PUHUJANTUNNISTUS KONVOLUUTIONEUROVERKOLLA HYÖDYNTÄEN REUNALASKENTAA

Lappeenranta-Lahti University of Technology LUT

Kandidaatintutkinto Laskennallinen tekniikka, Kandidaatintyö

2023

Alex Karonen

Tarkastaja: TkT Henri Petrow

KIITOKSET

Haluan lyhyesti kiittää kaikkia henkilöitä, jotka ovat osallisuudestaan riippumatta auttaneet tämän projektin loppuunsaattamisessa. Erityiset kiitokset kuluneesta keväästä kuuluvat ohjaajilleni Henri Petrowille sekä Lasse Lensulle, ilman teitä ei tätä kandidaatintyötä edes olisi. Kiitokset kanssaopiskelijoille mahtavasta työympäristöstä sekä henkisestä tuesta, jonka takaatte.

Lappeenranta, Suomi Alex Karonen
19.5.2023

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Laskennallinen tekniikka

Alex Karonen

Puhujantunnistus konvoluutioneuroverkolla hyödyntäen reunalaskentaa

Kandidaatintyö
2023
36 sivua, 11 kuvaa, 11 taulukkoa, 1 liite
Tarkastajat: TkT Henri Petrow

Hakusanat: puhujantunnistus, reunalaskenta, ohjelmoitava porttipiiri, FPGA, mel-kepstrikertoimet, MFCC, konvoluutioneuroverkot, Pynq-Z2

Tässä kandidaatintyössä pyrittiin toteuttamaan puhujantunnistava konvoluutioneuroverkko ohjelmoitavalla porttipiirillä (FPGA). Työssä käytettiin Mozilla Common Voice -projektin suomenkielistä puheaineistoa. Aineisto muunnettiin konvoluutioneuroverkolle sopivaksi mel-taajuuskerroin-spektrogrammiksi. Neuroverkko pyrittiin muuntamaan Pynq-Z2 -kehitysalustalle ajettavaan muotoon käyttäen hls4ml-kirjastoa mutta alustan resurssit eivät riittäneet tähän. Sen sijaan tutkittiin erikokoisten neuroverkkojen resurssivaatimuksia FPGA-toteutuksena ja verrattiin teoreettisia luokittelutarkkuuksia hls4ml:n simulointia hyödyntäen.

Työssä luodut neuroverkot olivat luokittelutarkkuuden perusteelala varteenotettavia käytännön sovelluksiin. Konvoluutioneuroverkot yleisesti vaativat paljon laskentaresursseja ja FPGA-alustoilla nämä resurssit ovat rajalliset. Tästä syystä toteutetut neuroverkot pyrittiin pitämään mahdollisimman pieninä kuitenkin tavoitellen järkeviä luokittelutarkkuuksia. Mikään toteutetuista malleista ei vastannut Pynq-Z2:n resursseja synteesiraporttien pohjalta. Työn mallien resurssivaatimusarviot vastaavat ennestään tuotettuja tuloksia vaikka estimaatit eivät ole suoraan vertailukelpoisia jo olemassa olevien tulosten kanssa.

LYHENTEET

ANN	artificial neural network	keinotekoinen neuroverkko
BRAM	block random access memory	lohkomuisti
CNN	convolutional neural network	konvoluutioneuroverkko
DSP	digital signal processor	digitaalinen signaaliprosessori
FF	flip-flop	kiikku
FPGA	field-programmable gate array	ohjelmoitava porttipiiri
GMM	Gaussian mixture models	gaussiset sekoitemallit
HMM	hidden Markov models	Markovin mallit
IoT	internet of things	laitteiden internet
IP	intellectual property -core	uudelleenohjelmoitava ydin
LPCC	linear predictive cepstral coefficients	lineaarisen ennustuksen kertoimet
LUT	lookup table	hakutaulu
MFCC	mel-frequency cepstral coefficients	mel-kepstrikertoimet
PLP	perpetual linear prediction	jatkuva lineaarinen ennustus
ReLu	rectified linear unit	korjattu lineaarinen yksikkö
SoC	system-on-a-chip	perinteinen systeemipiiri
STFT	short-time Fourier transform	lyhyen aikaikkunan Fourier-muunnos

SISÄLTÖ

1 JOHDANTO

1.1 Tausta

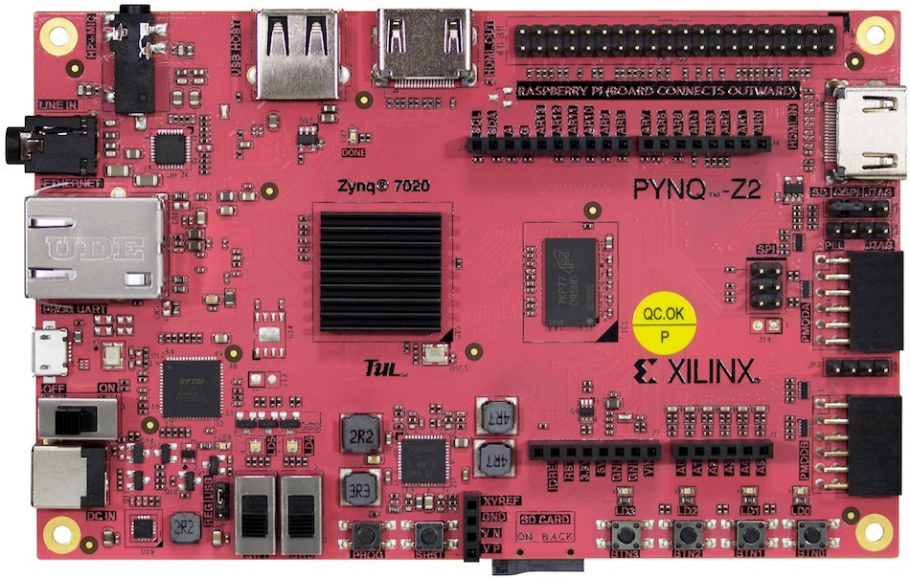
Puheentunnistus on nykypäivänä pitkälle tutkittu ja ajankohtainen aihe. Suurten yritysten, kuten Googlen ja Microsoftin kehittämät kielentulkintamallit ovat osa jokapäiväistä elämäämme, sillä kyseiset mallit löytyvät jo lähes jokaisesta mobiililaitteesta. Alunperin kun kyseistä alaa lähdettiin kehittämään, tavoitteena oli saada koneet tuottamaan lähes luonnollista kieltä. Tänä päivänä sovellukset mahdollistavat ihmispuheen imitoinnin ja muuntamisen [1] lähes reaaliaikaisesti jopa niin tarkasti, että kuulijan on vaikea hahmottaa onko puhuja kone vai ihminen. Sovellukset kuten Voice.Ai [2] ovat saavuttaneet suuren suosion tämän päivän sosiaalisessa mediassa.

Puhetta tunnistavilla sekä puhuvilla laitteilla on suuri hyöty lääketieteellisessä tarkoituksessa. Mykät, sokeat ja kuurot kuuluvat niiden ihmisten joukkoon, jotka hyötyvät laitteista ja sovelluksista joissa hyödynnetään puheentunnistusta. Kuurot voivat käyttää kommunikoinnin apuna puheesta-tekstiksi -sovelluksia [1], jotka löytyvät tänä päivänä lähes jokaisesta mobiililaitteesta jossain muodossa. Sokeat henkilöt pystyvät hyödyntämään samankaltaisia sovelluksia esimerkiksi lomakkeiden täyttämiseen netissä tai tekstistäpuheeksi -sovellusten [1] avulla heille voidaan lukea tekstiä erilaisilta päätteiltä. Samoin mykät voivat kommunikoida verbaalisesti puhetta tuottavien sovellusten [1] avulla.

Puheentunnistus-malleja voidaan hyödyntää myös turvallisuusratkaisuina. Ihmisen ääntä voidaan käyttää biometrisenä tunnisteena [3] esimerkiksi kaksivaiheisessa tunnistautumisessa erinäisiin sovelluksiin. Tämän lisäksi ihmisen äänestä voidaan tunnistaa myös erilaisia tunteita, kuten vihaa tai iloa, joiden avulla voidaan automaattisesti analysoida ihmisten käytöstä [4]. Tämä usein yhdistetään kuvamateriaaliin, kuten videokuvaan, jotta voidaan tunnistaa esimerkiksi julkisilla paikoilla uhkaavia käytösmalleja.

Puheentunnistus vaatii tänä päivänä tehokasta laitteistoa toimiakseen luotettavasti. Monet nykyaikaisista puheentunnistavista malleista hyödyntävät pilvipalveluita, mikä helpottaa niiden käyttöä paikassa kuin paikassa, kunhan toimiva internetyhteys on saatavilla. Kaikissa sovelluksissa tämä ei välttämättä kuitenkaan ole haluttu ratkaisu jos tiedonkäsittely halutaan toteuttaa lokaalisti ja itsenäisesti, ilman kolmannen osapuolen apua. Tässä astuvat esiin korttitietokoneet ja sulautetut järjestelmät, kuten työssä käytetty TUL Pynq-Z2 (kuva 1) [5]. Niiden kompakti koko ja integroituvuus muun laitteiston kanssa mahdollistavat erinäköisten koneoppivien mallien sulauttamisen jo olemassa oleviin laitteistoihin.

Edellä mainittu videokuvamateriaalin tulkinta mahdollisesti puheentunnistusta hyödyntäen [6] voidaan toteuttaa sulauttamalla jo olemassa olevaan valvontajärjestelmään puheentunnistava moduuli, joka tulkitsee valvontajärjestelmän tallentamaa äänidataa.



Kuva 1. Työn pääkohdealusta XUP Pynq-Z2 ohjelmoitava porttipiiri. [5]

1.2 Tutkimusongelma, tavoitteet ja rajaus

Tämä työ keskittyy puhujantunnistuksessa käytettävän keinotekoisen neuroverkon suorituskyvyn mittaamiseen ja vertailuun sekä perinteisenä suoritintoteutuksena että ohjelmoitavaan porttipiiriin (engl. Field-Programmable Gate Array, FPGA) pohjautuvana kiihdytettynä toteutuksena. Työssä puhujantunnistus toteutetaan luomalla keinotekoinen neuroverkko käyttäen Googlen Tensorflow Python -kirjastoa. Kyseinen neuroverkko opetetaan ja testataan käyttäen Mozilla Common Voice -projektin keräämää aineistoa. Neuroverkko käännetään opetuksen jälkeen Pynq Z2 -korttitietokoneen ja sen FPGA-kiihdytinmoduulin ajettavaan muotoon käyttäen hls4ml-kirjastoa ja sen hyödyntämää Vivado-nimistä laiteohjelmointisovellusta. Mallin suorituskykyä verrataan perinteisen pöytätietokoneen keskusprosessoritoteutuksen ja Pynq-Z2:lla ajetun toteutuksen välillä. Lisäksi pohditaan mahdollisia ongelmia ja haittapuolia sekä etuja mitä eri toteutukset tuovat mukanaan.

Työ on rajattu käytettävän aineiston, ohjelmistojen sekä laitteiston mukaan sopivaksi. Työssä ei keskitytä puheentunnistavan mallin kehitykseen tai optimointiin, vaan pyritään soveltamaan jo olemassa olevia toteutuksia työssä käytettävän aineiston ja laitteiston mukaisiksi. Käytettävä aineisto käsitellään mallille sopivaan muotoon.

1.3 Työn rakenne

Tässä raportissa tutustutaan aluksi yleisesti puheen- ja puhujantunnistukseen kirjallisuuskatsauksen muodossa osiossa 2. Lisäksi samassa osiossa esitellään kuinka koneoppivia malleja voidaan tänä päivänä toteuttaa kevyillä korttitietokoneilla ja erityisesti FPGA-alustoilla, kuten työssä käytetyllä Pynq-Z2:lla. Osiossa 3 esitellään käytetyt menetelmät sekä ohjelmistot pureutumatta niiden yksityiskohtiin. Osio 4 käsittelee lyhyesti työssä toteutetut kokeet sekä käytetyn aineiston. Tulokset ja niiden analysointi esitetään raportin osiossa 5. Työn viimeinen kappale 6 tiivistää työssä toteutetut asiat sekä kokoaa työn tulokset ja pohtii mahdollisia jatkotutkimusmahdollisuuksia.

2 OPPIVA PUHEENTUNNISTUS JA OHJELMOITAVAT PORTTIPIIRIT

2.1 Puheentunnistus

Puheen- ja puhujantunnistus elää tänä päivänä kulta-aikaansa. Suuret yritykset, kuten Google ja Microsoft ovat investoineet viime vuosien aikana puheentunnistus-sovelluksien kehitykseen, joka on edistänyt alan kehittymistä suuresti. Puheentunnistus-sovelluksien käyttötarkoitukset vaihtelevat laajasti terveydenhuollosta tietoturvallisuuteen. Biometrisenä tunnisteena puheentunnistus houkuttaa hyödyntäjiään sen halpuudella, helppoudella ja tehokkuudellaan [7].

Puheentunnistuksessa yleisimmin käytetyt menetelmät jakautuvat perinteisiin menetelmiin ja koneoppimismenetelmiin. Perinteiset menetelmät pitävät sisällään templaattimalleja sekä tilastollisia malleja, kuten Markovin mallit (engl. Hidden Markov Model, HMM) [8] ja gaussiset sekoitemallit (engl. Gaussian Mixture Model, GMM) [9]. Tänä päivänä kuitenkin koneoppivat mallit, kuten keinotekoiset neuroverkot voittavat perinteiset mallit suorituskyvyllään [10]. Tarkemmin ottaen nykypäivän tehokkaimmat puheen- ja puhujantunnistus mallit ovat konvoluutioneuroverkkoja (engl. Convolutional Neural Networks, CNNs) [11].

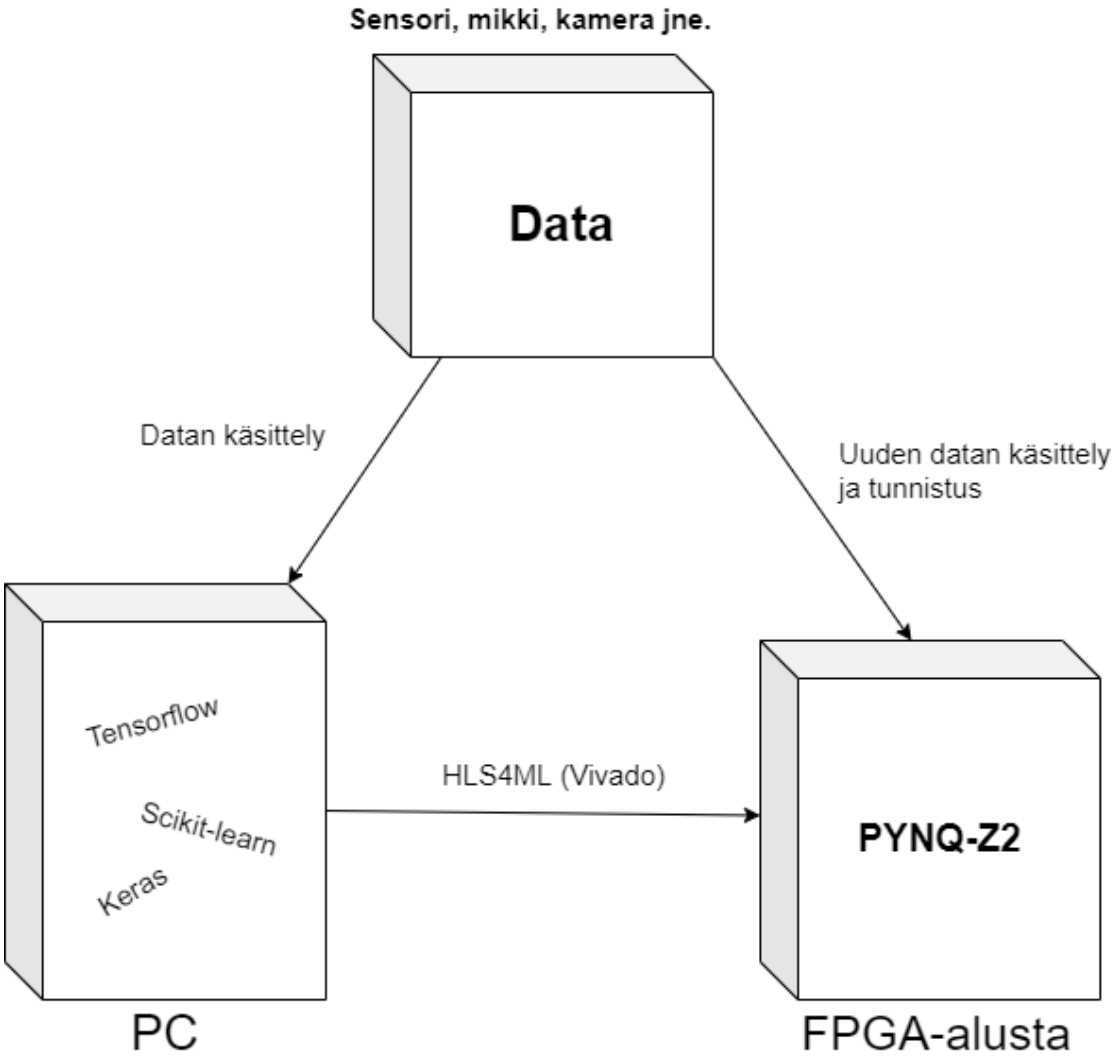
Koneoppivien mallien luomisessa merkittävimmät vaikuttavat tekijät ovat piirteiden erottaminen aineistosta sekä itse hahmontunnistavan mallin kehitys. Esimerkkejä moderneissa malleissa useimmin käytetyistä piirteistä ovat lineaarisen ennustuksen kepstrikertoimet (engl. Linear Predictive Cepstral Coefficients, LPCC), mel-kepstrikertoimet (engl. Mel-Frequency Cepstral Coefficcients, MFCC) [12] sekä jatkuva lineaarinen ennustus (engl. Perpetual Linear Prediction, PLP) [13]. LPCC ja MFCC ovat tiedetysti tehokkaita menetelmiä puhujantunnistuksessa piirteiden erottamiseksi, joista jälkimmäinen (MFCC) [14] on tänä päivänä hyvin yleisessä käytössä sen suorituskyvyn ansiosta [11].

2.2 Koneoppiminen ohjelmoitavilla porttipiireillä

Laitteiden internet (engl. Internet of Things, IoT) on nykypäivänä yksi keskustelluimmista teknologian aiheista. Datavirtojen kasvaessa yhä suuremmiksi, tarvitaan laitteistoja jotka pysyvät suurien datamassojen käsittelyssä mukana. Perinteisten systeemipiiriin (engl. System-on-a-Chip, SoC) pohjautuvien toteutusten käyttökelpoisuus sulautetuissa järjestelmissä suurien datamassojen käsittelyssä on laskenut vuosien varrella niiden tehovaatimusten kasvaessa epäkäytännöllisiksi [15]. SoC:llä tarkoitetaan tässä sulautettuja mikrokontrollereita ja mikroprosessoreita. FPGA-moduulit ovat yleensä jonkin perinteisen prosessorin sekä FPGA-kiihdyttimen sisältäviä SoC:jä. Esimerkiksi työssä käytetty Pynq-Z2 sisältää XC7Z020-1CLG400C SoC:n jossa on FPGA-kiihdytinmoduulin lisäksi Dual ARM® Cortex™-A9 MPCore™ -moniydin suoritin. Lähivuosina on alettu tutkimaan FPGA-alustojen käyttökelpoisuutta IoT-toteutuksissa, varsinkin koneoppimisen puolella niiden laskentatehokkuuden ja uudelleenohjelmoitavuuden johdosta.

FPGA-reunalaskentaa hyödyntävät alustat sopivat teoriassa hyvin koneoppimiseen niiden hyödyntämän rinnakkaislaskennan nopeuden ansiosta [16]. Lisäksi FPGA-moduulit ovat energiatehokkaampia sekä monesti halvempia kuin niiden perinteiset SoC-vastineet. Suuret prosessoriyhtiöt, kuten AMD (Xilinx) ja Intel ovat tänä päivänä merkittävästi mukana FPGA-moduulien tuotannossa, joka on kiihdyttänyt kyseisen teknologian kehitystä viime vuosina.

Koneoppimisen toteuttaminen FPGA-alustoilla toimii uudelleenohjelmoitavien ydinten (engl. Intellectual Property -cores, IPs) avulla, jotka ovat uudelleenkäytettäviä logiikkayksiköitä. IP:t voidaan ohjelmoida ja tarpeen mukaan uudelleenohjelmoida käyttötarkoitukseensa sopivaksi käyttäen matalan tason laitteistokieliä, kuten Verilogia. Jotkin IP:t pystytään ohjelmoimaan jopa käyttäen C-ohjelmointikieltä sen matalan tason systeemiohjelmointimahdollisuuksien ansiosta. Koneoppivan neuroverkkomallin toteutus FPGA-alustalla seuraa jotakuinkin kuvassa 2 esitettyä kaaviota.

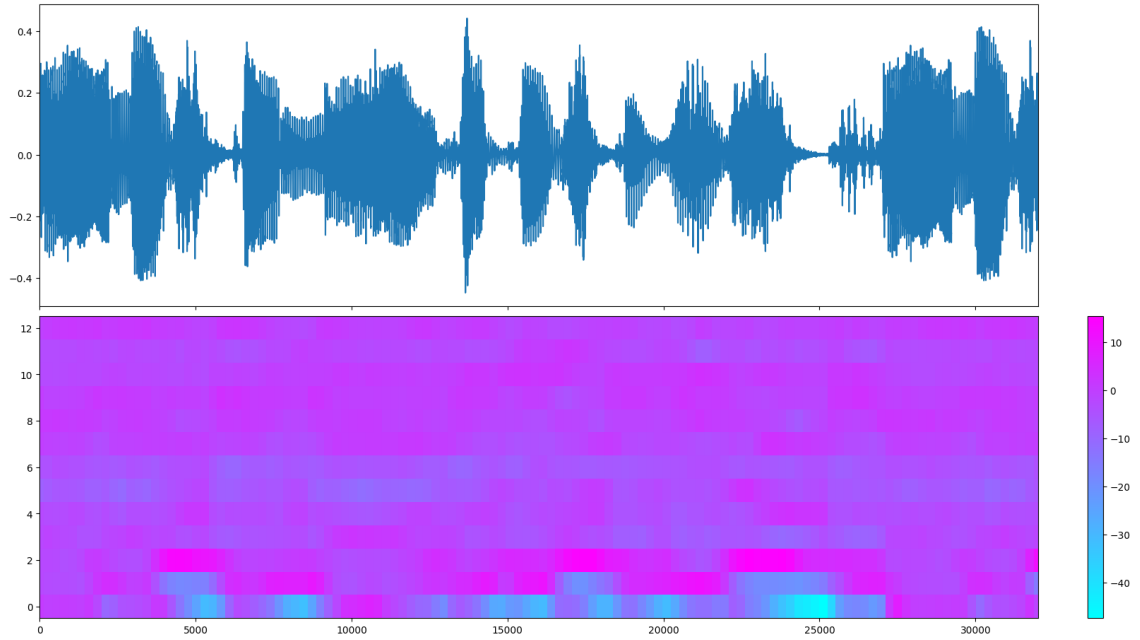


Kuva 2. Koneoppivan mallin toteutus FPGA-alustoilla. Aineiston käsittely ja mallin kehitys esimerkiksi Tensorflow/Keras tai PyTorch-ohjelmakirjastoilla toteutetaan emotietokoneella (PC), muunnos FPGA-alustaa varten toteutetaan emotietokoneella hls4ml-kirjastoa hyödyntäen, uuden aineiston tunnistus koulutetulla mallilla toteutetaan FPGA-alustalla.

3 PUHUJANTUNNISTUS OHJELMOITAVALLE PORTTIPIIRILLE SUUNNITELLULLA KONVOLUUTIONEUROVERKOLLA

3.1 Spektrogrammi ja mel-taajuuskertoimet

Äänidatan analysointi ja sen ominaisuuksien talteenotto suoraan sen signaalimuodosta on hankalaa tai lähes mahdotonta. Tästä syystä signaali usein muunnetaan spektrogrammiksi. Spektrogrammi on yleensä kaksiulotteinen kuva, joka kuvaa ääninäytteen eri taajuuksia ajan suhteen. Spektrogrammissa yksittäisen kuvapisteen väri kertoo kyseisen taajuuden voimakkuuden desibeleissä tai desibeleissä suhteessa korkeimpaan havaittuun äänipaineeseen (kuva 3).



Kuva 3. Esimerkki äänisignaalista sitä vastaavasta MFCC-spektrogrammista.

Spektrogrammeja käytetään puheentunnistuksessa kuvankaltaisena syötteenä yleensä konvoluutioneuroverkoille, joiden avulla tyypillisesti opitaan relevantteja piirteitä kuvadatasta. Spektrogrammeja voidaan luoda monella eri menetelmällä, kuten suodattimien avulla sekä käyttämällä Fourier- tai aallokemuunnosta. Näistä yleisin käytetty menetelmä on lyhyen aikavälin Fourier-muunnos (engl. Short-time Fourier Transform, STFT). STFT on

määritelmän mukaan

$$STFTx_n(h,k) = X(h,k) = \sum_{n=0}^{N-1} x_{n+h} w_n e^{-i2\pi \frac{kn}{N}}, \tag{1}$$

jossa x_{n+h} on ääninäytteestä näytteistetty lyhyt ajanjakso (pituudeltaan h), joka on ikkunoitu ikkunafunktiolla w_n. Tästä ikkunoidusta näytteestä otetaan diskreetti Fourier-muunnos $e^{-i2\pi \frac{kn}{N}}$ missä k on Fourier-sarjan indeksi. Sama toteutetaan kaikille näytteistetyille ikkunoille, jotka summataan yhteen spektriksi. Kosinisumma-ikkunointifunktiot ovat muotoa:

$$w_n = \sum_{k=0}^{K} (-1)^k a_k \cos\left(\frac{2\pi kn}{N}\right), \quad 0 \leq n \leq N. \tag{2}$$

STFT:ssä käytetään usein Hann- tai Hamming-ikkunointia aikaikkunoiden luomiseksi alkuperäisestä signaalista. Kummatkin ovat esimerkkejä yhtälön 2 mukaisista kosinisumma-ikkunointifunktioista, kun $K = 1$:

$$w_n = a_0 - \underbrace{(1 - a_0)}_{a_1} \cdot \cos\left(\frac{2\pi n}{N}\right), \quad 0 \leq n \leq N.$$

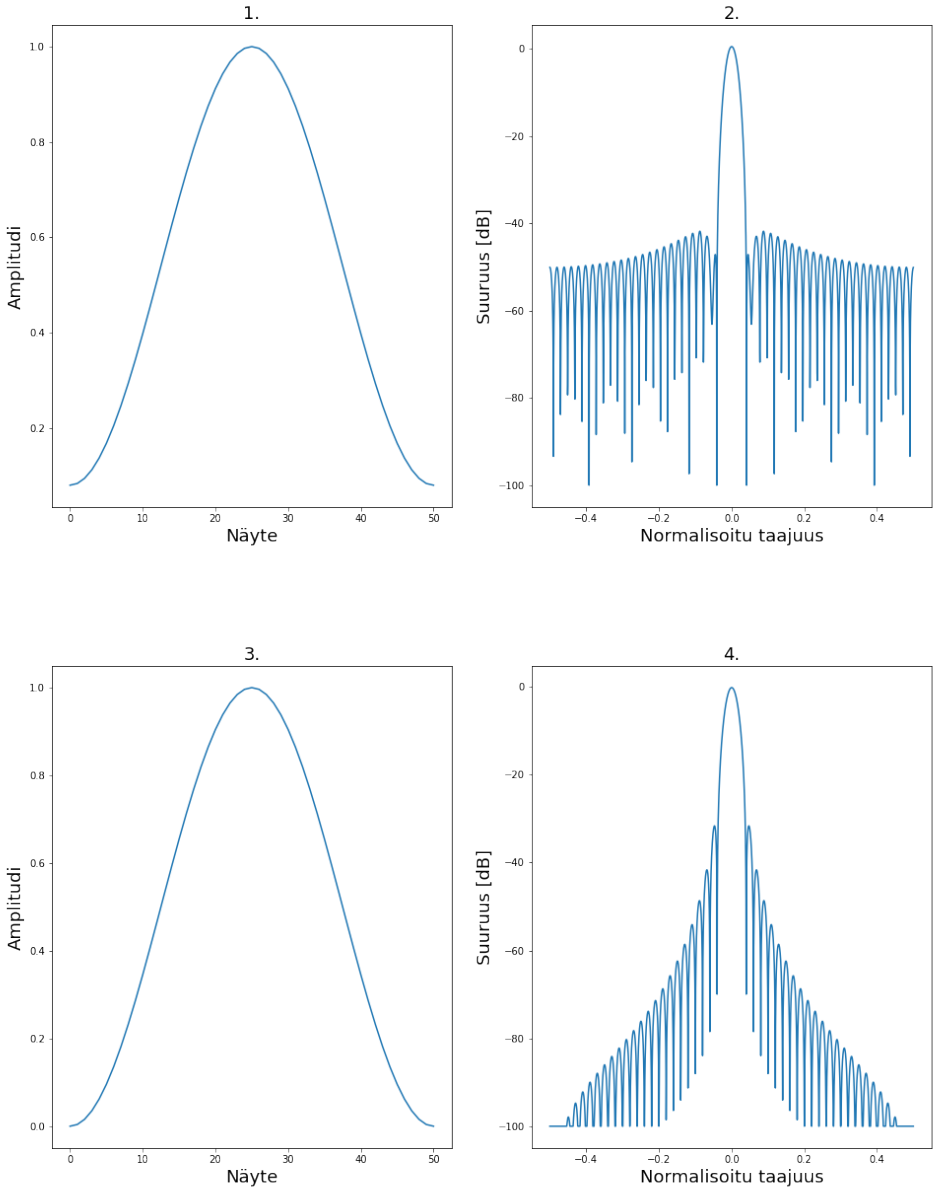
Kun asetetaan $a_0 = 0.5$, saadaan Hann-ikkunointifunktio ja vastaavasti jos asetetaan $a_0 = \frac{25}{46}$, saadaan Hamming-ikkunointifunktio. Kuten kuvasta 4 voidaan huomata, tämä pieni muuttujaero näissä kahdessa ikkunointifunktiossa vastaa noin 18 desibelin eroa per oktaavi. Ikkunointifunktion valinta riippuu käytettävän aineiston muodosta, esimerkiksi Hamming-ikkunointia käytetään data-analyysissa sen vähäisen spektraalisen vuodon vuoksi [17].

Kaavasta 1 saadusta spektristä otetaan usein logaritmi, jotta saadaan spektri vastaamaan desibelien logaritmista skaalaa. Tästä saatu spektri kuvataan lämpökarttana ajan suhteen, jossa värit kuvaavat desibeli-intensiteettiä. Logaritminen spektri saadaan kaavalla

$$S_L = 20 \log_{10}(X(h,k)).$$

Näitä logaritmisia spektrogrammeja voidaan käyttää sellaisenaan jo syötteenä puheentunnistusmalleille. Tosin tänä päivänä lähes jokainen puheentunnistusmalli hyödyntää MFCC-spektrogrammeja syötteenään [14]. Yleisen kepstri-algoritmin perusaskeleet on esitetty algoritmissa 1.

MFCC-algoritmi on eräs kepstri-algoritmi, jossa alkuperäinen spektri muunnetaan niin kutsuttuun log-mel-skaalaan ja tästä log-mel-spektristä otetaan talteen sen määrittävät



Kuva 4. Hamming-ikkunointifunktion havainnollistus (1.) sekä taajuusvaste (2.). Hann-ikkunointifunktion havainnollistus (3.) sekä taajuusvaste (4.).

Algoritmi 1 Kepstri-algoritmi

1: **funktio** KEPSTRIMUUNNOS(S) ▷ Äänisignaali S
2: $x_n := Hamming(S)$ ▷ Signaalin ikkunointi
3: $X_n := STFT(x_n)$ ▷ Lyhyen aikavälin Fourier-muunnos
4: $S_L = 20\log_{10}(X_n)$ ▷ Logaritminen spektri
5: $K := DFT(S_L)$ ▷ Toinen aika-taajuus muunnos (Fourier/kosini)
6: **palauta** K ▷ Kepstri-spektrogrammi
7: **lopeta funktio**

kertoimet spektrin amplitudista. MFCC-algoritmi on muunnettu kepstri-algoritmi sen toiminta on esitetty algoritmissa 2.

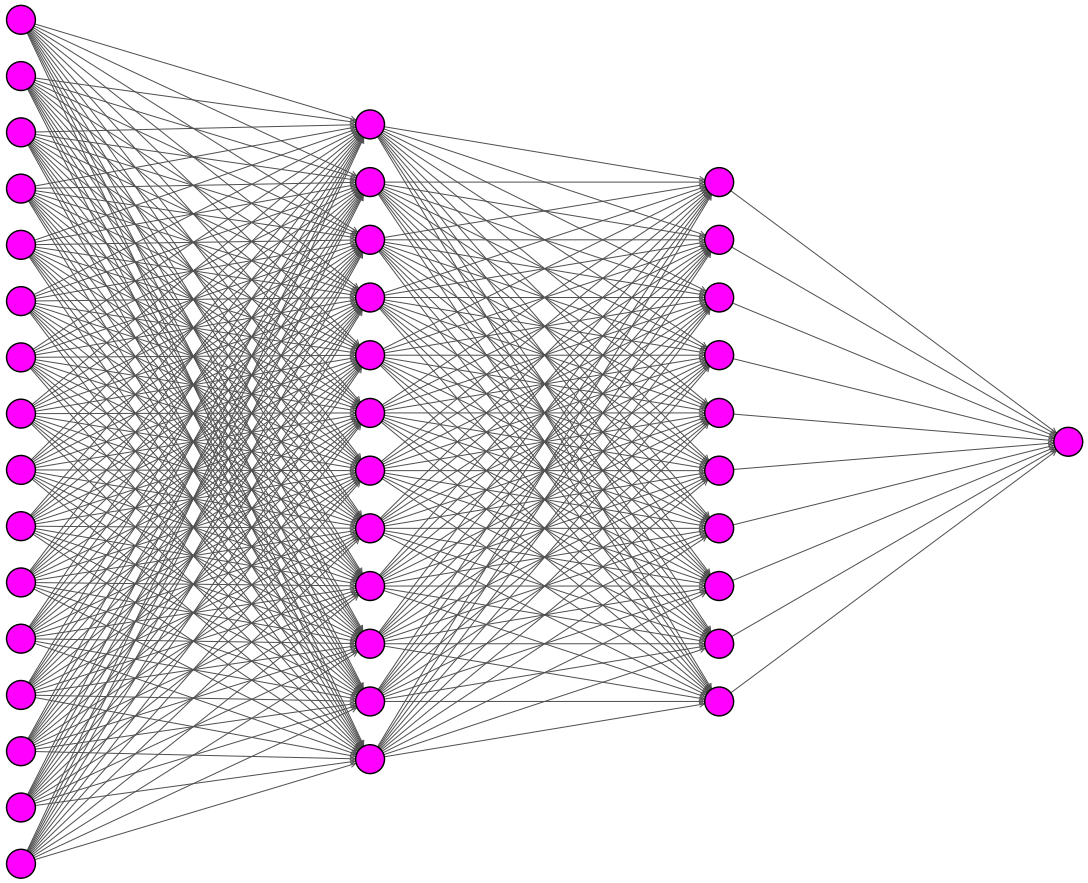
Algoritmi 2 MFCC-algoritmi

funktio MFCC(S) ▷ Signaali S
  $X_n := DFT(S_n)$ ▷ DFT ikkunoidusta signaalista
  $X_{mel} := 2595 \log_{10}\left(1 + \frac{f_X}{700}\right)$ ▷ Taajuuksien muuntaminen mel-skaalaan
  $L_{mel} := \log(|X_{mel} * F_{mel}|^2)$ ▷ Logaritmi mel-filtterin F_{mel} taajuualueilta
  $M := DCT(L_{mel})$ ▷ Diskreetti kosinimuunnos log-mel-spektristä
  $MFCC_n = M[A_n]$ ▷ MFCC-kertoimet amplitudeja saadusta spektristä
  palauta MFCC ▷ MFCC-kertoimet mel-filtterin mukaisilta taajuusalueilta
lopeta funktio

MFCC on nykypäivänä yleisin käytetty piirremenetelmä puheentunnistuksessa. Jos asiasta ei mainita artikkelissa tai raportissa, joka käsittelee puheen- tai puhujantunnistavia malleja, voi lähes olettaa että MFCC:tä on käytetty käsitellyissä malleissa.

3.2 Konvoluutioneurovekko

Keinotekoisilla neuroverkoilla (engl. Artificial Neural Networks, ANNs) tarkoitetaan graafeja, joiden solmut ovat kytketty kerroksittain toisiinsa (kuva 5). Näitä solmuja kutsutaan usein neuroneiksi ja niiden välisiä kytköksiä synapseiksi, sillä keinotekoiset neuroverkot ovat mallinnettu aivojen rakenteen mukaisiksi. Yksittäinen solmu saa arvonsa edellisen kerroksen solmujen summasta, joka usein syötetään aktivointifunktion läpi. Solmuilla on myös painokertoimet, joiden avulla voidaan painottaa haluttuja solmuja enemmän kuin toisia. Aktivointifunktiot ovat yksinkertaisimmillaan askelfunktioita, jolloin kerroksen solmujen summan ylittäessä askelfunktion kynnysarvon saa kyseinen solmu arvon 1 ja muuten arvon 0. Neuroverkon viimeinen kerros toimii useimmiten luokittimena. Yksinkertaisin luokitin on yksittäinen solmu. Solmun aktivoituessa on syöte arvioitu kuuluvan luokkaan 1 ja muuten luokkaan 0. Tällöin neuroni toimii lineaarisen luokittimen tavoin, jossa hypertaso jakaa parametriulottuvuuden kahtia. Useamman luokan tapauksessa usein käytetään aktivointifunktiona logistisia-funktioita (engl. sigmoid-function), jolloin yksittäiset solmut voivat saada mitä tahansa arvoja väliltä $[0, 1]$. Tällöin viimeisen kerroksen yksittäisen solmun arvot ovat voidaan tulkita todennäköisyytenä, että syöte kuuluu kyseisen solmun määrittelemään luokkaan [18].

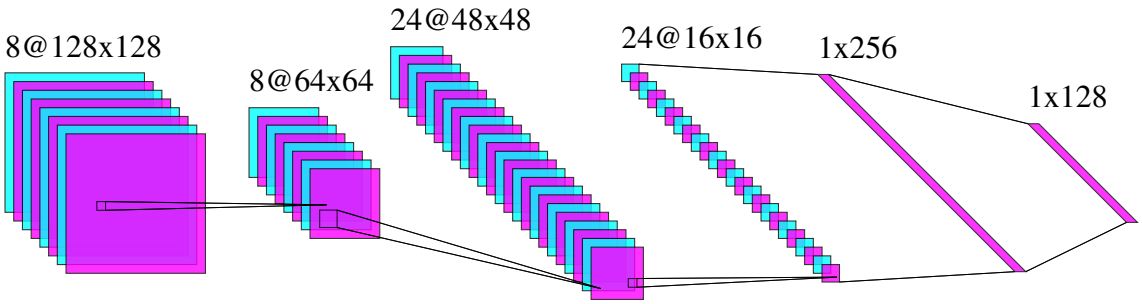


Kuva 5. Yksinkertainen neljäkerroksinen täysin kytketty neuroverkko.

Neuroverkon opetus voi tapahtua kolmella eri tavalla. Ohjatussa oppimisessa neuroverkolle annetaan jo luokiteltuja syötteitä ja näiden perusteella pyritään muuttamaan verkon painokertoimia ja aktivointifunktioiden kynnysarvoja niin, että lopullinen ulostulo vastaa syötteen luokkaa. Ohjaamattomassa oppimisessa neuroverkko saa syötteekseen luokittelemattomia syötteitä. Vahvistusoppimisessa neuroverkko oppii ympäristönsä antaman palautteen mukaan. Eteenpäin kytketyissä neuroverkoissa kaikissa näissä opetusmenetelmissä takana on vastavirta-algoritmi (engl. backpropagation), gradienttimenetelmä (engl. gradient decent) ja häviöfunktio (engl. loss-function). Häviöfunktion arvo kertoo neuroverkon tuottaman tuloksen numeerisen hyvyyden. Häviöfunktion minimoinnissa hyödynnetään gradienttimenetelmää. Neuroverkkojen tapauksessa minimointi joudutaan laskemaan jokaiselle solmulle, joka on laskennallisesti raskasta solmujen määrän kasvaessa. Tähän avuksi hyödynnetään vastavirta-algoritmia, joka nopeuttaa gradienttien laskentaa aloittamalla laskenta verkon viimeisestä kerroksesta [18].

Konvoluutioneuroverkko on keinotekoinen neuroverkko, joka sisältää vähintään yhden konvoluutiokerroksen (kuva 6). Konvoluutiokerros pystyy käsittelemään moniulotteisia syötteitä, minkä takia se on hyvin yleisesti käytetty esimerkiksi kuvantunnistuksessa, pu-

heentunnistuksessa ja hahmontunnistuksessa [19]. Yleensä kuitenkin konvoluutioneuroverkot sisältävät useampia konvoluutiokerroksia, joiden välillä on usein yhdistäviä kerroksia. Yhdistävien kerrosten tarkoitus on tiivistää konvoluutiokerroksien tuottama informaatio. Konvoluutiokerroksien yksi ominaispiirteistä on parametrien jakaminen (engl. Parameter sharing), joka tarkoittaa, että jokaisen parametriluokan käytetyt arvot ovat samat, toisin kuin täysin linkitetyssä verkkokerroksessa, jossa jokainen painokerroin ja kynnysarvo lasketaan erikseen. Toinen konvoluutiokerroksille ominainen piirre on harva linkitys, eli kerrokset ovat harvemmin linkitetty edellisiin ja seuraaviin kerroksiin neuroverkossa. Nämä kaksi ominaisuutta tekevät CNN:n kouluttamisesta nopeaa ja tehokasta.



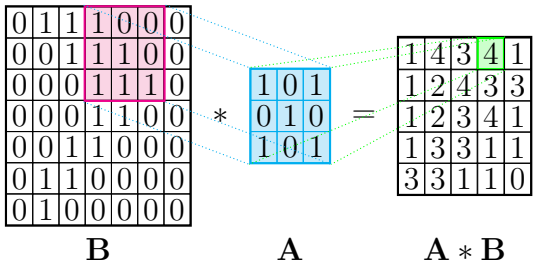
Kuva 6. Esimerkki LeNet-tyylisen [20] CNN:n arkkitehtuurista, joka sisältä kaksi konvoluutiokerrosta, joiden välissä maksimiyhdiste kerrokset sekä kaksi täysin kytkettyä kerrosta.

Konvoluutiokerrosten tehtävä neuroverkoissa on oppia syötteestä esimerkiksi luokitteluongelman ratkaisemisen kannalta hyödyllistä piirreinformaatiota. Kerrokset sisältävät ennalta määrätyn määrän konvoluutiokerneleitä. Yksittäinen konvoluutiokerneli tuottaa yhden konvoluutiopiirrekartan saamastaan syötteestä. Kaksiuloitteinen konvoluutio on kahden kaksiulotteisen matriisin välinen pistetulo ja on määritelty seuraavasti:

$$C(i,j) = \sum_{m=0}^{M_a-1} \sum_{n=0}^{N_a-1} A(m,n) \cdot B(i-m, j-n), \tag{3}$$

jossa A on $M_a \times N_a$ matriisi ja B on $M_b \times N_b$ matriisi. Indeksit i ja j ovat määritelty $0 \leq i < M_a + M_b - 1$ ja $0 \leq j < N_a + N_b - 1$ [21]. Konvoluutio tässä on pistetulo konvoluutiokernelin ja syötteen yksittäisen näytenaapuruston välillä (vrt. kaava 3, kun A = konvoluutiokerneli ja B = syötteen näytenaapurusto). Operaatio toteutetaan koko syötteen yli kuvapiste kerrallaan ja saatu konvoluutiopiirre on summa pistetulo-operaatioista, kuten kuva 7 havainnollistaa.

Konvoluutiokerroksen hyperparametrit ovat konvoluutiokernelien koko, kernelien määrä, konvoluution askelpituus sekä syötteen täydentäminen (engl. Padding). Saatu konvoluutiopiirrekartta syötetään aktivointifunktion läpi ennen syöttöä verkon seuraavaan kerrok-



Kuva 7. 2D-konvoluutio. Matriisi B kuvaa syötettä ja A kuvaa konvoluutiokerneliä.

seen, joka on yleensä yhdistekerros (engl. Pooling layer). Yhdistävien kerrosten yleisimpiä operaatioita on keskiarvoyhdiste (engl. Average pooling) tai maksimiyhdiste (engl. Maximum pooling).

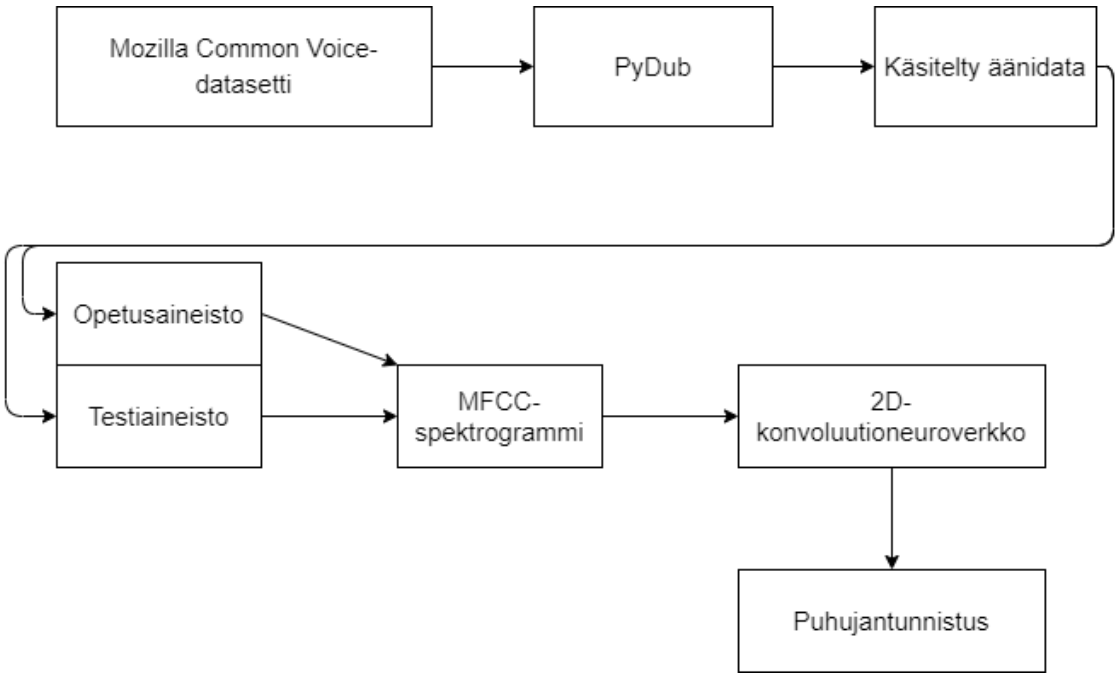
3.3 Aineiston esikäsittely

Työssä käytettiin Mozilla Common Voice -kirjaston suomenkielistä avoimen lähteen ääniaineistoa. Kyseinen aineisto ja käsittelyn vaiheet ovat tarkemmin esitelty osiossa 4.1.

Äänitiedostojen käsittely ja kääntäminen on toteutettu pääsääntöisesti käyttäen pydub-nimistä kirjastoa, joka hyödyntää FFmpeg-äänitiedostojen käsittelysovellusta [22]. FFmpeg on avoimen lähdekoodin projekti, joka sisältää laajan kattauksen kirjastoja videon, äänen ja muiden multimediatiedostojen käsittelyyn sekä suoratoistoon. Lopullisten äänisignaalien käsittely spektrogrammeiksi toteutettiin käyttäen Googlen Tensorflow:n valmiita signaalinkäsittelyfunktioita.

3.4 Puhujantunnistavan mallin kehitys

Puhujantunnistukseen luodut keinotekoiset neuroverkot ovat toteutettu ja opetettu hyödyntäen Tensorflow-kirjastoa, joka on avoimen lähdekoodin koneoppimiskirjasto Pythonille. Tensorflow hyödyntää Keras-nimistä avoimen lähdekoodin kirjastoa koneoppimismallien määrittelyyn. Työssä toteutetut mallit hyödyntävät Kerasin laajaa neuroverkkomoduulia, joka kattaa kaikki nykypäivänä laajalti käytössä olevat neuroverkkokerrokset konvoluutioista takaisinkytkettyihin verkkoihin. Lisäksi mallien karsimisessa hyödynnettiin Tensorflow:n erillistä mallin optimointikirjastoa. Puhujantunnistavan mallin kehityksen askeleet ovat esitetty kuvassa 8.



Kuva 8. Työssä toteutetun puhujantunnistavan mallin kehityksen vuokaavio.

3.5 Käännös ohjelmoitavaa porttipiiriä varten

Moderneilla neuroverkkokirjastoilla luodut neuroverkkomallit voidaan kääntää FPGA-korttitietokoneen ajettavaan muotoon käyttäen esimerkiksi hls4ml-kirjastoa [23]. Hls4ml on kirjasto, joka luo korkean tason synteesikielen laiteohjelmistototeutuksia avoimen lähteen koneoppimiskirjastojen avulla tuotetuista neuroverkkomalleista. Se hyödyntää Vivado HLS, Intel HLS ja Vitis HLS sovelluksia taustasovelluksinaan FPGA-alustoille toteutusten kehityksessä. Hls4ml tukee Tensorflow:lla [24], Kerasilla [25], PyTorch:lla [26], ONNX:llä [27] ja QKerasilla [28] luotujen neuroverkko mallien kääntämistä FPGA-alustoille.

Konvoluutioneuroverkkoa luodessa FPGA-alustoille täytyy kuitenkin ottaa huomioon laitteiston rajoitteet datan syötössä [29]. Tämä tarkoittaa sitä, että konvoluutioneuroverkon yksittäisten kerrosten parametrien määrä ei saa ylittää sisääntuloporttien lukumäärää alustalla tai muuten käännösprosessissa toteutusta ei kyetä optimoimaan vasteajan suhteen. Tämä ei toisaalta tarkoita, etteikö olisi mahdollista luoda leveämpää konvoluutioneuroverkkoa kyseisellä alustalla; käännösprosessin optimointi perustuu tällöin resurssien, kuten hakutaulujen ja kiikkujen käytön minimointiin.

Hls4ml luo FPGA-korttitietokoneelle tarvittavat ajuritiedostot neuroverkon laitteistokirjastoa varten. Laitteistokirjaston avulla luotua neuroverkkoa voidaan ajaa hyödyntäen

kohdealustan FPGA-moduulia. Pynq-korttitietokoneet ovat konfiguroitu niin, että niillä ajettavia sovelluksia voidaan nopeuttaa käyttäen samalla piirillä olevaa FPGA-moduulia normaalin suorittimen lisäksi. Neuroverkon laitteistokirjasto ladataan korttitietokoneella niin kutsuttuna ohjelmointikerroksena (engl. Overlay), jolloin sen määrittelemät prosessit ajetaan FPGA-kiihdytinmoduulia hyödyntäen.

4 KOKEET

4.1 Aineisto

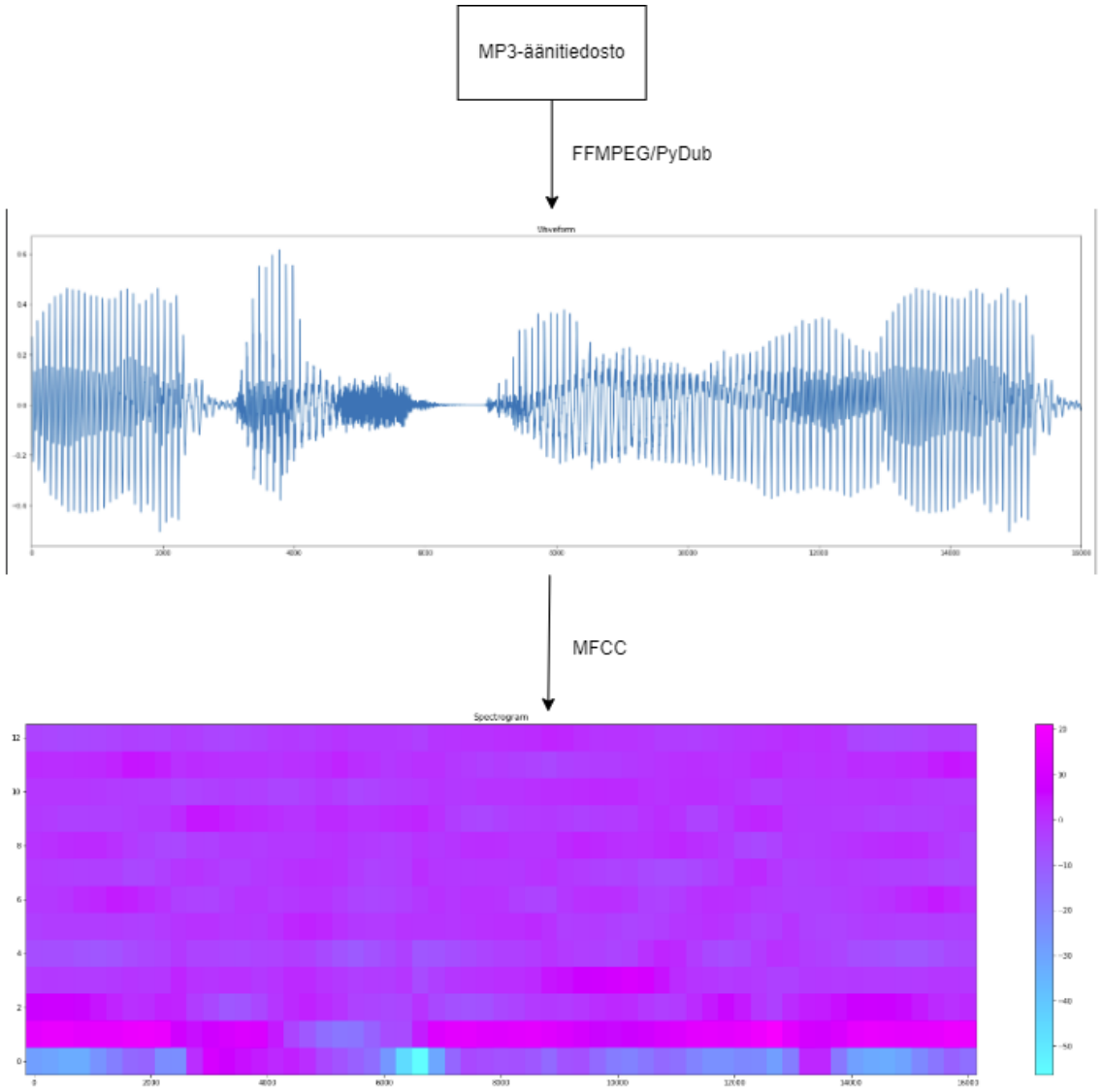
Työssä käytetään Mozilla Common Voice -projektin keräämää suomenkielistä avointa ääniaineistoa. Aineistossa on satunnaisesti luotujen lause- tai sanasyötteiden mukaan vapaaehtoisten henkilöiden lukemia anonyymeja ääninäytteitä MP3-muodossa. Suomenkielisessä aineistossa on noin 13 tuhatta ääninäytettä, joista noin 7 tuhatta on tarkistettu valideiksi projektissa mukana olevien vapaaehtoisten toimesta. Validit ääninäytteet muunnettiin alkuperäisestä MP3-muodosta WAV-muotoon äänen spektritiedon talteenottoa varten.

Koska aineisto on täysin vapaaehtoistoiminnalla luotu, ei ääninäytteiden laadusta ollut takeita. Näytteiden laatu ja pituus ovat hyvin vaihtelevia, mikä voisi aiheuttaa epätarkkuutta niin opetuksessa kuin testauksessa. Epätarkkuutta ja näytteiden epätasapainoisuutta pyrittiin minimoimaan siistimällä aineistoa mahdollisimman paljon ja vakioimalla näytteiden ominaisuuksia kuten näytteenottotaajuutta ja näytteen pituutta. Aineiston käsittelyyn käytetyt ohjelmistot ovat esitelty kappaleessa 3.3.

Aineisto siistittiin poistamalla hiljaiset kohdat näytteistä. Tällöin yksittäisen ääntä sisältävän näytteen keskimääräinen pituus oli alle 3 sekuntia. Koska ääninäytteiden pituudet olivat edelleen satunnaisia, haluttiin satunnaisuuden aiheuttamat mahdolliset virheelliset luokittelut minimoida vakioimalla ääninäytteiden pituudet kahteen sekuntiin joko katkaisemalla näytteet kahden sekunnin pituisiksi tai toistamalla näytteitä kunnes ne olivat halutun pituisia. Tämän lisäksi näytteiden näytteenottotaajuus vakioitiin 16 kHz:iin, jolloin yksittäinen ääninäyte sisältää 32 tuhatta data-alkiota. Aineisto muunnettiin tämän jälkeen ensin signaalimuotoon käyttäen FFmpeg-sovellusta, jotta siitä voitiin ottaa talteen spektrogrammi-informaatio.

Tämä signaaliaineisto muunnettiin MFCC-algoritmilla (algoritmi 2 sivulla 15) taajuuskertoimet sisältäväksi spektrogrammiksi, joita käytettiin mallissa kuvamaisina syötteinä. Yksittäisen data-alkion käsittelyprosessi on esitelty kuvassa 9. Aineisto mallia varten luotiin hyödyntäen Tensorflow:n valmiita aineiston luontiin tarkoitettuja funktioita. Aineisto jaettiin opetus- ja testausosioihin 80/20 (%) suhteella satunnaisesti. Jaossa varmistettiin kuitenkin, että opetusaineisto sisälsi ainakin yhden näytteen jokaiselta aineiston henkilöltä. Käsittelyn jälkeen data-aineisto sisälsi yhteensä 119 eri henkilön ääninäytteet eli aineisto sisälsi yhteensä 119 luokkaa. Aineistosta lopulta käytettiin vain 26 suurinta luok-

kaa lopullisessa sovelluksessa luokittelutehtävän pienentämiseksi laitteiston resurssivaatimusten takia.



Kuva 9. Data-alkion muunnosprosessin virtauskaavio.

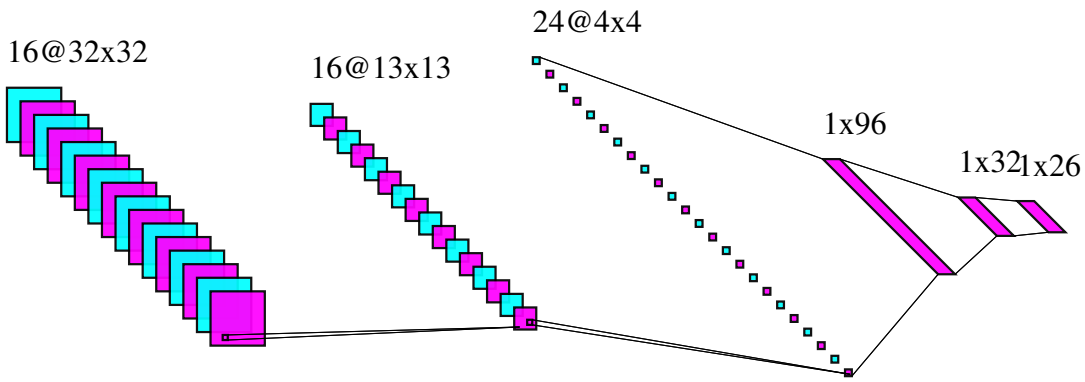
4.2 Mallit

Työssä luotujen konvoluutioneuroverkkojen haluttiin olevan riittävän pieniä, jotta kyseiset mallit voitaisiin teoriassa implementoida täysin rinnakkaisesti FPGA-alustalla. Tämä tarkoittaa sitä, että yksittäisten kerrosten parametrien määrän ei haluttu ylittävän Vivadon asettamaa 4096 parametrin rajaa. Suurempien kerrosten toteuttaminen alustalla on mahdollista. Tällöin resurssien uudelleenkäyttökerrointa (engl. Re-use Factor) täytyisi

kasvattaa vastaavasti yksittäisille kerroksille manuaalisesti.

Työssä toteutetut neuroverkot ovat sovellettuja versioita hls4ml:n esimerkin mukaisesta konvoluutioneuroverkosta. Toteutetut neuroverkot sisältävät kolme konvoluutiokerrosta, joiden välissä on eränormalisointikerros (engl. Batch Normalization Layer) sekä kaksiulotteinen maksimiyhdistekerros 2×2 yhdistekernelillä. Ensimmäiset kaksi konvoluutiokerrosta sisältävät mallista riippuen 16, 12 tai 8 kerneliä kooltaan 3×3 ja viimeinen konvoluutiokerros sisältää 24, 16 tai 12 kappaletta 3×3 kerneliä. Kaikkien kerrosten aktivointifunktiona käytetään ReLu-funktiota (engl. Rectified linear unit, ReLu) ja kernelien regularisoinnissa käytetään L1 eli lasso-regressiota 0.0001 kertoimella.

Konvoluutiokerrosten jälkeen neuroverkon syöte muunnetaan yksiuloitteiseksi eli muotoon $1 \times (Out_1 \cdot Out_2 \cdot Out_3)$, jossa Out_n on edellisen kerroksen ulostulon ulottuvuuden n koko. Tämä syötetään 30% pudotuskerroksen läpi (engl. Dropout-layer), joka satunnaisesti pudottaa 30% edellisen kerroksen ulostulon arvoista nollaan opetusvaiheessa, mikä vähentää mallin ylioppimisen mahdollisuutta. Pudotuskerroksen nollastapoikkeavat alkiot syötetään neuroverkon ainoaan varsinaiseen täysin kytkettyyn kerrokseen, joka sisältää joko 32, 24 tai 16 solmua. Tämän ulostulo vielä eränormalisoidaan ennen 26 solmun täysin kytkettyä luokittelukerrosta. Tämä kerros kertoo verkon tuottaman luokittelutuloksen todennäköisyytenä luokkien välillä. Top-1 ennustettu luokka on viimeisen kerroksen arvoltaan suurin solmu. Top-K ennustustarkkuutta voidaan myös tarkastella luokittelun yhteydessä, jolloin tutkitaan viimeisen kerroksen solmuista, että onko oikea luokka K:n suurimman solmun joukossa. Kuvassa 10 on havainnollistettu eräs työssä toteutettu neuroverkko.

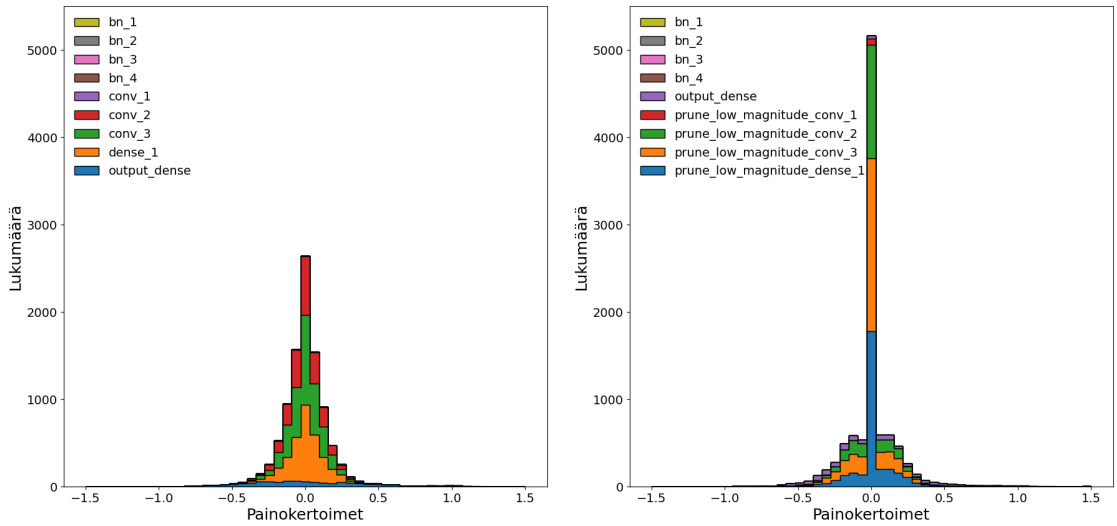


Kuva 10. Neuroverkon 1 (CNN_1/PCNN_1) havainnollistus. Ensimmäiset kolme kerrosta konvoluutiokerroksia ja viimeiset kaksi täysin kytkettyjä kerroksia.

Neuroverkkojen arkkitehtuurit ovat esitetty tarkemmin vielä taulukoissa A1.9, A1.10 ja A1.11. Jokaisesta erikokoisesta neuroverkosta luotiin karsittu malli (engl. Pruned model)

käyttäen Tensorflow:n mallin optimointi kirjastoa. Karsinnan ideana on karsia opetusvaiheessa pohjamallista sellaiset kertoimet, jotka eivät vaikuta lopulliseen luokittelutulokseen merkittävästi, asettamalla niiden arvot nollaan. Tällöin kyseisen mallin laskenta helpottuu ja näin resurssivaatimukset laskevat mallin tarkkuuden säilyessä. Tässä tavoitteelliseksi harvuudeksi karsituille malleille asetettiin 50% eli karsimisen tavoitteena oli karsia puolet kertoimista nollaan. Karsitut versiot luotiin käyttäen polynomin hajottamista (engl. Polynomial Decay) karsintanopeutena. Taulukoissa esitellyt arkkitehtuurit ovat samanlaiset myös karsituilla versioilla.

Työssä käytetty karsintaprosessi vastaa hls4ml:n esimerkin mukaista karsintaprosessia [29]. Karsinta aloitetaan mallin kymmenennellä opetuskierroksella (engl. Epoch). Kuvasta 11 huomataan, karsintaprosessin vaikutus mallien painokerrointen jakaumassa.



(a) Pohjamalli (b) Karsittu malli

Kuva 11. Pohjamallin vs. karsitun mallin painokertoimet.

4.3 Parametrit ja suorituskyvyn arviointi

Työssä mitattiin kolmen erikokoisen pohjamallin sekä niiden karsittujen mallien luokittelutarkkuutta verrattuna niiden hls4ml-versioihin. Lisäksi tutkittiin hls4ml-kvantisoitujen mallien estimoituja resurssivaatimuksia hls4ml:n ja Vivadon tuottamien synteesiraporttien pohjalta. Mitatut laskentaresurssit ovat lohkomuisti (engl. block random access memory, BRAM), digitaalinen signaaliprosessorit (engl. digital signal processor, DSP), kiikut

(engl. flip-flop, FF) sekä hakutaulut (engl. lookup table, LUT).

Kaikkien mallien laskentatarkkuus asetettiin tarkkuuteen ’$\mathrm{ap\_fixed} < 16, 6 >$’ hls4ml:n avulla. Edeltävä merkintä kertoo laskentatarkkuuden käyttävän 16 bittisiä numeroita, joista 6 on varattu lukujen kokonaisosien (desimaalipilkun vasemmanpuolisien) käsittelyyn ja loput 10 desimaaliosien käsittelyyn. Lisäksi mallien sekä yksittäisten kerrosten resurssien uudelleenkäyttökerroin vakioitiin 64:ään resurssiestimaattien vertailukelpoisuutta varten. Valittu uudelleenkäyttökerroin valittiin hls4ml:n Pynq-Z2:lla toteutetun neuroverkkoesimerkin implementaation pohjalta, jossa käytettiin samaa uudelleenkäyttökerrointa täysinkytketyn neuroverkon toteutuksessa.

4.4 Tulokset

Työssä ei onnistuttu toteuttamaan alkuperäisen suunnitelman mukaisesti konvoluutioneuroverkkoa Pynq-Z2 alustalle. Toteutetut neuroverkot vaativat FPGA-alustalta enemmän resursseja kuin Pynq-Z2:n SoC sisältää. Työssä kokeiltiin hls4ml:n resurssioptimointia neuroverkon FPGA-implementaatiossa mutta tämä osoittautui epäkäytännölliseksi, sillä käännetyt neuroverkot eivät toimineet FPGA-alustalla odotetusti. Koska kyseisiä neuroverkkoja ei saatu toimimaan FPGA-alustalla, tässä raportissa ei ole vertailukelpoisia tuloksia mallien nopeudesta suoritintoteutuksen ja FPGA-toteutuksen välillä.

Sen sijaan työssä verrattiin kuuden erilaisen konvoluutioneuroverkon estimoituja resurssivaatimuksia ja kyseisten neuroverkkojen luokittelutarkkuuksia alkuperäisten mallien sekä niiden kvantisoitujen vastineiden välillä. Täyden laskentatarkkuuden ja hls4ml kvantisoitujen mallien tarkkuudet ovat esitetty taulukossa 1. Kyseisestä taulukosta nähdään, että vaikka mallien koko pienenee vasemmalta oikealle, Keras-mallien tarkkuudet pysyvät kaikki yli 90%. Näin systemaattinen korkea tarkkuus pienelläkin neuroverkolla kertoo MFCC-spektrogrammien kuvaavien piirteiden korkeasta suorituskyvystä täydellä laskentatarkkuudella.

Taulukko 1. Mallien tarkkuuksia ennen hls4ml-kvantisointia sekä sen jälkeen. CNN-mallit ovat pohjamalleja ja PCNN-mallit ovat pohjamalleista karsittuja versioita.

	CNN_1	PCNN_1	CNN_2	PCNN_2	CNN_3	PCNN_3
Keras Top-1 [%]	96.5	95.4	95.1	94.3	95.1	92.9
Keras Top-3 [%]	100	98.8	99.5	99.3	99.3	99.3
hls4ml Top-1 [%]	79.1	72.7	64.8	82.5	68.1	68.8
hls4ml Top-3 [%]	90.7	87.0	77.4	91.7	85.5	85.7

Resurssiestimaatit ovat Vivadon tuottamia synteesi-estimaatteja kyseiselle mallille ja ovat tiedettävästi usein liian suuria oikeaan implementaatioon verrattuna. Varsinaista implementaatiota ei voitu työssä toteutetuille verkoille luoda liian suurten resurssiestimaattien takia. Vaikka estimaatit ovat usein yliampuvia, antavat ne kuitenkin suuruusluokan eri resurssien käytölle ja työssä käytetyn alustan resurssit eivät ole riittävät järkevän implementaation luomiseen. Kaikkien mallien resurssien käyttöaste on esitetty prosentteina suhteessa Pynq-Z2:n resursseihin (taulukko 2).

Taulukko 2. Pynq-Z2:n XC7Z020-1CLG400C-mikropiirin laskentaresurssit. Resurssien lyhenteiden merkitys esitetty tekstissä osiossa 4.3.

Resurssi	BRAM	DSP	FF	LUT
Määrä	280	220	106.4K	53.2K

Taulukoissa 3 ja 4 on esitelty työssä toteutetuista neuroverkoista keskikokoisen mallin resurssiestimaatteja. Kuten kyseisistä taulukoista sekä liitteen taulukoista A1.5, A1.6, A1.7 ja A1.8 voidaan huomata, konvoluutioneuroverkot vaativat paljon resursseja FPGA-alustalta, varsinkin hakutauluja.

Taulukko 3. 2. pohjamallin resurssiestimaatit

Malli: CNN_2	BRAM	DSP	FF	LUT
Käytetty	99	114	138.5K	181.5K
Käyttöaste [%]	35	51	130	341

Taulukko 4. 2. karsitun mallin resurssiestimaatit

Malli: PCNN_2	BRAM	DSP	FF	LUT
Käytetty	99	101	99.9K	117.8K
Käyttöaste [%]	35	45	93	221

5 POHDINTA

5.1 Tulosten vertailu

Tuloksista voidaan huomata, että hls4ml:n kvantisointi verottaa oletetusti kullakin neuroverkolla luokittelutarkkuutta. Kuitenkin kunkin kvantisoidun neuroverkon Top-1 tarkkuus pysyy varteenotettavissa rajoissa käytännönsovellusta varten. Top-3 tarkkuudet on taulukoitu mukaan havainnollistamaan, että käytännönsovelluksessa voitaisiin halutessa käyttää Top-K tarkkuutta luokittelun hyväksynnässä. Karsittu versio CNN_2:sta (taulukossa PCNN_2) erottuu hls4ml-tarkkuuksien suhteen muista malleista, sillä kyseinen verkko ei syystä tai toisesta kärsi yhtä pahasti mallin kvantisoinnista verrattuna muihin malleihin. Tässä huomataan pieni satunnaisuus hls4ml:n kvantisoinnissa siinä suhteessa, ettei ole täysin mahdollista ennustaa kvantisoidun mallin suorituskykyä ennen testaamista. PCNN_2 on täten vastoin odotuksia kaikista malleista stabiilein pienten laskentatarkkuuden muutosten suhteen.

Yksikään työssä toteutetuista neuroverkkomalleista ei sellaisenaan vastannut estimaattien suhteen resurssivaatimuksiltaan Pynq-Z2:n resursseja. PCNN_2 on malleista sellainen, joka olisi hieman resurssirikkaammalle FPGA-alustalle kuten PYNQ-ZU:lle järkevä yrittää implementoida. Kyseinen malli on myös kvantisoituna malleista mielenkiintoisin luokittelutarkkuuden suhteen kuten aiemmin jo aihetta käsiteltiin kappaleessa 4.4. PCNN_3 (taulukko A1.8) alkaa olla toisaalta jo lähellä resurssivaatimuksiltaan mahdollisesti implementoitavaa mallia. Vaikka tuotetut resurssiarviot sellaisenaan eivät ole suoraan vertailukelpoisia, verrattuna Aarrestad *et al*. [29] saamiin tuloksiin hls4ml:n resurssien käytöstä samankaltaisilla konvoluutioneuroverkoilla, tässä työssä saadut resurssiestimaattitulokset vahvistavat kyseisen raportin tuloksia. Tuloksista voidaan huomata lineaarisuutta mallin koon ja resurssien käytön kuten kiikkujen, hakutaulujen sekä lohkomuistin suhteen, kuten Aarrestad *et al*. raportissaan myös ovat todenneet.

5.2 Jatkotutkimukset

Tämän raportin tulokset vahvistavat, että mallin optimointi karsimalla ennen hls4ml-mallin luontia on suurin vaikuttava tekijä resurssivaatimusten vähentämisessä. Työkalut kuten QKeras ja AutoQKeras ovat tänä päivänä tehokkaita ja varteenotettavia aputyökaluja mallien lisäoptimoinnissa, kuten myös Aarrestad *et al*. raportissaan toteavat. Neuroverkkojen arkkitehtuuria voisi mahdollisesti muuttaa toisenlaiseksi, joka saattaisi

vähentää resurssivaatimuksia. Toisaalta usein helpoin ratkaisu resurssien loppuessa on vaihtaa suurempaan FPGA-alustaan, kuten Xilinx UltraScale+ -sarjan FPGA-moduulin sisältävään alustaan.

Käytännöllisenä sovelluksena myös neuroverkko tulisi kouluttaa käyttötarkoituksen mukaisella aineistolla, mikä tarkottaisi, että kyseinen aineisto tulisi kerätä tai olla jo olemassa, jotta toteutettua sovellusta voitaisiin hyödyntää. Työssä käytetty aineisto oli laadultaan vaihtelevaa ja hyvin epätasapainoinen luokkien välillä, mikä vaikutti työn tulosten tarkkuuteen.

FPGA-alustaa hyödyntävä käytännön käyttökohde tulisi olla riittävän staattinen luokittelutehtävä, jottei neuroverkkoa tarvitsisi uudelleenkouluttaa sekä implementoida laitteistolle sopivaan muotoon. Syntetisoinnin ja käännösprosessin raskaus tekee tästä FPGA-alustaa hyödyntävästä neuroverkkosovelluksesta epäkäytännöllisen esimerkiksi biometrisen tunnistuksen sovelluksena, sillä uusien henkilöiden lisääminen mallin tunnistettavien joukkoon on hidasta ja työlästä. Tämä on yleisestikin automatisoidun puhujantunnistuksen suurin ongelma, sillä jokaisen uuden henkilön täytyy syöttää mallille näyte opetusta varten, ennen kuin mallin on mahdollista kyetä tunnistamaan kyseisen henkilön. Eräs ratkaisu tähän olisi luoda malli, joka oppii ääniaineistosta yleisesti merkitykselliset piirteet riittävän laajasta aineistosta. Tätä mallia voitaisiin hyödyntää puhujantunnistuksessa samankaltaisuusmittaan perustuvassa luokittimessa. Kohdesovellusta voitaisiin toisaalta muuttaa biometrisen varmistuksen mukaiseksi, jolloin tehtävänä olisi selvittää onko henkilö se kuka hän väittää olevansa. Tällöin luokittelu muuttuisi binääriseksi kun selvitetään onko henkilön väite tosi ääninäytteen avulla.

6 JOHTOPÄÄTÖKSET

Tässä työssä puhujantunnistus pyrittiin toteutettamaan luomalla keinotekoinen konvoluutioneuroverkko käyttäen Googlen Tensorflow kirjastoa. Työssä käytettiin Mozilla Common Voice -projektin luomaa suomenkielistä puheaineistoa neuroverkon opetuksessa sekä testaamisessa. Aineisto esikäsiteltiin käyttäen pydub-nimistä kirjastoa Python-ohjelmointikielellä. Esikäsitelty aineisto muunnettiin neuroverkolle sopivaan spektrogrammi-muotoon käyttäen MFCC-menetelmää. Neuroverkko pyrittiin muuntamaan Pynq-Z2 SoC:n ajettavaan muotoon käyttäen hls4ml-kirjastoa mutta tässä epäonnistuttiin alustan resurssivajeen takia. Sen sijaan tutkittiin erikokoisten neuroverkkojen resurssivaatimuksia FPGA-toteutuksena ja verrattiin näiden luokittelusuorituskykyä hls4ml:n mallisimulaatiota hyödyntäen.

Tulosten perusteella työssä luodut neuroverkot ovat tarkkuudeltaan varteenotettavia käytännön sovelluksiin. Tämä tulos kertoo enimmäkseen MFCC-piirteiden vahvuudesta puhujantunnistuksessa konvoluutioneuroverkoilla, mikä oli suurin syy kyseisen menetelmän valintaan työn toteutuksessa. Konvoluutioneuroverkot yleisesti vaativat paljon laskentaresursseja ja FPGA-alustoilla nämä resurssit ovat rajalliset. Tästä syystä työssä toteutetut neuroverkot pyrittiin pitämään mahdollisimman pieninä kuitenkin pyrkien säilyttämään korkean luokittelutarkkuuden. Toisaalta kyseisten neuroverkkojen toteutus FPGA-alustoilla ja varsinkaan työn pääsääntöisellä kohdealustalla Pynq-Z2:lla on hankalaa kyseisen alustan resurssien vähyyden takia. Mikään työssä toteutetuista malleista ei vastannut Pynq-Z2:n resursseja Vivadon synteesiraporttien pohjalta.

Vaikka työssä toteutetut pohjamallit ovat varsin pieniä neuroverkoiksi nykypäivän standardeilla, mallit kuluttavat resursseja turhan paljon ollakseen varteenotettavia kandidaatteja implementoitavaksi FPGA-alustoille. Tästä syystä usein pohjamalleja ei edes yritetäkään implementoida sellaisenaan vaan mallien esioptimointi yksi tärkeimmistä vaiheista neuroverkkojen toteutuksessa FPGA-alustoilla. Työssä toteutettujen neuroverkkojen resurssivaatimusestimaatit vastaavat jo ennestään tuotettuja tuloksia.

Suurin hyöty FPGA-alustojen käytöstä neuroverkkojen toteutuksessa tulee niiden perinteisiä suoritintoteutuksia vastaavan laskentatehon kautta käyttämällä murto-osan suoritinten vaatimasta energiasta puhumattakaan näytönohjainten vaatimasta sähkötehosta. Lisäksi FPGA-alustojen kompakti koko mahdollistaa niiden sulauttamisen jo olemassa oleviin järjestelmiin kuten teollisuuslaitteisiin.

LÄHTEET

[1] Yishuang Ning, Sheng He, Zhiyong Wu, Chunxiao Xing, and Liang-Jie Zhang. A review of deep learning based speech synthesis. *Applied Sciences*, 9:4050, 09 2019.

[2] H. Ahrens. Voice.Ai, 2023. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: `https://voice.ai`.

[3] Sumit Sarin, Antriksh Mittal, Anirudh Chugh, and Smriti Srivastava. Cnn-based multimodal touchless biometric recognition system using gait and speech. *Journal of intelligent & fuzzy systems*, 42(2):981–990, 2022.

[4] Lin Jiang, Ping Tan, Junfeng Yang, Xingbao Liu, and Chao Wang. Speech emotion recognition using emotion perception spectral feature. *Concurrency and computation*, 33(11), 2021.

[5] AMD Xilinx. Pynq-z2, 2018. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: `https://www.xilinx.com/support/university/xup-boards/XUPPYNQ-Z2.html#overview`.

[6] Zhihong Zeng, M. Pantic, G.I. Roisman, and T.S. Huang. A survey of affect recognition methods: Audio, visual, and spontaneous expressions. *Institute of Electrical and Electronics Engineers transactions on pattern analysis and machine intelligence*, 31(1):39–58, 2009.

[7] Tomi Kinnunen and Haizhou Li. An overview of text-independent speaker recognition: From features to supervectors. *Speech Communication*, 52(1):12–40, January 2010.

[8] Dr. R. K. Srivastava and Digesh Pandey. Speech recognition using HMM and Soft Computing. *Materials Today: Proceedings*, 51:1878–1883, January 2022.

[9] Rania Chakroun and Mondher Frikha. Robust Text-independent Speaker recognition with Short Utterances using Gaussian Mixture Models. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pages 2204–2209, June 2020. ISSN: 2376-6506.

[10] Craig S. Greenberg, Lisa P. Mason, Seyed Omid Sadjadi, and Douglas A. Reynolds. Two decades of speaker recognition evaluation at the national institute of standards and technology. *Computer Speech & Language*, 60:101032, March 2020.

[11] Yongfeng Li, Shuaishuai Chang, and QingE Wu. A short utterance speaker recognition method with improved cepstrum–CNN. *SN Applied Sciences*, 4(12):1–11, 2022. Place: Cham Publisher: Springer International Publishing.

[12] Sukmawati Nur Endah, Satriyo Adhy, and Sutikno Sutikno. Comparison of feature extraction mel frequency cepstral coefficients and linear predictive coding in automatic speech recognition for indonesian. *Telkomnika*, 15(1):292–, 2017.

[13] S Lokesh, Priyan Malarvizhi Kumar, M Ramya Devi, P Parthasarathy, and C Gokulnath. An automatic tamil speech recognition system by using bidirectional recurrent neural network with self-organizing map. *Neural computing & applications*, 31(5):1521–1531, 2019.

[14] Zhaodong Lin, Changan Di, and Xiong Chen. Bionic optimization of MFCC features based on speaker fast recognition. *Applied Acoustics*, 173:107682, February 2021.

[15] Ruiqi Chen, Tianyu Wu, Yuchen Zheng, and Ming Ling. MLoF: Machine Learning Accelerators for the Low-Cost FPGA Platforms. *Applied sciences*, 12(1):89–, 2021. Place: Basel Publisher: MDPI AG.

[16] Maya Gokhale and Lesley Shannon. FPGA Computing. *Institute of Electrical and Electronics Engineers Micro*, 41(4):6–7, 2021. Place: United States Publisher: IEEE.

[17] Md. Jahangir Alam, Patrick Kenny, and Douglas O'Shaughnessy. Low-variance multitaper mel-frequency cepstral coefficient features for speech and speaker recognition systems. *Cognitive computation*, 5(4):533–544, 2013.

[18] Anders Krogh. What are artificial neural networks? *Nature Biotechnology*, 26(2):195–197, February 2008.

[19] Bhatt *et al.* . CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope. *Electronics*, 10(20):2470, January 2021. Number: 20 Publisher: Multidisciplinary Digital Publishing Institute.

[20] Siham Tabik, Daniel Peralta, Andrés Herrera-Poyatos, and Francisco Herrera. A snapshot of image pre-processing for convolutional neural networks: case study of mnist. *International journal of computational intelligence systems*, 10(1):555–, 2017.

[21] MathWorks. MATLAB documentation, 2023. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: `https://se.mathworks.com/help`.

[22] Bobby Bingham Fabrice Bellard. FFmpeg, 2000. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: https://ffmpeg.org/documentation.html.

[23] FastML Team. fastmachinelearning/hls4ml, 2021. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: https://github.com/fastmachinelearning/hls4ml.

[24] Martín Abadi *et al.* . TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: https://tensorflow.org/.

[25] Francois Chollet *et al.* . Keras, 2015. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: https://github.com/fchollet/keras.

[26] Paszke *et al.* . Pytorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[27] ONNX community. Open neural network exchange, 2017. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: https://github.com/onnx.

[28] C. Coelho. QKeras, 2019. [Verkkoaineisto]. [Viitattu 19.5.2023]. Saatavissa: https://github.com/google/qkeras.

[29] Thea Aarrestad et al. Fast convolutional neural networks on FPGAs with hls4ml. *Machine Learning: Science and Technology*, 2(4):045015, 2021.

LIITE 1.

Taulukko A1.5. 1. pohjamallin resurssiestimaatit

Malli: CNN_1	BRAM	DSP	FF	LUT
Käytetty	131	192	228.6K	317.6K
Käyttöaste [%]	46	87	214	597

Taulukko A1.6. 1. karsitun mallin resurssiestimaatit

Malli: PCNN_1	BRAM	DSP	FF	LUT
Käytetty	131	168	157.5K	190.2K
Käyttöaste [%]	46	76	148	357

Taulukko A1.7. 3. pohjamallin resurssiestimaatit

Malli: CNN_3	BRAM	DSP	FF	LUT
Käytetty	67	62	82.7K	104.1K
Käyttöaste [%]	23	28	77	195

Taulukko A1.8. 3. karsitun mallin resurssiestimaatit

Malli: PCNN_3	BRAM	DSP	FF	LUT
Käytetty	67	58	63.4K	74.7K
Käyttöaste [%]	23	26	59	140

LIITE 1.

Taulukko A1.9. Mallien CNN_1 ja PCNN_1 arkkitehtuuri

Kerros	Ulostulon muoto	Parametrien määrä
Syöte	(32, 32, 1)	-
2D-konvoluutio	(30, 30, 16)	160
Eränormalisointi	(30, 30, 16)	64
Maksimiyhdiste	(15, 15, 16)	-
2D-konvoluutio	(13, 13, 16)	2 320
Eränormalisointi	(13, 13, 16)	64
Maksimiyhdiste	(6, 6, 16)	-
2D-konvoluutio	(4, 4, 24)	3 480
Eränormalisointi	(4, 4, 24)	96
Maksimiyhdiste	(2, 2, 24)	-
Tasaus	(96)	-
Pudotus (30%)	(32)	-
Täysin kytketty	(32)	3 104
Eränormalisointi	(32)	128
Luokittelu	(26)	858

Ei-koulutettavien parametrien määrä	176
Parametrien määrä yhteensä	10 274

Taulukko A1.10. Mallien CNN_2 ja PCNN_2 arkkitehtuuri

Kerros	Ulostulon muoto	Parametrien määrä
Syöte	(32, 32, 1)	-
2D-konvoluutio	(30, 30, 12)	120
Eränormalisointi	(30, 30, 12)	48
Maksimiyhdiste	(15, 15, 12)	-
2D-konvoluutio	(13, 13, 12)	1 308
Eränormalisointi	(13, 13, 12)	48
Maksimiyhdiste	(6, 6, 12)	-
2D-konvoluutio	(4, 4, 16)	1 744
Eränormalisointi	(4, 4, 16)	64
Maksimiyhdiste	(2, 2, 16)	-
Tasaus	(64)	-
Pudotus (30%)	(24)	-
Täysin kytketty	(24)	1 560
Eränormalisointi	(24)	96
Luokittelu	(26)	650

Ei-koulutettavien parametrien määrä	128
Parametrien määrä yhteensä	5 638

LIITE 1.

Taulukko A1.11. Mallien CNN_3 ja PCNN_3 arkkitehtuuri

Kerros	Ulostulon muoto	Parametrien määrä
Syöte	(32, 32, 1)	-
2D-konvoluutio	(30, 30, 8)	80
Eränormalisointi	(30, 30, 8)	32
Maksimiyhdiste	(15, 15, 8)	-
2D-konvoluutio	(13, 13, 8)	584
Eränormalisointi	(13, 13, 8)	32
Maksimiyhdiste	(6, 6, 8)	-
2D-konvoluutio	(4, 4, 12)	876
Eränormalisointi	(4, 4, 12)	48
Maksimiyhdiste	(2, 2, 12)	-
Tasaus	(48)	-
Pudotus (30%)	(16)	-
Täysin kytketty	(16)	784
Eränormalisointi	(16)	64
Luokittelu	(26)	442

Ei-koulutettavien parametrien määrä	88
Parametrien määrä yhteensä	2 942