Oamk Journal

Oulun ammattikorkeakoulun julkaisuja

METATIEDOT

Tyyppi: Blogi
Julkaisija: Oulun ammattikorkeakoulu
Julkaisunumero: 70/2024
Julkaisuvuosi: 2024
Tekijätiedot: Jyrkkä Kari, Grönroos Thomas, Manninen Jere, Partanen Riku

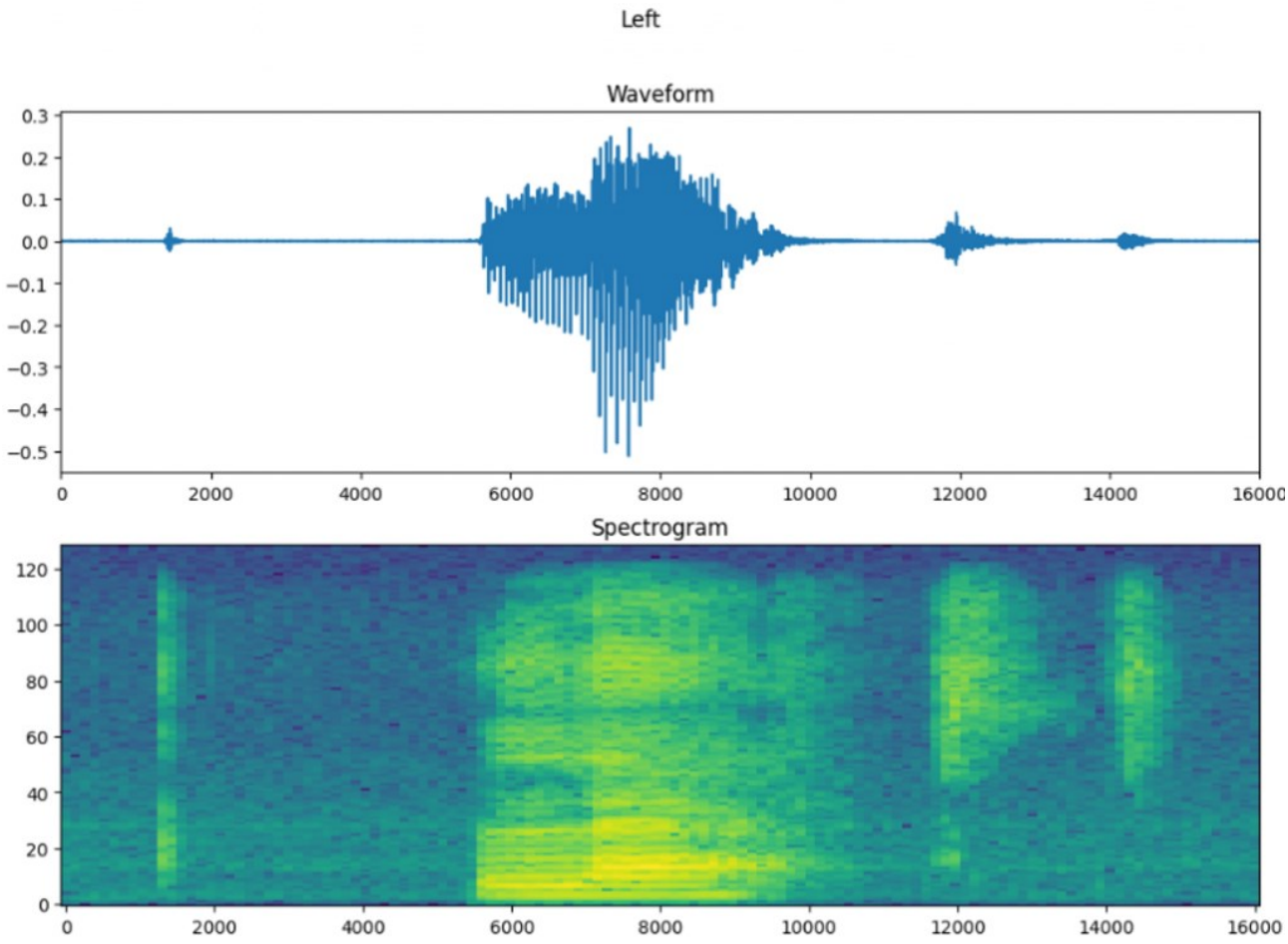
Kieli: suomi
Pysyvä osoite: http://urn.fi/urn:nbn:fi-fe2024060645997
Tiivistelmä: Oulun ammattikorkeakoulun informaatiotekniikan opiskelijaryhmä toteutti projektiopintoina äänikomentoja tunnistavaa järjestelmää tekoälyn avulla. Projektiryhmän tavoitteena oli opetella äänikomentojen tunnistamiseen soveltuvan konvoluutioneuroverkon toimintaperiaate ja ohjelmoida tarvittavat algoritmin osat Python-ohjelmointikielellä. Lopullinen tavoite oli toteuttaa äänikomentojen tunnistus Nordic Semiconductorin Thingy:53 laitteeseen C-kielellä ohjelmoituna. Konvoluutioneuroverkon periaate opittiin ja algoritmin osat saatiin ohjelmoitua Pythonilla. C-kieliset algoritmitoteutukset saatiin toteutettua Thingy:53 laitteessa, mutta kokonaisuutta integroitaessa jouduttiin toteamaan laitteen sisäisen muisti olevan liian pieni kaikkien algoritmin osien peräkkäiseen suoritukseen.

Äänikomentojen tunnistusta mikrokontrollerissa tekoälyn avulla

14.6.2024 - Jyrkkä Kari, Grönroos Thomas, Manninen Jere, Partanen Riku

Oulun ammattikorkeakoulun tietotekniikan tutkinto-ohjelmassa toteutettiin kevään 2024 neljännen periodin aikana 10 opintopisteen yritysprojektikurssi projektiopintoina. Projektin aiheena oli toteuttaa mikrofonilla nauhoitettujen äänikomentojen tunnistus Nordic Semicoductorin Thingy53-laitteelle. Äänikomentojen tunnistukseen käytetään konvoluutioneuroverkkoa, jolle syötetään äänisignaalista laskettu spektrogrammi eli ”kuva” äänikomennosta. Projektin haasteena oli yhtäältä tekoälyalgoritmin ymmärtäminen ja toisaalta sen toteutus muisti- ja suorituskykyrajoitteiseen mikrokontrolleriin.

Äänen tunnistusalgoritmit tunnistavat äänen rakenteita spektrogrammista, joka lasketaan aikatason äänisignaalista paloittaisten Fourier-muunnosten avulla. Tällöin saadaan aikaan ”kuva”, missä y-akselilla on äänen taajuussisältö ja x-akseli näyttää signaalin eri ajankohtina (kuva 1). Jos tällainen ”kuva” syötetään konvoluutioneuroverkolle, joka suodattimien avulla erottaa kuvasta piirteitä, voidaan näiden piireiden avulla tehdä luokittelu tavallisen neuroverkon avulla.



KUVA 1. Komennon ”left” aikatason aaltomuoto ja sen taajuus- ja aikatason spektrogrammi kuva [1].

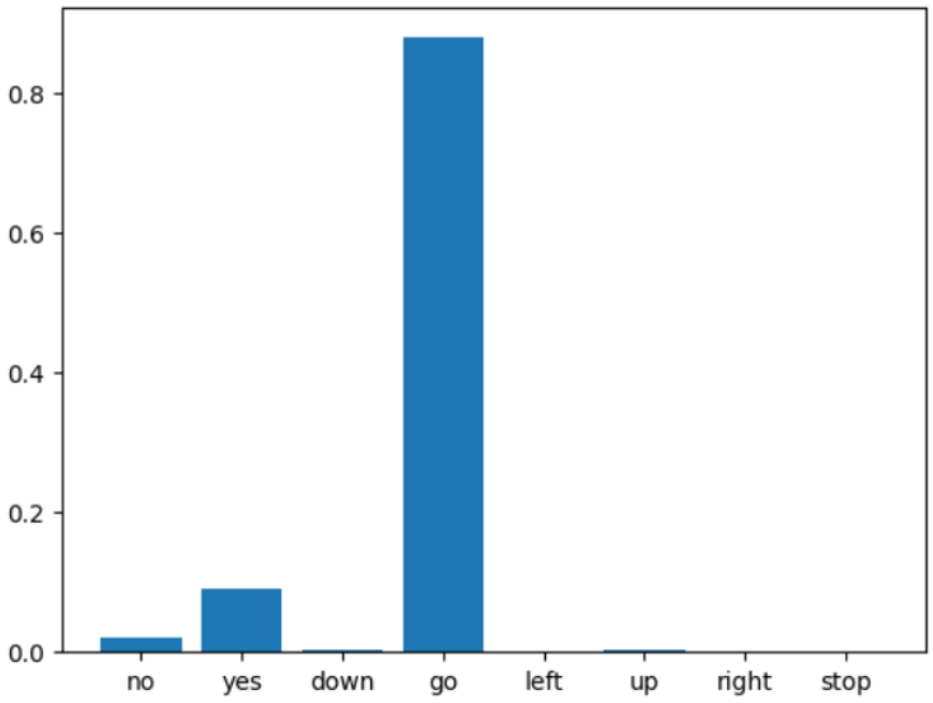
Kuvassa 2 on esitetty käytetty konvoluutioneuroverkon malli, jota hyödynnetään Google Colabissa mallin opettamiseen kahdeksan eri äänikomennon (stop, up, yes, down, no, left, right, go) avulla. Mallin opettamiseen käytettiin Speech Commands dataset [2] aineistoa. Projektissa haluttiin opetella mallin toimintaperiaate ja haluttiin ymmärtää, mitä tapahtuu kuvan 3 model.predict -funktion sisällä, kun äänen spektrogrammi syötetään opetetulle mallille ja tuloksena on kullekin kahdeksalle äänikomennolle todennäköisyysarvo kuvan 4 mukaisesti.

```
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    norm_layer,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels),
])
```

KUVA 2. Konvoluutioneuroverkon malli Google Colabissa [1].

```
y_pred = model.predict(test_spectrogram_ds)
```

KUVA 3. Äänikomennon tunnistaminen model.predict -komennolla [1].



KUVA 4. Tunnistuksen tulos eli äänikomentojen todennäköisyysarvot [1].

Projekti toteutettiin siten, että kullekin opiskelijalle annettiin ensin tehtäväksi toteuttaa joku mallin vaatimista vaiheista omalla Python aliohjelmalla. Näin eri algoritmin osat ja toimintaperiaate ymmärrettiin ja algoritmin osan oikeasta toiminnasta saatiin varmuus, kun oman Python aliohjelman tulosta verrattiin mallin antamaan tulokseen. Tämän jälkeen sama aliohjelma toteutettiin C- tai C++-kielellä, koska projektin lopullinen tavoite oli toteuttaa äänikomentojen tunnistus Nordic Thingy:53 mikrokontrollerialustalle. Alustassa on valmiina mikrofoni, jolla äänikomennot oli tarkoitus nauhoittaa.

Seuraavaksi projektiin osallistuneet opiskelijat kertovat algoritmin opetteluvaiheesta ja haasteista, joita koettiin algoritmia ja mikrofoniohjainta Nordic Thingy:53 alustalle koodatessa.

Äänikomentoja tunnistavan mallin opetteleminen

Projekti aloitettiin käymällä läpi Tensorflow:in esimerkki ”Simple audio recognition: Recognizing keywords” ja opettelemalla, miten kyseinen neuroverkko toimii vaihe vaiheelta. Esimerkissä oleva konvoluutioneuroverkko koulutettiin tunnistamaan sanat ”down”, ”go”, ”left”, ”no”, ”right”, ”stop”, ”up” sekä ”yes” Googlen tarjoaman 100 000 äänitiedostosetin avulla. Tämän jälkeen neuroverkko pystyi tunnistamaan annetussa äänitiedostossa sanotun sanan.

Oma konvoluutioneuroverkkomme rakenne ja toimintatapa olivat samat kuin Tensorflow:in esimerkissä, mutta Tensorflow:in valmiiden funktioiden käyttämisen sijaan toteutimme itse kaikki neuroverkon vaiheet aluksi Pythonilla, jonka jälkeen vielä uudestaan C:llä. Jaoimme neuroverkon eri funktioiden toteuttamiset ryhmämme jäsenten kesken niin, että jokaisella oli oma työstettävä osa-alue projektissa.

Tehtäväkseni valikoitui toisen konvoluutiofunktion toteutus. Projektia ennen kokemukseni konvoluution käytöstä neuroverkossa oli vain valmiin Tensorflow:in funktion kutsuminen, joten minun täytyi heti alkuun opiskella funktion toiminnallisuus perusteellisesti. Katsoin aiheesta paljon videoita, jotka visualisoivat prosessin eri vaiheita. Tämä auttoi ymmärtämään, mitä sisään menevälle datalle tarkalleen tapahtuu funktiossa sekä miten eri parametrit muuttavat tuloksia.

Konvoluutiofunktion toteutus Pythonilla lähti hieman hitaasti käyntiin vaaditun taustatutkimustyön takia. Myös koodin toimintamenetelmän suunnittelu vaati useamman päivän mietintää ja testausta. Näiden jälkeen ohjelmointi onnistui kuitenkin suhteellisen

sujakasti, ja minulla oli muutaman viikon jälkeen toimiva funktio. Projektin seuraava vaihe, funktion muuntaminen C-kielelle, onnistui vaivatta. Perusteellisen testaamisen jälkeen olin saanut omat vastuualueeni valmiiksi etuajassa, joten projektin lopuksi autoin vielä toista ryhmän jäsentä normalisointifunktion koodaamisessa C:llä.

Projekti oli hauska toteuttaa. Nautin suuresti siitä, että pääsin tekemään vielä yhden sellaisen opintojeni loppusuoralla. Toisen vuoden neuroverkkokurssista oli ehtinyt kulua jo useampi vuosi ja taitoni tästä osa-alueesta olivat ehtineet hieman ruostua, joten projektin ansiosta minulla tuli kerrattua ja opeteltua kyseistä osa-aluetta vielä perinpohjaisemmin.

Ryhmämme toimintamallina käytimme Scrum-menetelmää. Olin töissä ennen toiminut kyseisen prosessimallin parissa, mutta tällä kerralla olin viikon ajan ensimmäistä kertaa toimintamenetelmän vetäjänä. Scrum Masterin rooli pakotti minut olemaan perillä muiden ryhmän jäsenten tilanteista ja tuotoksista, minkä ansiosta pystyin projektin lopussa auttamaan kaikkien funktioiden yhdistelyssä yhdeksi toimivaksi ohjelmaksi.

Mikrokontrolleritoteutus

Äänentunnistusmallin toteuttaminen Nordicin Thingy:53 mikrokontrollerille tiedettiin hankalaksi jo projektin alussa. Saadaanko mallin vaatimat parametrit sopimaan muistirajoitettuun mikrokontrolleriin. nRF5340 SoC sisältää 1 MB sisäistä flash-muistia, joka sisältää suoritettavan ohjelmakoodin sekä read-only datan, ja 512 KB RAM muistia, josta 64 KB on varattu kommunikaatioon network coprocessorin kanssa. Lopulta itse prosessorilla on siis käytettävissä 448 KB RAM muistia ja noin 800 KB flash-muistia.

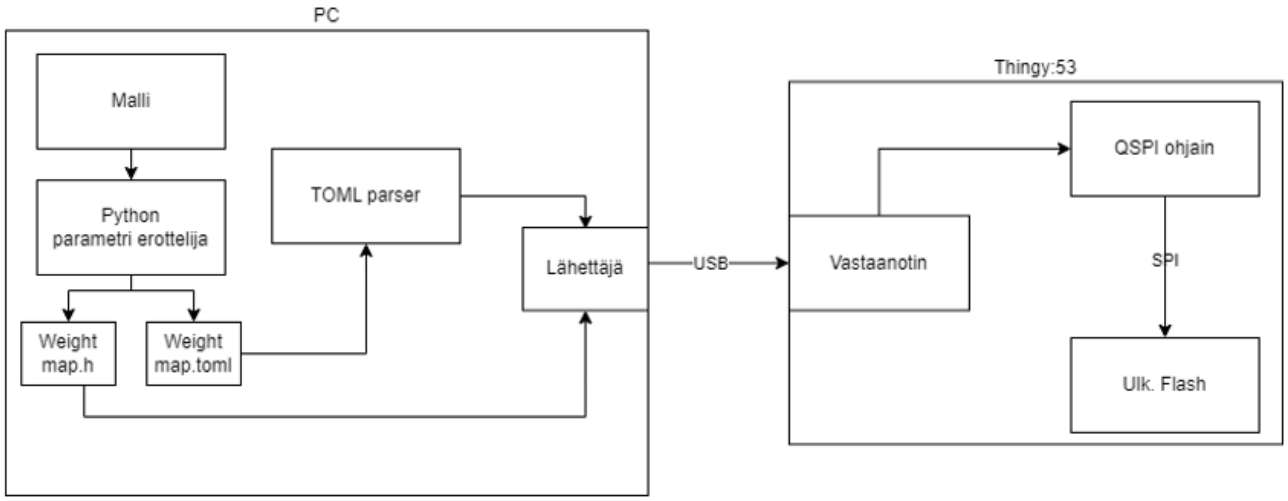
Projektissa toteutetussa äänikomentoja tunnistavassa mallissa on yhteensä 822 219 parametriä, joista jokainen vie perustarkkana liukulukuna, float, 32 bittiä eli 4 tavua. Yhteensä mallin koko on siis ~3,14 MB. Mallin mahduttaminen nRF5340 SoC:n sisäiseen flash-muistiin vaatisi mallin pienentämistä 24 prosenttiin sen alkuperäisestä koosta.

Yksi keino pienentää tarvittavan muistin määrää on käyttää 16-bittisiä, puolitarkkoja liukulukuja. Nimensä mukaisesti puolitarkat liukuluvut käyttävät puolet perustarkan liukuluvun bittimäärästä lukujen esitykseen, jonka seurauksena muistinkäyttö puolittuu, mutta myös tarkkuus ja mahdollisten esitettävien lukujen määrä pienenee. Puolitarkat liukuluvut ovat kuitenkin suosittuja koneoppimisen maailmassa, koska menetetyllä tarkkuudella ei välttämättä ole lopputuloksen kannalta suurta merkitystä.

Onneksi Nordic Semiconductorin Thingy:53 sisältää piirilevyllään 8 MB ulkoista flash-muistia, jonne alkuperäinen ~3,14 MB malli mahtuu komeasti. Normaalisti datan kirjoittaminen flash-muistiin pitäisi olla vain yhden linker-scriptin kirjoittaminen, jossa määritellään osio muistista ja data, joka sinne kirjataan.

Ulkoinen flash-muisti on kytketty QSPI-oheislaitteen kautta prosessoriin ja sitä ei saatu toimimaan linker-scriptien avulla yrityksistä huolimatta. Lopulta päätettiin, että olisi nopeampaa, ja mielenkiintoisempaa, kirjoittaa itse ajurit QSPI oheislaitteella ja ohjelma, joka osaa lähettää dataa sarjayhteyden välityksellä. Itse ajurit QSPI:lle olivat helpoin osa. Pitää vain siirtää oikeat arvot konfigurointirekisteriin ja kirjoitus tai luku onnistuu samalla periaatteella.

Vaikein osa oli kaiken ympärille tarvittava kuvan 5 mukainen järjestelmä. Ensimmäiseksi piti kirjoittaa ohjelma, joka osaa kaivaa kaikki painokertoimet opetetusta mallista, kirjaa ne C:n header-tiedostoon ja antaa ulos TOML-tiedoston, joka kuvailee kirjattua tietoa. Tämä toteutettiin Pythonilla. Seuraavaksi, piti kirjoittaa ohjelma, joka osaa lukea kyseisen TOML-tiedoston ja sen perusteella laskea muistiosoitteet painokertoimille ja yhdistää ne tasoon. Tämä toteutettiin C:llä. Lopulta, piti kirjoittaa ohjelma, joka lähettää datan mikrokontrollerille. Ohjelma kertoo mikrokontrollerille, että se haluaa kirjata dataa, paljonko dataa se haluaa kirjata, ja lähettää datan. Mikrokontrolleri vastaa näihin komentoihin joko ACK (acknowledged) tai NAK (not acknowledged), riippuen siitä, täyttikö komento kaikki vaatimukset.



KUVA 5. Mallin parametrien siirtojärjestelmä.

Varsinaisen neuroverkon koodin toteuttaminen mikrokontrollerille ei poikkea normaalista C-ohjelmasta, joka olisi kirjoitettu pöytäkoneelle juurikaan, mutta mikrokontrollerin

rajallisen stack-muistin (pinomuisti) takia algoritmin input eli spektrogrammi ja välitulokset täytyy säilöä heap-muistiin (kekomuisti). Myös verkon käyttämien kerneleiden parametrien määrä on liian iso (3,14 MB) mahduttaa RAM-muistiin, joten parametrit täytyy lukea ohjelman ajon aikana Flash-muistista suoraan. Thingy:53:n ulkoinen 8 MB:n Flash-muisti mahdollistaa jopa noin 8 MB neuroverkon käytön, kunhan välitulosten ja spektrogrammin koko pysyy Thingy:53:n rajallisen sisäisen RAM-muistin puitteissa.

Projektin aikana ehdittiin toteuttaa mallin parametrien siirtäminen mikrokontrollerille sekä mikrofonin datan lukeminen ja sen muuntaminen spektrogrammi algoritmin vaatimaan muotoon. Itse mallin toteutusta ei saatu toimimaan mikrokontrollerilla projektin rajallisen ajan ja mikrokontrollerin rajallisen RAM-muistin takia.

Projekti oli kokonaisuudessaan erittäin mielenkiintoien ja toi mukanaan mielenkiintoisia ongelmia esimerkiksi rajallisen muistin optimoinnin suhteen. Projektissa opin, kuinka konvoluutioneuroverkot toimivat ja mitä vaaditaan sellaisen ajamiseen rajallisessa ympäristössä.

Projektin tulokset

Projektin tuloksena kahdeksan opiskelijaa sai suoritettua 10 opintopisteen yritysprojektin kevään 2024 haastavassa tilanteessa, missä monet opiskelijat ovat jääneet vaille yritysprojektipaikkaa tai kesäharjoittelupaikkaa. Opiskelijat kertasivat ja oppivat Python- ja C-kielen taitojaan projektin tehtävissä ja äänikomentojen tunnistuksen periaate konvoluutioneuroverkkojen avulla tuli kaikille tutuksi.

Jo projektin alussa oli selvää, että Nordicin Thingy:53 alusta tulee aiheuttamaan suurimmat ongelmat. Projektin aikana tuli todistettua, että alustan mikrofoni saadaan toimimaan ja että suurenkin tekoälymallin vaatimat parametrit saadaan talletettua laitteen ulkoiseen muistiin ja talletettuja arvoja voidaan hyödyntää ohjelmaa ajettaessa. Ongelmaksi muodostui laitteen pieni sisäinen RAM-muisti, joka ei riittänyt ainakaan dynaamista muistinhallintaa käytettäessä. Nähtäväksi jää, voisiko puhekomentojen tunnistusalgoritmit saada toimimaan staattisen muistinvarauksen avulla, missä sisäiseen muistiin varataan kaksi isoa muistilohkoa: toinen inputille ja toinen laskennan outputille.

Projekti oli selkeästi opiskelijoille mieluinen. Projektiryhmä työskenteli aktiivisesti koululla joka päivä ja saavutetut tulokset yllättivät projektiohjaajan. Puhekomentojen tunnistusalgoritmit saatiin toteutettua Python- ja C-kielisinä aliohjelmina. Projektin

loppuesityksessä ryhmä joutui toteamaan, että toimivaa Nordic Thingy:53 demoa ei saatu aikaan muistin riittämättömyyden takia. C-kielisen toteutuksen integroiminen toimivaksi kokonaisuudeksi yksittäisistä C-kielisistä algoritmin osista jäi myös kesken. Jere ja Thomas jatkoivat C-kielisen toteutuksen integrointia omasta mielenkiinnostaan ja saivat sen toimimaan pian projektin loppuesityksen jälkeen.

Kari Jyrkkä
lehtori
ICT ja liiketoiminta
Oulun ammattikorkeakoulu

Thomas Grönroos
Opiskelee Oulun ammattikorkeakoulussa ICT ja liiketoiminta -osaamisalalla

Jere Manninen
Opiskelija
Opiskelee Oulun ammattikorkeakoulussa ICT ja liiketoiminta -osaamisalalla

Riku Partanen
Opiskelija
Opiskelee Oulun ammattikorkeakoulussa ICT ja liiketoiminta -osaamisalalla

Lähteet

[1] Tensorflow. (23.3.2024). *Simple audio recognition: Recognizing keywords*. Haettu 19.4.2024 osoitteesta https://www.tensorflow.org/tutorials/audio/simple_audio

[2] Tensorflow. (13.1.2023). *speech_commands*. Haettu 19.4.2024 osoitteesta https://www.tensorflow.org/datasets/catalog/speech_commands