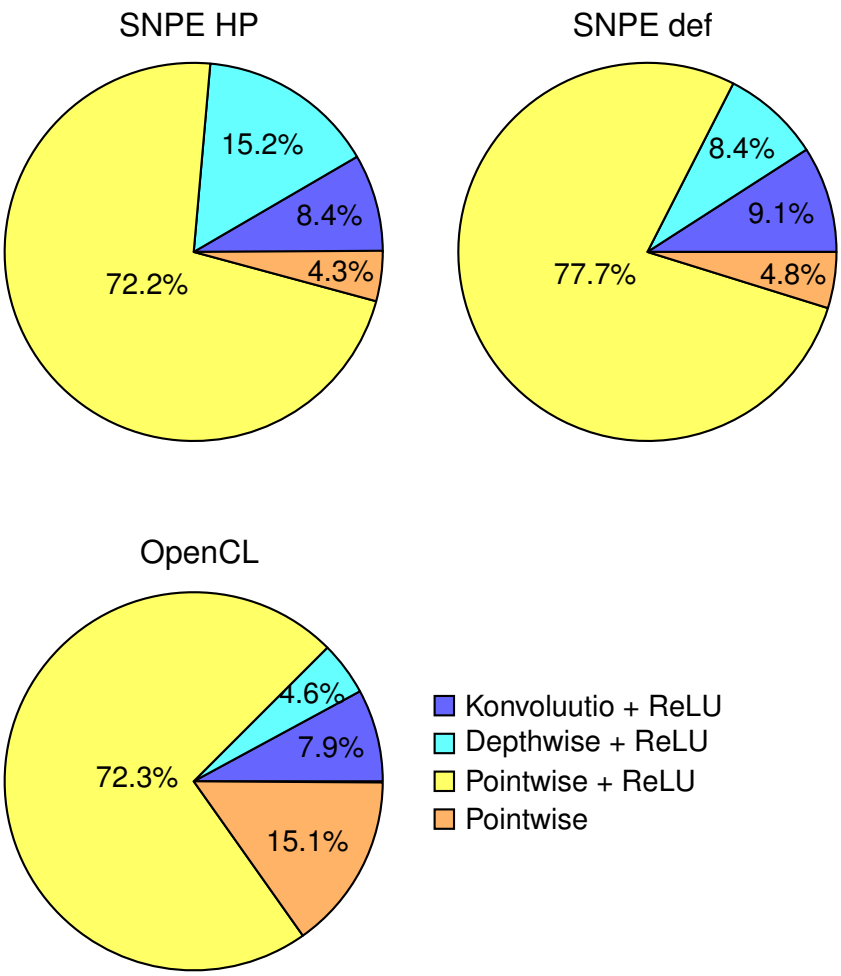
OpenCL-kernelien laskenta-ajat sekä OpenCL että SNPE GPU toteutuksien osalta esitetty liitteessä A. Lisäksi eri konvoluutio-operaatioiden yhteenlasketut laskenta-ajat taulukossa 5.4 ja laskenta-aikojen suhteelliset osuudet kuvassa 5.2.

Liitteen taulukosta voidaan erottaa MobileNet SSD-kohteenilmaisimen neuroverkon eri rakenteet. Taulukon 27 ensimmäistä konvoluutio-operaatiota muodostavat SSD- kohteenilmaisimen selkärankana toimivan MobileNet- luokittelijan, aina *Konvoluutio_13_1*- operaatioon asti. Tämän jälkeen seuraavat 8 operaatiota ovat SSD:n lisäkonvoluutiokerroksia, joiden avulla tunnistetaan kohteita eri mittakaavoissa. Viimeiset 12 operaatiota muodostavat piirrekartoista kohteiden sijainnit ja luokat. Data etenee neuroverkossa lineaarisesti ensimmäisestä konvoluutiokerroksesta aina SSD:n viimeiseen *Konvoluutio_5*-lisäkerrokseen asti. Lopun 12 kerrosta saavat sisääntulonsa neuroverkon aikaisemman rakenteen eri kerroksista. Näiden viimeisten kerrosten yhteydet merkitty kerrosten nimiin numeroilla 0 - 5.

Taulukko 5.4. *Konvoluutio-operaatioiden yhteenlasketut laskenta-ajat.*

Konvoluutio operaatiot	OpenCL (ms)	Default (ms)	HP (ms)
Konvoluutio + ReLU	14,9	5,23	2,10
Depthwise + ReLU	8,72	4,83	3,02
Pointwise + ReLU	136,3	44,4	18,0
Pointwise	28,5	2,73	1,06
Yhteensä	188,5	57,2	24,2

Taulukossa 5.4 nähdään eri konvoluutio-operaatioiden yhteenlasketut laskenta-ajat. Yhteenlasketuista ajoista nähdään, että suurin osa kernelien suoritukseen kuluvasta ajasta kuluu *pointwise*-operaatioiden suorittamiseen. Tämä nähdään myös kuvassa 5.2, jossa esitetty yhteenlaskettujen laskenta-aikojen suhteelliset osuudet. *Pointwise*-operaatioiden suuri suhteellinen osuus oli toki odotettu tulos, kuten esimerkiksi Howard et al. osoittivat, MobileNet-luokittelijassa *pointwise*-operaatioiden osuus parametrien kokonaismäärästä oli 74.6 % ja MAC-laskentaoperaatioiden osuus 94,9 % [14]. Suhteellisista osuuksista nähdään myös suuri ero neuroverkon loppuosan lineaaristen *pointwise*-operaatioiden



Kuva 5.2. *OpenCL-kernelien laskenta-aikojen suhteelliset osuudet.*

osalta OpenCL ja SNPE:N toteutusten välillä. SNPE:n suorituskykyprofiileja vertailtaessa nähdään, että suurin ero profiilien laskenta-aikojen välillä saadaan *pointwise*-operaatioista, joka nähdään myös diagrammeista, joissa *pointwise*-operaation suhteellinen osuus SNPE:n *high performance*-profiililla on kaikkein pienin, vain noin 76,5 % kun se *default*-profiililla oli 82,5 % ja OpenCL-toteutuksella 87,4 %.

Suurin mahdollinen virhelähde laskentanopeuksien mittauksissa voi aiheutua käyttöjärjestelmän keskeytyksistä. Kuten edellisessä kappaleessa todettiin, käyttöjärjestelmä varaa tarvittaessa CPU ytimiä taustaprosessien suorittamiseen keskeyttämällä muiden sovellusten säikeitä. Näiden keskeytysten ajankohtaa ja niiden vaatimien resurssien määrää on mahdoton ennakoida. Tämä tarkoittaa sitä, että neuroverkon laskentaan käytettävissä olevien CPU ytimien määrä voi vaihdella mittausten välillä. Tätä keskeytyksistä aiheuttamaa virhettä pyrittiin minimoimaan kasvattamalla mittausten määrää, jolloin keskeytyksistä johtuvien yksittäisten piikkien osuus mitattujen laskenta-aikojen keskiarvossa pienenee. Mittausten määrän kasvattamista rajoittaa kuitenkin lämpötilanmittauksissa (luku 5.4) havaittu CPU ytimen ylikuumeneminen, joka aiheuttaa kellotaajuuden alenemisen ja sitä kautta vaikuttaa laskenta-aikoihin. Toisaalta, jos keskeytyksiä tapahtuu säännöllisesti ei mittausten määrän lisääminen pienennä näistä säännöllisistä keskeytyksistä johtuvaa

virhettä. Mittausten määrä valittiin niin, ettei CPU ytimen ylikuumenemista mittauksien aikana tapahdu ja yksittäisten keskeytyksien aiheuttama virhe olisi mahdollisimman pieni.

Yhteenvetona laskentanopeuksien mittauksista voidaan todeta, että mittauksissa kohteenilmaisimena käytetyn neuroverkon suorittaminen CPU:lla on hidasta verrattuna GPU- tai DSP-yksiköillä kiihdytettyyn laskentaan. Lisäksi käytetylle prosessoriarkkitehtuurille optimoimattomalla neuroverkon toteutuksella ei saatu suorituskykyetua CPU:hun nähden, kuten nähdään OpenCL-toteutuksen laskentanopeuksista. Sen sijaan prosessoriarkkitehtuurille optimoidulla toteutuksella saatiin tässä tapauksessa moninkertainen etu optimoimattomaan verrattuna. OpenCL-kernelien laskenta-ajoista nähdään, että optimoidun ja optimoimattoman toteutuksen välinen ero on kaikissa käytetyn neuroverkkoarkkitehtuurin konvoluutio-operaatioissa. Edellä mainittujen seikkojen perusteella tämä havaittu ero voidaan yleistää myös muihin konvoluutioneuroverkkoihin, ja mittauksissa havaitun suorituskyvyn parannuksen kaltainen hyöty voidaan saavuttaa myös muilla konvoluutioneuroverkoilla.

5.3 Tehonkulutukset

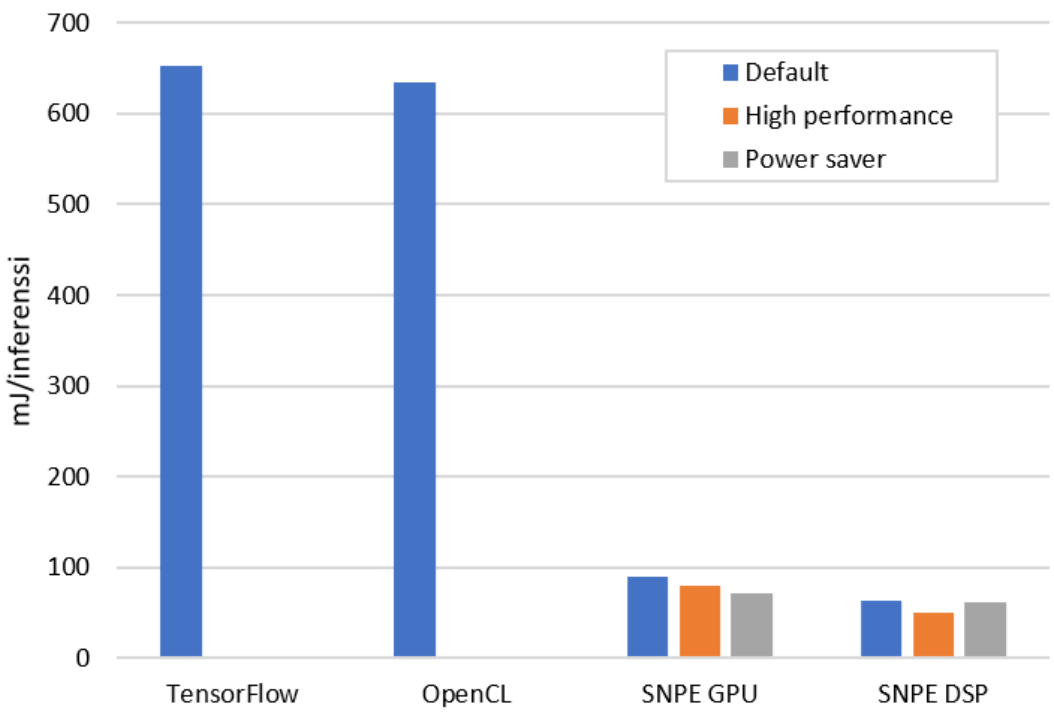
Tehonkulutuksen mittaukset suoritettiin MobileNet SSD- verkolle. TensorFlow käyttää verkon suorittamiseen CPU:ta, OpenCL-verkko GPU:ta. SNPE:n verkkoa ajettiin sekä GPU:lla että DSP:llä *default*, *high performance* sekä *power saver* -profiileilla. Mittauksissa mitattiin systeemin kokonaistehonkulutus, josta vähennettiin tyhjäkäynnin tehonkulutus, joka oli 1250 mW. Tarkempi kuvaus mittauksista esitetty luvussa 4.3.3.

Tehonkulutukset käytetylle verkolle on esitetty taulukossa 5.5. Taulukossa kulutukset on esitetty sekä tehonkulutuksina milliwatteina että energiankulutuksina millijouleina per inferenssi. Energiankulutukset millijouleina per inferenssi on esitetty myös pylväsdiagrammina kuvassa 5.3.

Taulukko 5.5. *Tehon- ja energian kulutukset eri toteutuksille.*

	default		high performance		power saver	
	mW	mJ/f	mW	mJ/f	mW	mJ/f
TensorFlow	2085	652				
OpenCL	1945	635				
SNPE GPU	831	89,2	1707	79,9	657	71,3
SNPE DSP	726	63,5	647	49,4	559	61,8

Tarkasteltaessa tehonkulutuksia havaitaan, että TensorFlow:n ja OpenCL-verkon tehonkulutukset ovat huomattavasti korkeammat kuin SNPE:n verkon, pois lukien GPU:n *high performance*-profiili, jolla tehonkulutus on lähes yhtä suuri kuin ensin mainituissa. Energiankulutuksissa taas havaitaan vielä suurempi ero verrattaessa TensorFlow:n- ja OpenCL-verkkoja SNPE-verkkoon, mikä johtuu SNPE:llä käytettyjen laskentayksikköjen nopeam-

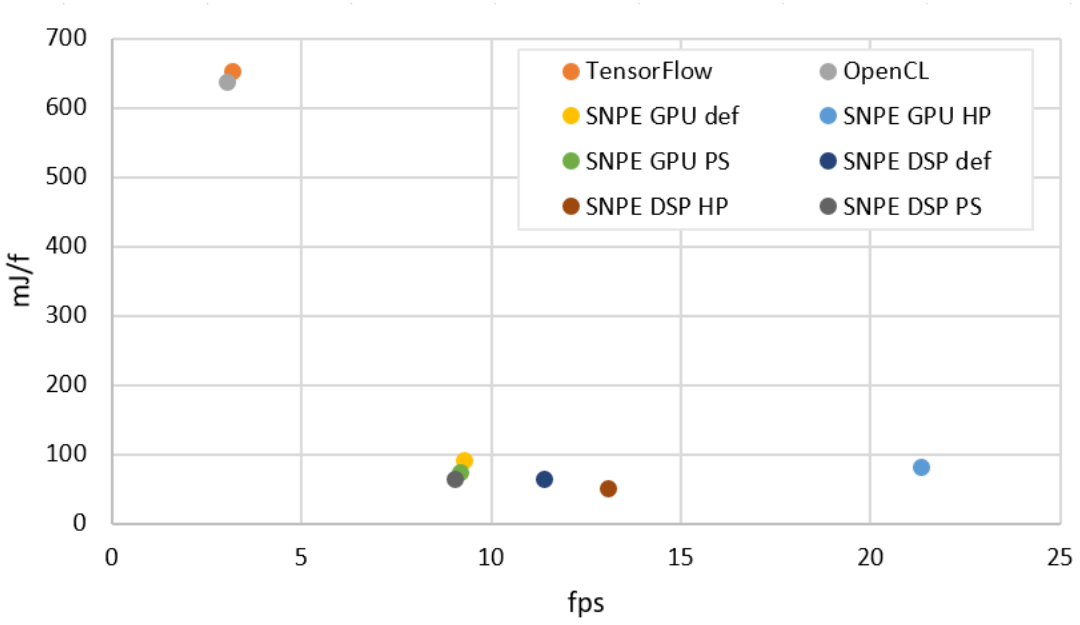


Kuva 5.3. *Energiankulutukset eri toteutuksille.*

masta laskentanopeudesta. Selkeimmin tämä näkyy SNPE:n GPU:n *high performance* -profiilin energiankulutuksessa, joka korkeasta tehonkulutuksesta huolimatta asettuu *default*- ja *power saver* -profiilienien väliin. Mittauksissa pienimmät tehon- ja energiankulutukset saatiin SNPE:llä käytettäessä DSP:tä verkon ajoon. DSP:llä eri suorituskykyprofiilien välillä ei havaittu suuria eroja kuten GPU:ta käytettäessä, vaan myös *high performance* -profiililla tehonkulutus jäi jopa *default*-profiilia pienemmäksi.

Kuvassa 5.4 esitetty suoritusnopeuden suhde inferenssin suorittamiseen kuluvaan energiaan. Kuva kertoo selkeästi käytetylle laitearkkitehtuurille optimoidun ja optimoimattoman toteutuksen eron energiatehokkuudessa ja laskentanopeudessa. Optimoimattomat TensorFlow:n ja OpenCL:n toteutukset kohteenilmaisimesta kuluttavat paljon energiaa ja ovat hitaita verrattuna SNPE:n optimoituun toteutukseen. Energiatehokkuudessa SNPE:n toteutus jää kaikilla profiileilla alle 100 millijoulen, kun taas TensorFlow:n ja OpenCL:n toteutukset kuluttavat yli 600 mJ energiaa. Energiankulutukseltaan tehokkaimpaan DSP:n *high performance* -profiiliin verrattuna TensorFlow kuluttaa jopa 13 kertaa enemmän energiaa.

Akunkestojen laskennassa tarvittavan akun kapasiteetin arvoksi saatiin 15,2 Wh. Akun kestot laskettiin kahdella tavalla, käyttäen kokonaistehonkulutusta ja pelkästään inferenssin tehonkulutusta. Akun kestoaika *h* laskettiin käyttämällä mitattua järjestelmän kokonaistehonkulutusta ja *h'* käyttämällä taulukossa 5.5 esitettyjä arvoja, jotka saatiin vähentämällä kokonaistehonkulutuksesta tyhjänkäyntitehonkulutus. Tulokset esitetty taulukossa 5.6 ja diagrammina kuvassa 5.5.



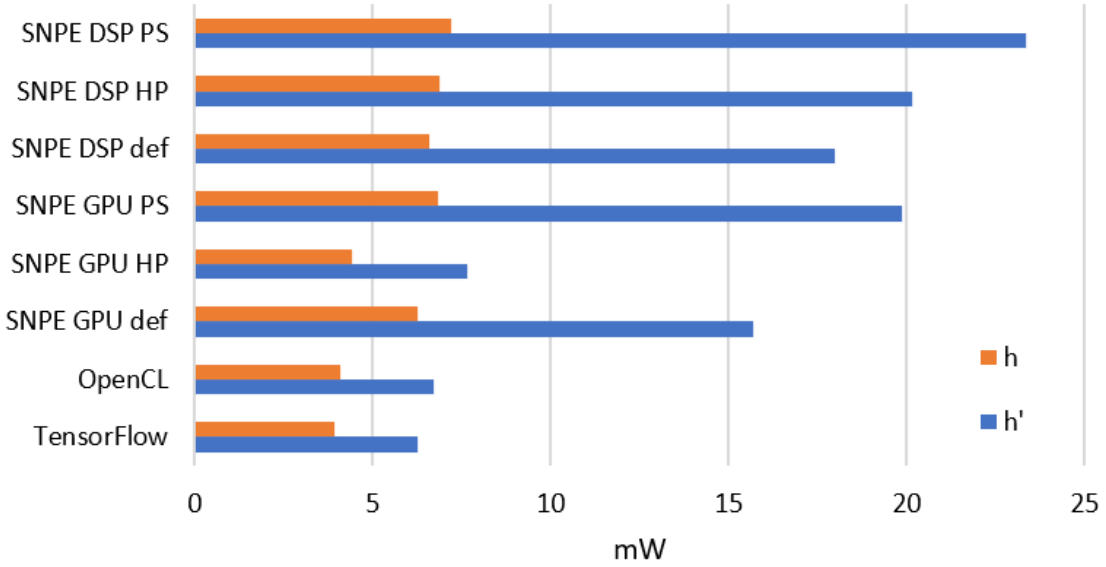
Kuva 5.4. *Suoritusnopeuden suhde inferenssin suorittamiseen kuluvaan energiaan.*

Akunkestoista nähdään, että niissä on suuria eroja suoritinyksikköjen ja suorituskykyprofiilien välillä, niin kuin tehonkulutusten perusteella sopii odottaakin. Lyhimmät akunkestot saatiin TensorFlow:n ja OpenCL toteutuksille, joilla akunkesto jäi noin 4 tuntiin. SNPE:n toteutukset olivat yleisesti parhaat akunkestoltaan eikä niissä ollut suuria eroja, pois lukien *high performance* -profiili GPU:lla, jonka suorituskyvyn maksimoinnista johtuva suuri tehonkulutus lyhentää akunkeston lähes TensorFlow:n ja OpenCL:n tasolle. SNPE:n suorituskykyprofiileilla *power saver* -profiililla saadaan parhaat akunkestot suoritusyksiköistä. DSP:llä tosin myös *high performance* -profiilin akunkesto oli lähellä *power saver* -profiilin akunkestoa. DSP:n *power saver* profiililla saatu pisin akunkesto oli lähes kaksi kertaa pidempi kuin TensorFlow:n CPU toteutuksen.

Taulukko 5.6. *Akunkestoajat järjestelmän kokonaistehonkulutuksella h ja pelkästään inferenssin tehonkulutuksilla h'.*

	default		high performance		power saver	
	h	h'	h	h'	h	h'
TensorFlow	3,9	6,3				
OpenCL	4,1	6,7				
SNPE GPU	6,3	15,7	4,4	7,6	6,8	19,9
SNPE DSP	6,6	18,0	6,9	20,2	7,2	23,3

Mittauksien mahdollisia virhelähteitä ovat käyttöjärjestelmän satunnaisten taustaprosessien aiheuttama lisä tehonkulutukseen, itse mittaukseen käytetyn Trepn Power Profiler -sovelluksen aiheuttama tehonkulutus. Käyttöjärjestelmän taustaprosessien virhettä pyrittiin poistamaan sulkemalla niitä käytöstä, esimerkiksi asettamalla käyttöjärjestelmä lentokonetilaan mittauksien ajaksi. Lentokonetila poistaa laitteen radioverkot käytöstä. Tä-



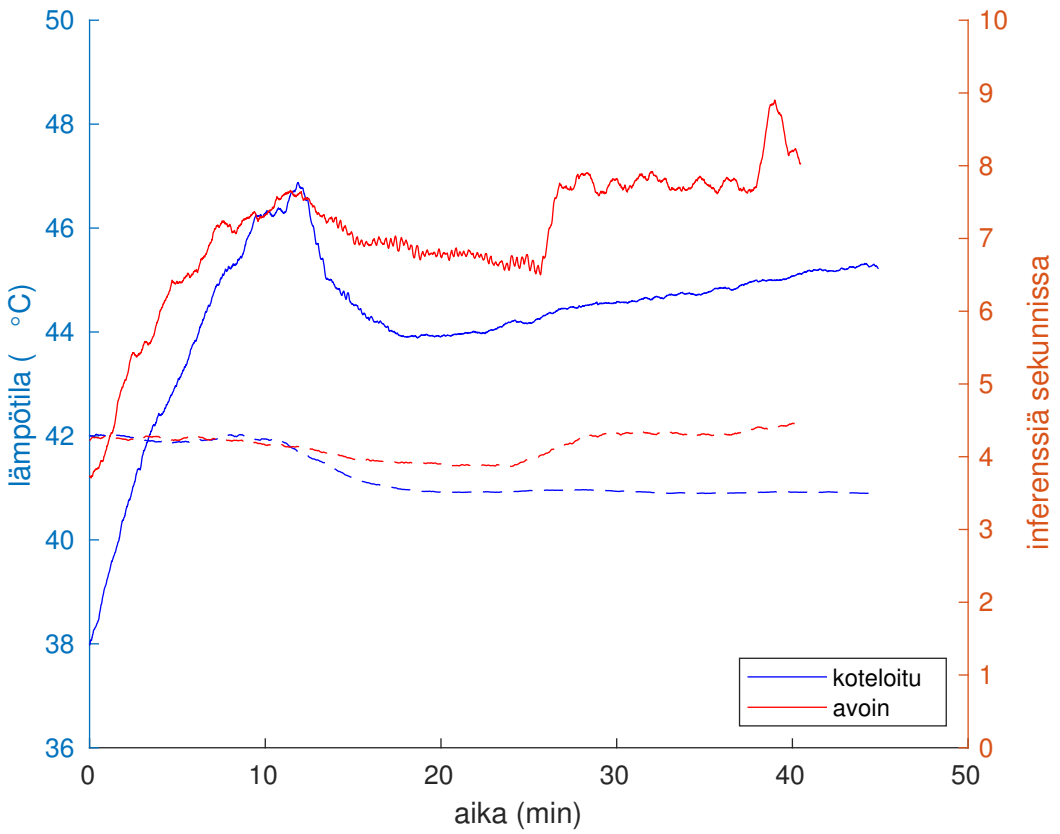
Kuva 5.5. *Taulukon 5.6 akun kestoajat pylväsdiagrammina.*

män lisäksi tehonkulutuksen mittauksien kesto pyrittiin valitsemaan siten, että se minimoi käyttöjärjestelmän satunnaisista taustaprosesseista johtuvat virheet.

Mittausten kestoa ylhäältä päin rajoittaa lämpötilanmittauksissa (luku 5.4) havaittu CPU ytimen ylikuumeneminen, jonka aiheuttama kellotaajuuden laskeminen voi myös aiheuttaa virhettä mittauksiin. Mittauksien kestot olivat 5-8 minuuttia, jolloin ne olivat kestoltaan pidempiä kuin Trepn Power Profilerille suositeltu vähintään 2 minuuttia ja lyhyempiä kuin lämpötilanmittauksissa havaitut ylikuumenemisesta johtuvan suorituskyvyn alentumisen alkamisajankohdat.

Trepn Power Profiler -sovelluksen aiheuttama tehonkulutuksen lisäys poistuu tuloksista erotettaessa tyhjäkäyntitehonkulutus mitatuista koko systeemin tehonkulutuksista, koska myös tyhjäkäyntitehonkulutuksen mittaus tehtiin samalla sovelluksella, jolloin sen aiheuttama tehonkulutuksen lisäys on mitattu tyhjäkäyntitehonkulutukseen. Muiden mahdollisten taustalla toimivien sovellusten, joiden toimintaa taustalla ei ole pystytty sulkemaan, aiheuttama mahdollinen virhe poistuu myös tyhjäkäyntitehonkulutuksen erotuksen mukana mittauksista. Edellä mainituilla tavoilla virhelähteet on saatu minimoitua siten, että ne eivät aiheuta tehon- ja energiankulutuksien analysoinnin ja vertailun kannalta merkittävää virhettä.

Tehonkulutuksen mittauksissa havaittiin selkeä ero suoritinyksiköille optimoitujen ja optimoimattomien toteutuksien välillä, kuten havaittiin laskentanopeuksien mittauksissakin. Optimoimattomien TensorFlow:n ja OpenCL-toteutuksien tehonkulutukset olivat huomattavasti suuremmat kuin optimoiduilla SNPE:n toteutuksilla. Ainoastaan SNPE:n GPU:lla *high performance*-profiililla tehonkulutukset olivat, suorituskyvyn maksimoinnin takia, lähes TensorFlow:n ja OpenCL:n tasolla. Tehonkulutukset vaikuttavat suoraan myös akun-



Kuva 5.6. *Keskusprosessorin lämpötilan ja laskentanopeuksien mittaustulokset yhtäjaksoisen inferenssin suorittamisen ajalta. Lämpötilat on esitetty yhtäjaksoisella viivalla ja laskentanopeudet katkoviivalla.*

kestoon, ja akunkestoissa nähdäänkin samalla tavalla ero optimoitujen ja optimoimattomien toteutuksien välillä. Näiden mittausten perusteella tehonkulutuksen ja akunkeston näkökulmasta DSP-yksikkö oli mittauksien paras käytetyn neuroverkon laskentaan, tosin suurta eroa ei ollut SNPE:n GPU *high performance*-profiiliin nähden.

5.4 Lämpötilanmittaukset

Lämpötilan mittauksissa mitattiin CPU:n lämpötilaa yhtäjaksoisen MobileNet SSD- kohteenilmaisimen suorituksen aikana. Lämpötilan mittauksien aikana suoritettiin myös ilmaisimen kokonaislaskenta-ajan mittaukset. Mittauksiin käytettiin TensorFlow:n toteutusta ja mittauksien kestot olivat noin 40 - 45 minuuttia. Mittaustulokset esitetty kuvassa 5.6. Kuvassa lämpötilat esitetty yhtäjaksoisella viivalla ja suoritusnopeudet katkoviivalla.

Mittaustuloksista nähdään selviä lämpötilojen ja laskentanopeuksien vaihteluja mittauksien aikana. Selkeimmin tämä näkyy koteloidulla laitteella tehdyssä mittauksessa, jossa lämpötila ensin nousee noin 10 minuutin ajan ja laskee sitten nopeasti. Laskun jälkeen lämpötila pysyy ylintä mitattua lämpötilaa alempana. Samaan aikaan lämpötilan laskun

kanssa myös laskentanopeus laskee jääden mittauksen loppuun alemmalle tasolle. Myös koteloimattomalla laitteella nähdään lämpötilan ja suoritusnopeuden lasku, mutta molemmat nousevat takaisin ylemmälle tasolle. Vastaavaa ylikuumenemista havaitsi myös Seppälä opinnäytetyössään.

Tällainen lämpötilan ja laskentanopeuden vaihtelu johtuu CPU ytimien ylikuumenemisesta. Ytimen ylikuumetessa käyttöjärjestelmä laskee ytimien kellotaajuutta, joka vähentää ytimien kuormitusta ja seurauksena lämpötila ja laskentanopeus laskee. Tämä on tietenkin ongelma, jos laskentanopeus on kriittinen ominaisuus sovelluksessa. Käytetyssä laitteessa tosin CPU oli tässä tapauksessa hitain ajoympäristö ja tehonkulutuskin oli suurinta. GPU:lla ja DSP:llä vastaavaa lämpötilojen tai laskentanopeuksien vaihtelua ei havaittu, mikä on tietenkin loogista, koska tällöin pääosa laskentakuormasta siirtyy CPU:lta kyseisille yksiköille.

Mittauksissa mitatut lämpötilat eivät todennäköisesti tässä tapauksessa kerro ytimien lämpötiloja. Laitteessa on useita lämpötilasensoreita, mutta sensoreiden sijainnit eivät ole julkista tietoa. Mittaussovellus käyttää näistä sensoreista saatavaa dataa lämpötilan määritykseen, mutta tietoa siitä mitä sensoria tai sensoreita sovellus käyttää ei ole saatavilla. Tässä tapauksessa itse lämpötila-arvo ei kuitenkaan ole tärkeä, vaan mittauksien tuloksissa nähty lämpötilan ja laskentanopeuksien käyttäytyminen yhtäjaksoisen suorituksen aikana. Ytimien todelliset lämpötilat olivat todennäköisesti huomattavasti korkeammat. Mittauksien perusteella voidaan todeta, että käytettäessä CPU:ta kohteenilmaisemisen suorittamiseen ylikuumeneminen voi aiheuttaa suorituskyvyn alenemisen ja se on syytä ottaa huomioon.

6 YHTEENVETO

Työn tarkoituksena oli tutkia, soveltuuko mobiililaite neuroverkkopohjaisen ajoneuvoluokittelijan suorittamiseen ja miten laitteen eri prosessointiyksiköt soveltuvat tähän tehtävään. Näitä kysymyksiä tutkittiin erityisesti suorituskyvyn ja energiatehokkuuden näkökulmista. Mittaukset suoritettiin Snapdragon 835 -prosessorilla, joka on yleinen prosessointiyksikkö mobiililaitteissa ja kohteenilmaisimena käytettiin MobileNet SSD -arkkitehtuuria.

Suorituskykyyn liittyvät laskentanopeudet (fps) ja tehonkulutukset (mW) on esitetty taulukossa 6.1. Taulukosta nähdään TensorFlow:n, OpenCL ja SNPE toteutuksilla tehdyt mittaukset. SNPE:n osalta mittaukset tehtiin kolmella suorituskykyprofiililla sekä GPU:lla ja DSP:llä. TensorFlow:n toteutus käyttää CPU:ta ja OpenCL GPU:ta neuroverkon laskentaan. Mittaustuloksista nähdään selkeä ero prosessoriarkkitehtuurille optimoidun SNPE:n ja optimoimattomien TensorFlow:n ja OpenCL:n toteutuksien välillä. Nopein oli GPU:lla laskennan suorittava SNPE:n *high performance* -profiili. Hitain taas myös GPU:ta käyttävä OpenCL toteutus, jonka geneerisellä toteutuksella ei saada etua suorituskyvyssä TensorFlow:n vastaavaan nähden. Samanlainen ero optimoidun ja optimoimattoman toteutuksen välillä nähdään myös tehonkulutuksissa. DSP oli vähäkulutuksisin kaikilla suorituskykyprofiileilla, ja TensorFlow ja OpenCL toteutukset kuluttivat eniten.

Taulukko 6.1. *MobileNet SSD -kohteenilmaisimen eri toteutuksien ja suorituskykyprofiilien laskentanopeudet (fps) ja tehonkulutukset (mW).*

	default		high performance		energy saver	
	fps	mW	fps	mW	fps	mW
TensorFlow	3,20	2085				
OpenCL	3,07	1945				
SNPE GPU	9,32	831	21,37	1707	9,21	657
SNPE DSP	11,42	726	13,10	647	9,06	559

Mittauksiin käytetty laite soveltuu hyvin niin neuroverkkopohjaisen ajoneuvoluokittelijan kuin yleisesti konvoluutioneuroverkkojen suorittamiseen. Käytettäessä GPU:ta tai DSP:tä neuroverkon suorittamiseen saadaan tarpeeksi laskentanopeutta ja energiatehokkuutta, jotta laitetta voidaan käyttää akkuvirralla jopa tunteja. Kohteenilmaisimena käytetty MobileNet SSD- neuroverkkoarkkitehtuuri, jonka suunnittelussa on otettu huomioon sulautettujen laitteiden rajoitteet, soveltuu myös hyvin tähän käyttötarkoitukseen. Tässä tapauksessa GPU:lla saatiin paras suorituskyky ja DSP taas oli energiatehokkain. Molemmat

prosessoriyksiköt soveltuvat neuroverkkopohjaisen ajoneuvoluokittelijan reaaliaikaiseen suorittamiseen mittauksiin käytetyllä laitteella.

Laitteen käyttöön neuroverkkosovelluksissa liittyy kuitenkin tiettyjä rajoituksia. Grafiikka- ja digitaalisen signaaliprosessoreiden tehokas hyödyntäminen vaatii prosessoriarkkitehtuuria tukevan ohjelmakirjaston käyttöä. Yleiskäyttöisellä neuroverkkokirjastolla toteutettu neuroverkkosovellus on hidas ja kuluttaa paljon energiaa, eikä tällöin saada suorituskykytai tehokkuusetua keskusprosessoriin nähden. Keskusprosessorilla neuroverkon suoritus on siis hidasta ja tehotonta, mutta voi myös aiheuttaa keskusprosessorin ylikuumenemisen, joka entisestään huonontaa suorituskykyä. Lisäksi Snapdragon-prosessoreja tukeva ohjelmakirjasto oli mittausten suoritushetkellä hieman keskeneräinen. Keskeneräisyydestä johtuen ohjelmakirjastosta puuttuu tuki joillekin kohteenilmaisimissa käytettäviin operaatioihin ja se voi rajoittaa joidenkin kohteenilmaisinarkkitehtuurien käyttöä.

Neuroverkkoja on myös mahdollista ajaa laittessa rinnakkain erillisissä ohjelmasäikeissä. Tällöin kyse on kuitenkin toisistaan erillisistä neuroverkoista, mikä mahdollistaa eri kiihdytinyksikköjen tai neuroverkkoarkkitehtuurien käytön rinnakkain. Tämä voisi tarkoittaa esimerkiksi eri kameroista saatavan videodatan rinnakkaista analysointia. Käytettäessä samaa datalähdettä ja neuroverkkoa kahdessa säikeessä voidaan saada yli 50 % parannus suorituskyvyssä, kuten Seppälä opinnäytetyössään osoittaa [32]. Saman säikeen sisällä ei kuitenkaan ole mahdollista käyttää eri kiihdytinyksikköjä, vaan ohjelmisto käyttää aina kiihdytinyksikköä, jolle säie on alustettu.

Erilaisten tekoäly- ja konenäkösovellusten määrä tulee tulevaisuudessa lisääntymään ja samalla myös sovellukset monimutkaistuvat. Usein näitä sovelluksia käytetään erilaisissa sulautetuissa järjestelmissä, kuten itseohjautuvissa ajoneuvoissa. Lisääntyvä monimutkaisuus ja korkeat suorituskykyvaatimukset asettavat painetta kehittää myös laskentaa suorittavaa laitteistoa. Laitteistokehitys tässä tapauksessa tarkoittaa erilaisia neuroverkkospesifisiä yksikköjä tai prosessoreja, joilla voidaan saada merkittäviä etuja suorituskyvyssä ja energiatehokkuudessa yleiskäyttöisiin grafiikka- ja keskusprosessoreihin nähden.

Työ osoittaa, että mobiilialustoja voi käyttää koneoppimissovellusten suorittamiseen jopa akkukäyttöisesti. Käytämällä prosessoriarkkitehtuurille optimoituja ohjelmakirjastoja voidaan parantaa sekä suorituskykyä että energiatehokkuutta huomattavasti.

LÄHDELUETTELO

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu ja X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. English. Maaliskuu 2016. URL: https://arxiv.org/abs/1603.04467.

[2] *Arm ML Processor*. URL: https://developer.arm.com/ip-products/processors/machine-learning/arm-ml-processor (viitattu 07. 01. 2020).

[3] *Arm NN - Software Developer Kit*. URL: https://www.arm.com/products/silicon-ip-cpu/machine-learning/arm-nn (viitattu 07. 01. 2020).

[4] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang ja Z. Zhang. *MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems*. English. Joulukuu 2015. URL: https://arxiv.org/abs/1512.01274.

[5] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro ja E. Shelhamer. *cuDNN: Efficient Primitives for Deep Learning*. English. Lokakuu 2014. URL: https://arxiv.org/abs/1410.0759.

[6] F. Chollet et al. *Keras*. https://github.com/fchollet/keras. 2015.

[7] R. Collobert, S. Bengio ja J. Mariéthoz. *Torch: a modular machine learning software library*. Idiap-RR Idiap-RR-46-2002. IDIAP, 2002.

[8] J. Deng, W. Dong, R. Socher, L. Li, K. Li ja L. Fei-Fei. ImageNet: A large-scale hierarchical image database. Teoksessa: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. ID: 1. 2009, 248–255. DOI: 10.1109/CVPR.2009.5206848.

[9] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn ja A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88.2 (2010). ID: Everingham2010, 303–338. ISSN: 1573-1405. DOI: 10.1007/s11263-009-0275-4. URL: https://doi.org/10.1007/s11263-009-0275-4.

[10] *FDDB: Face Detection Data set and Benchmark*. URL: http://vis-www.cs.umass.edu/fddb/ (viitattu 27. 03. 2019).

[11] R. Girshick, J. Donahue, T. Darrell ja J. Malik. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.1 (2016), 142–158. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2015.2437384.

[12] K. He, X. Zhang, S. Ren ja J. Sun. Deep Residual Learning for Image Recognition. *arXiv.org* (2015). ID: proquest2083823373.

[13] M. A. Hoque, M. Siekkinen, K. N. Khan, Y. Xiao ja S. Tarkoma. Modeling, profiling, and debugging the energy consumption of mobile devices. English. *ACM Computing Surveys* 48.3 (2015). URL: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954349303&doi=10.1145%2f2840723&partnerID=40&md5=152c6bd57ff52404bce6556e36b55d73.

[14] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto ja H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 16. huhtikuuta 2017. URL: https://arxiv.org/abs/1704.04861.

[15] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama ja K. Murphy. Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors. Teoksessa: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Heinäkuu 2017, 3296–3297. DOI: 10.1109/CVPR.2017.351.

[16] *Intrinsyc*. URL: https://www.intrinsyc.com/ (viitattu 14. 03. 2019).

[17] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama ja T. Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. en (kesäkuu 2014). URL: https://arxiv.org/abs/1408.5093v1.

[18] A. Krizhevsky, I. Sutskever ja G. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM* 60.6 (2017). ID: acm3065386, 84–90. ISSN: 0001-0782. DOI: 10.1145/3065386.

[19] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár ja C. L. Zitnick. Microsoft COCO: Common Objects in Context. *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu ja A. C. Berg. SSD: Single Shot MultiBox Detector. English (joulukuu 2015). DOI: 10.1007/978-3-319-46448-0_2. URL: https://arxiv.org/abs/1512.02325.

[21] W. S. McCulloch ja W. Pitts. A logical calculus of the ideas immanent in nervous activity. English. *Bulletin of Mathematical Biology* 52.1 (1990), 99–115. ISSN: 0092-8240. DOI: 10.1016/S0092-8240(05)80006-0. URL: https://www.sciencedirect.com/science/article/pii/S0092824005800060.

[22] *Microsoft cognitive toolkit (CNTK), an open source deep-learning toolkit*. 2018. URL: https://docs.microsoft.com/en-us/cognitive-toolkit/ (viitattu 08. 05. 2019).

[23] G. Nguyen, S. Dlugolinsky, M. Bobák, V. Tran, Á. L. García, I. Heredia, P. Malík ja L. Hluchý. Machine Learning and Deep Learning frameworks and libraries for large-scale data mining: a survey. *Artificial Intelligence Review* (2019). ID: Nguyen2019. ISSN: 1573-7462. DOI: 10.1007/s10462-018-09679-z. URL: https://doi.org/10.1007/s10462-018-09679-z.

[24] J. Nickolls, I. Buck, M. Garland ja K. Skadron. Scalable Parallel Programming with CUDA. *Queue* 6.2 (maaliskuu 2008), 40–53. ISSN: 1542-7730. DOI: 10.1145/1365490.1365500. URL: http://doi.acm.org/10.1145/1365490.1365500.

[25] *NVIDIA TensorRT - Programmable Inference Accelerator*. URL: https://developer.nvidia.com/tensorrt (viitattu 07. 01. 2020).

[26] *ONNX. Open Neural Network Exchange.
*. 2017. URL: https://github.com/onnx/onnx..

[27] *Open-Q 835 Development Kit based on the Snapdragon 835 (APQ8098) Processor User Guide*. Intrinsyc. 2017. URL: https://support.intrinsyc.com/attachments/download/1604/Intrinsyc_Open-Q_835_Development_Kit_User_Guide_v1.0.pdf.

[28] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga ja A. Lerer. *Automatic differentiation in PyTorch*. 2017. URL: https://openreview.net/forum?id=BJJsrmfCZ.

[29] *Qualcomm® Snapdragon™ Profiler User Guide*. 21. syyskuuta 2018.

[30] J. Redmon, S. Divvala, R. Girshick ja A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv.org* (2016). ID: proquest2078987609.

[31] R. Al-Rfou et al. *Theano: A Python framework for fast computation of mathematical expressions*. English. Toukokuu 2016. URL: https://arxiv.org/abs/1605.02688.

[32] S. Seppälä. Performance of Neural Network Image Classification on Mobile CPU and GPU. Tohtorinväitöskirja. Huhtikuu 2018, 86. URL: sipiseppala.fi/docs/masters_seppala_2018.pdf.

[33] *Snapdragon Neural Processing Engine SDK, Reference Guide*. URL: https://developer.qualcomm.com/docs/snpe/index.html (viitattu 13. 05. 2019).

[34] J. E. Stone, D. Gohara ja G. Shi. OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Computing in Science & Engineering* 12.3 (2010). ID: 1, 66–73. ISSN: 1521-9615. DOI: 10.1109/MCSE.2010.69.

[35] *System (Java Platform SE 7). nanoTime*. 6. lokakuuta 2018. URL: https://docs.oracle.com/javase/7/docs/api/java/lang/System.html#nanoTime() (viitattu 13. 03. 2019).

[36] *System (Java Platform SE 7). sleep*. 6. lokakuuta 2018. URL: https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html#sleep(long) (viitattu 13. 03. 2019).

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke ja A. Rabinovich. Going Deeper with Convolutions (2014). ID: arxiv1409.4842.

A LIITE: KONVOLUUTIO-OPERAATIOIDEN LASKENTA-AJAT

Taulukko A.1. *MobileNet SSD -neuroverkon konvoluutio-operaatioiden OpenCL-kernelien laskenta-aikoja SNPE- ja OpenCL-toteutuksille.*

MobileNet SSD			
Konvoluutio-operaatiot	OpenCL (ms)	SNPE Default (ms)	SNPE High Performance (ms)
Konvoluutio + ReLU	9,12	1,08	0,49
Depthwise_1 + ReLU	1,27	0,80	0,41
Pointwise_1 + ReLU	7,76	2,2	0,96
Depthwise_2 + ReLU	0,91	0,62	0,44
Pointwise_2 + ReLU	5,64	1,98	0,81
Depthwise_3 + ReLU	1,46	0,78	0,42
Pointwise_3 + ReLU	10,7	3,69	1,42
Depthwise_4 + ReLU	0,57	0,34	0,23
Pointwise_4 + ReLU	5,85	1,99	0,76
Depthwise_5 + ReLU	0,85	0,48	0,25
Pointwise_5 + ReLU	12,4	3,82	1,45
Depthwise_6 + ReLU	0,30	0,20	0,16
Pointwise_6 + ReLU	6,46	1,95	0,74
Depthwise_7 + ReLU	0,53	0,26	0,18
Pointwise_7 + ReLU	12,5	3,85	1,62
Depthwise_8 + ReLU	0,53	0,26	0,20
Pointwise_8 + ReLU	12,6	3,87	1,61
Depthwise_9 + ReLU	0,52	0,26	0,18
Pointwise_9 + ReLU	12,5	3,90	1,76

Konvoluutio operaatiot	OpenCL (ms)	Default (ms)	HP (ms)
Depthwise_10 + ReLU	0,53	0,26	0,19
Pointwise_10 + ReLU	12,5	3,85	1,57
Depthwise_11 + ReLU	0,53	0,26	0,18
Pointwise_11_0 + ReLU	12,5	3,86	1,67
Depthwise_12 + ReLU	0,27	0,14	0,09
Pointwise_12 + ReLU	7,01	2,50	0,94
Depthwise_13 + ReLU	0,45	0,16	0,09
Pointwise_13_1 + ReLU	13,8	4,94	1,95
Pointwise + ReLU	3,6	1,56	0,58
Konvoluutio_2 + ReLU	4,70	2,63	1,05
Pointwise + ReLU	0,29	0,24	0,09
Konvoluutio_3 + ReLU	0,50	0,58	0,23
Pointwise + ReLU	0,11	0,13	0,05
Konvoluutio_4 + ReLU	0,41	0,62	0,23
Pointwise + ReLU	0,10	0,12	0,05
Konvoluutio_5 + ReLU	0,21	0,32	0,11
Pointwise_0_S	1,66	0,28	0,15
Pointwise_0_L	8,76	0,28	0,16
Pointwise_1_S	1,03	0,47	0,18
Pointwise_1_L	14,6	0,53	0,19
Pointwise_2_S	0,21	0,25	0,08
Pointwise_2_L	1,48	0,27	0,09
Pointwise_3_S	0,09	0,12	0,04
Pointwise_3_L	0,32	0,15	0,04
Pointwise_4_S	0,09	0,11	0,04
Pointwise_4_L	0,12	0,14	0,04
Pointwise_5_S	0,06	0,07	0,02
Pointwise_5_L	0,08	0,07	0,02

S = sijainnit, L = luokat