jamk

Teollisen tuotantolaitteiston pyörimisnopeuden määritys mitatusta värähtelynäytteestä koneoppimisen menetelmin

Tommi Huhta

Opinnäytetyö AMK

Lokakuu 2022

Tietojenkäsittely ja tietoliikenne

Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Tommi Huhta

Teollisen tuotantolaitteiston pyörimisnopeuden määritys mitatusta värähtelynäytteestä koneoppimisen menetelmin

Jyväskylä: Jyväskylän ammattikorkeakoulu. Lokakuu 2022, 58 sivua.

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Taustaa: Teollisen laitteen käyntinopeus on olennainen tieto laitteiston toiminnan ja koneiden kunnon valvomisessa. Sen voi mitata sisäisellä anturilla, mutta sellaisen asentaminen on usein vaikeaa, kallista tai mahdotonta, ja edellyttää jatkuvaan toimintaan suunnitellun prosessin pysäyttämistä. Siksi olisi hyödyllistä määrittää nopeus ulkoisen anturin tuottamasta datasta. Laitteen itsensä tai ympäristön aiheuttamien sivuäänten johdosta ratkaisu ei ole yksinkertainen. **Tavoite**: Tutkimuksessa pyrittiin selvittämään voiko pyörimisnopeuden selvittää koneoppimisen, erityisesti syväoppimisen avulla. Datana käytettiin tuhansia kiihtyvyysantureiden PCM-näytteitä erilaisista pumpuista, tuulettimista ja sähkömoottoreista. **Menetelmät**: Erilaisia näytteistä signaalinkäsittelyn menetelmin erotettuja syötteitä käytettiin opetusdatana neuroverkoille. Neuroverkkomalleja koottiin itse kokeilemalla kerroksia ja rakenteita sekä testattiin valmiita esikoulutettuja malleja. Ohjelmia esikäsittelyyn, mallien käsittelyyn, opetukseen ja evaluointiin kirjoitettiin Pythonilla. **Tulokset**: Erilaisista syötedatoista parhaimmat ennusteet saatiin spektrogrammeilla. Useimpien laitetyyppien ennusteiden virheiden keskihajonnaksi saatiin alle 40 kierrosta minuutissa mitatusta datasta, joka on 1…3 prosenttia tyypillisestä käyntinopeudesta. Kuitenkin huomattavan paljon ennusteita poikkesi mitatuista suuresti. Siten syötteiden erottaminen raakadatasta ja neuroverkkomalleissa vaatii kehittämistä ennen tulosten soveltamista tuotantoon.

Avainsanat (asiasanat)

koneoppiminen, neuroverkot, syväoppiminen, värähtelyt, signaalikäsittely, TensorFlow, NumPy, SciPy

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Tommi Huhta

Determining rotational speed of industrial process equipment from measured vibration data sample by using machine learning methods

Jyväskylä: JAMK University of Applied Sciences, October 2022, 58 pages.

Information and Communication Technology. Degree Programme in Information and Communication Technology. Bachelor's thesis.

Language of publication: Finnish

Permission for open access publication: Yes

Abstract

Background: Rotational speed of industrial devices is essential information for monitoring operation and machine health. It can be measured with an internal sensor, but often installing it would be difficult, expensive or impossible, as it could require stopping process designed for continuous operation. Therefore it would be beneficial to determine the speed from data from an external sensor. The solution is non-trivial due to interference in the measured signal from side tones from the device and the environment. **Goals**: The aim was to study if rotational speed could be determined by machine learning methods, especially deep learning. Data used was thousands of PCM samples from acceleration sensors attached to various devices, such as pumps, fans and electric motors. **Methods**: Various kinds of feature data was extracted from the samples with methods of signal processing and then used to train various neural networks using TensorFlow. Different kinds of layers and structures and also pre-trained models were tested. Programs for preprocessing, training, handling and evaluating models were written in Python. **Results**: Of the tested input data, most prominent were spectrograms. For most types of machinery, accuracy evaluated by standard deviation of less than 40 RPM, which corresponds to approximately 1…3% of typical running speed, was achieved. However significant number of predictions deviated from measured values by a large margin. Therefore data extraction and the neural models require further development before being ready for production applications.

Keywords/tags (subjects)

machine learning, neural networks, deep learning, vibration data, signal processing, TensorFlow, NumPy, SciPy

Miscellaneous (Confidential information)

Sisällys

1 Johdanto **4**

1.1 Tutkimusmenetelmät 4

1.2 Tutkimuskysymykset ja tavoitteet 5

1.3 Toimeksiantajasta 5

2 Signaalinkäsittely **6**

2.1 Pulssikoodimodulaatio 6

2.2 Fourier-muunnos 7

2.3 Diskreetti Fourier-muunnos 8

2.4 Ikkunointi 9

2.5 Spektrogrammi 10

2.6 Autokorrelaatio 11

2.7 Kiihtyvyys-, nopeus- ja siirtymäsignaali 12

3 Koneoppiminen **13**

3.1 Tekoälyn ja koneoppiminen käsite 13

3.1.1 Tekoäly 13

3.1.2 Koneoppiminen 14

3.2 Regressio 15

3.3 Syväoppiminen 16

3.3.1 Johdatus 16

3.3.2 Neuroverkon rakenne ja toiminta 17

3.3.3 Vastavirtaan syöttö 19

3.3.4 Konvoluutio 20

3.3.5 Käsitteitä 21

4 Suunnitelma **21**

4.1 Työkalut 21

4.2 Datan käsittely 22

4.3 Neuroverkot ja niiden kouluttaminen 22

5 Toteutus **23**

5.1 Datan muoto . . . 23

5.2 Esikäsittely . . . 23

5.2.1 Näytelistan koostaminen . . . 23

5.2.2 Muunnos nopeusdataksi . . . 24

5.2.3 Spektrogrammien muodostaminen . . . 24

5.2.4 Ominaisuuksien erottaminen . . . 25

5.2.5 Jako opetus-, validointi- ja testidataan . . . 25

5.3 Opetusohjelma . . . 26

5.4 Takaisinkutsut . . . 26

5.5 Syötteiden syöttö . . . 27

5.6 Omien mallien muodostus . . . 28

5.7 Valmiiden mallien soveltaminen . . . 28

5.8 Lineaarinen regressio . . . 29

5.9 Käytetty ohjelmisto . . . 29

6 Tulokset **30**

6.1 Tulosten analysointi . . . 35

6.1.1 Syötedatat . . . 35

6.1.2 Parhaiden ennusteiden ja mallien lähempi tarkastelu . . . 35

6.2 Menetelmän tarkkuus ja luotettavuus . . . 36

7 Yhteenveto **36**

7.1 Pohdintaa . . . 36

7.2 Jatkokehitysmahdollisuudet . . . 37

Lähteet **38**

Liitteet **40**

Liite 1. Näytelistan generoiva ohjelma . . . 40

Liite 2. Kiihdyvyysdatan nopeusdataksi muuntava ohjelma . . . 42

Liite 3. Ominaisuuksia erottava ohjelma . . . 44

Liite 4. Spekrogrammit generoiva ohjelma . . . 48

Liite 5. Ohjelma mallien koulutukseen ja evaluaatioon 50
Liite 6. Neuroverkkomalli s_d1c2n2 57
Liite 6. Neuroverkkomalli s_d1c3n2 57
Liite 6. Neuroverkkomalli s_mobile2 / s_mobi-accel 58

Kuviot

Kuvio 1. Pulssikoodimodulaation muodostus 6
Kuvio 2. DFT kitaran A-nuotista 8
Kuvio 3. Ikkunointi ja vaikutus Fourier-muunnokseen 10
Kuvio 4. spektrogrammi kitaran G-soinnusta 10
Kuvio 5. Regressio: kalat 15
Kuvio 6. Neuroverkko 19
Kuvio 7. Konvoluutio 20
Kuvio 8. Esikäsittelyn työnkulku 23
Kuvio 9. Kuvakaappaus TensorBoardin oppimiskäyrästä 27

Taulukot

Taulukko 1. A-nuotin voimakkaimmat taajuudet 9
Taulukko 2. Aktivaatiofunktioita 18
Taulukko 3. Termistöä 21
Taulukko 4. DataFrameen talletetut tiedot 24
Taulukko 5. Käytetyt ohjelmistot 29
Taulukko 6. Ennusteiden keskihajonnat: osa 1 31
Taulukko 7. Ennusteiden keskihajonnat: osa 2 32
Taulukko 8. Parhaat mallit seteittäin 33
Taulukko 9. Mallien ennusteiden kuvaajia 34

1 Johdanto

Teollinen tuotantolaitteisto koostuu muun muassa erilaisista sähkömoottoreista, joiden kuntoa ja toimintaa tarkkaillaan. Tärkeä tieto moottorista on sen käyntinopeus, jotta nopeutta voidaan säätää oikeaksi tuotantoprosessin toimivuuden kannalta. Käyntinopeutta käytetään myös vikojen tunnistamiseen, kun käyntinopeuden harmonisilta taajuuksilta etsitään poikkeamia jotka voivat kertoa koneessa olevasta viasta.

Häiriöttömästä anturin signaalista moottorin käyntinopeus voidaan saada selville matemaattisesti. Koneen värinää voidaan verrata moottorin äänenkorkeuteen, jolloin moottorin käyntinopeus voidaan päätellä siitä. Nopeuden määrittäminen kuitenkin vaikeutuu, jos värinässä on sivuääniä. Sivuäänet saattavat johtua laakeriviasta, jonkun toisen koneen tai muun värinälähteen aiheuttamasta häiriöstä datassa. Oletuksena kuitenkin on, että värinädata sisältää tarvittavan informaation ja se on sieltä löydettävissä.

Neuroverkkoja käytetään paljon hahmon tunnistamiseen kuvista. Tyypillinen haaste on setti valokuvia, joista pyritään tunnistamaan ja luokittelemaan niissä esiintyvä hahmo. Kuvissa on esimerkiksi eri koirarotuja, lintulajeja, eri tyyppisiä esineitä tai muuta tunnistettavaa. Valokuvassa on kuitenkin lähes aina muutakin kuin itse kohde. Neuroverkko kuitenkin löytää kuvasta olennaiset piirteet, joilla kuvan varsinainen kohde voidaan tunnistaa. Ajatuksena oli, että neuroverkko samalla tavoin löytää värinädatasta piirteet joista käyntinopeus tunnistetaan.

1.1 Tutkimusmenetelmät

Työnkulku etenee mittaussignaalien käsittelystä koneoppimismallien luomiseen, kouluttamiseen ja tulosten analysointiin. Koneiden värinädata on tallennettu 48 kilohertsin näytteenottotaajuudella 24 bittiseen PCM-muotoon ja useimpien näytteiden pituus ajallisesti on noin kolme sekuntia. Näytteitä on otettu erilaisista moottoreista eri ajanjaksoina. Näytteistä tehdään erilaisia matemaattisia analyysejä, kuten Fourier-analyysi, autokorrelaatio, ja spektrogrammi. Analyysien tuloksia syötetään neuroverkolle, joita kokeillaan erilaisia sekä itse rakennettuja, että valmiita. Ohjelmointikielenä käytetään Pythonia. Neuroverkot toteutetaan TensorFlow-kirjastolla. Signaalianalyysiin käytetään mm. NumPy- ja SciPy-Python-moduuleita.

1.2 Tutkimuskysymykset ja tavoitteet

Tutkimuksessa pyrittiin saavuttamaan tuloksia, joita voitaisiin hyödyntää kaupallisissa sovelluksissa tuotantolaitteistojen valvonnassa. Mitään heti käyttöön otettavaa tuotetta ei kuitenkaan pyritty valmistamaan.

Työssä pyrittiin selvittämään:

- Voiko koneoppimisella saavuttaa riittävän luotettavaa ja tarkkaa käyntinopeuden määritystä?
- Mitä dataa anturin signaalista voidaan eristää? Miten nämä toimivat neuroverkkojen kanssa?
- Millainen rakenne neuroverkolla toimii parhaiten? Millaiset kerrokset ovat hyödyllisiä?
- Mitkä valmiit neuroverkot soveltuvat parhaiten ongelmaan?
- Voidaanko saatuja tuloksia hyödyntää kaupallisesti?

1.3 Toimeksiantajasta

Distence Oy on teknologiayritys, jonka tavoitteena on maksimoida teollisten laitteiden hyöty elinkaaren aikana. Tavoitteen saavuttamiseksi yritys kehittää ja toimittaa kunnonvalvontaan tarkoitettua SaaS-tuotetta (System as a Service) asiakkaille, jotta heidän operoimansa teollisten prosessien pyörivät laitteet kestävät pidempään, tuottavat enemmän ja suoriutuvat tehtävistään optimaalisella tavalla. Tällaisia laitteita ovat tyypillisesti sähkömoottorit, vaihteistot, pumput ja puhaltimet. SaaS-tuote koostuu valvottavaan kohteeseen asennettavasta päätelaitteesta antureineen, ja pilvipalvelusta, jonne tietoa kerätään kentältä internetin yli. Kerättyä tietoa analysoidaan automaattisesti ja asiakkaalle koostetaan päätösten tueksi tieto lähestyvästä huoltotarpeesta, jotta huolto on mahdollista suunnitella, resursoida ja toteuttaa optimaaliseen aikaan niin, että laitteen kunto ei pääse heikkenemään merkittävästi, eikä huoltoja ei myöskään tarvitse tehdä tarpeettomasti.

Perinteisesti laitteiston kuntoa on valvottu kiertämällä kohteita mittalaitteiden kanssa. Tällöin tarkastuksien välissä saattaa kulua pitkäkin aika, jona kone saattaa vikaantua ja hajota. Distence tarjoaa mallia, jossa koneet keräävät taustalla jatkuvasti tietoa, mutta pyytävät ihmisen huomiota niihin kohteisiin, joissa apua tarvitaan. Näin ihmisten aika vapautuu tulkitsemaan huoltotarpeita ja tekemään päätöksiä.

Distence on aloittanut toimintansa 20 vuotta sitten erilaisten koneiden valvontaan liittyvillä projekteilla. Myöhemmin projektit on päätetty tuotteistaa yhdeksi tuotteeksi, joka palvelee useita sektoreita selkeän strategian mukaisesti. Täten toiminta on uudistunut ja on nyt käytännössä uudelleen start-up vaiheessa. Kiinnostus laitteiston kunnonvalvontaan vaikuttaa olevan nousussa ja kysyntä myös Distencen tarjoamille palveluille kasvussa. Nyt Distence on uuden kasvun alkuvaiheessa. Tällä hetkellä Distence työllistää 10 henkeä, mutta asiakkaiden kautta toimintaa on jo yli 40 maassa.

2 Signaalinkäsittely

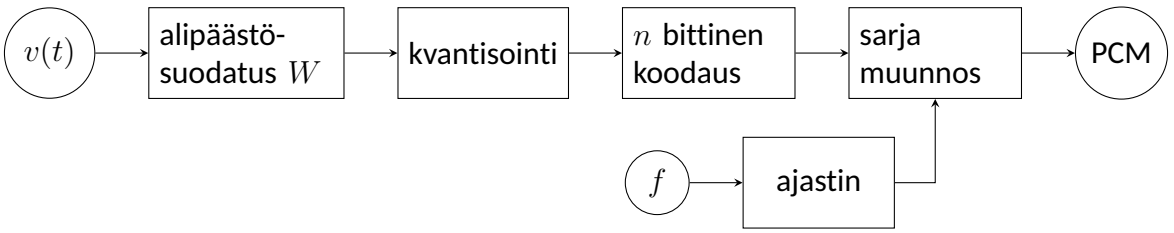
Signaalin käsittely on laaja asiakokonaisuus. Tässä käsitellään signaalinkäsittelyä siltä osin, kun se on oleellista tutkimuskohteen kannalta.

2.1 Pulssikoodimodulaatio

Pulssiamplitudimodulaatiolla (PCM) muunnetaan signaalin aaltomuoto sarjaksi kvantisoituja digitaalisia koodisanoja. Koodisana on luku, käytännössä yleensä tietty määrä bittejä. Jos koodisanassa on n bittiä kvantisoituja tasoja, niin

$$q = 2^n \tag{1}$$

Näytteenottotaajuus, eli montako koodisanaa modulaatiossa on aikayksikköä kohden on oltava vähintään kaksinkertainen suurimpaan haluttuun taajuuteen, joka halutaan koodata. Kuviossa 1 esitetään A/D-muunnin (ADC), jossa muodostetaan pulssikoodimoduloitu signaali, jossa Näytteenottotaajuus on f ja koodisana n-bittinen.



Kuvio 1: Pulssikoodimodulaation muodostus

Signaalista $v(t)$ suodatetaan pois taajuutta $W \leq \frac{f}{2}$ korkeammat taajuudet. Tämän jälkeen signaali kvantisoidaan pyöristämällä lähimpään tasoon 2^n vaihtoehdosta ja muunnetaan biteiksi. Kun näistä

biteistä otetaan ajastimen tahdittamana näytteitä, saadaan PCM-dataa. (Carlson, Crilly & Rutledge 2002, 495-497)

PCM-signaali on siis kvantisoitua, toisin sanoen **diskreettiä**, josta johtuen signaalia ei voida täydellisesti rekonstruoida. Käyttökohteesta riippuen tietty laatu on riittävä. PCM:n laatu on sitä parempi, mitä suurempi on näytteenottotaajuus ja mitä pidempi koodisana on. Esimerkiksi audio CD:ssä ääni on koodattu PCM:nä, jossa näytteenottotaajuus on 44100 Hz (jolloin $W_{max} = 22050Hz$) ja koodisanan pituus 16 bittiä. Tämä riittää kattamaan ihmisen kuuloalueen, jona yleisesti pidetään $20...20000Hz$.

2.2 Fourier-muunnos

Fourier muunnos on keino laskea signaalin spektri, eli miten signaali jakautuu eri taajuuksille. Olkoon $V(f)$ Fourier muunnos signaalille $v(t)$, tällöin

$$V(f) = \int_{-\infty}^{\infty} v(t)e^{-i2\pi ft}dt \tag{2}$$

jossa f on taajuus, t on aika. Termi $e^{-i2\pi ft}$ siirtää signaalin kompleksitasolle. Tämä siis integroidaan kaiken ajan yli. Saatu funktio on myös kompleksinen, siten jollekin taajuudelle saatu muunnosfunktion arvo on myös kompleksinen.

Muunnoksesta voidaan laskea takaisin signaali. Käänteinen Fourier-muunnos

$$v(t) = \int_{-\infty}^{\infty} V(f)e^{i2\pi ft}df \tag{3}$$

tuottaa alkuperäisen signaalin. (Carlson ym. 2002, 33)

Joissain lähteissä käytetään taajuuden tilalla kulmanopeutta w. Tällä ei sinänsä ole väliä, suureet ova kuitenkin suoraan verrannollisia ($w = 2\pi f$).

Käytännössä matemaattinen Fourier-muunnos on teoreettinen ja sen laskeminen tosimaailman sovelluksissa mahdotonta. Siksi joudutaankin turvautumaan approksimaatioihin.

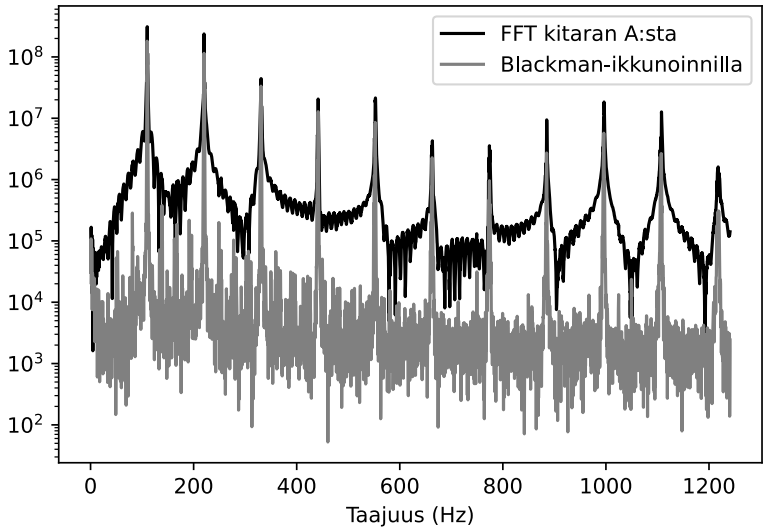
2.3 Diskreetti Fourier-muunnos

Diskreetille datalle, kuten PCM, sopii **nopean Fourier-muunnoksen** (FFT) alalaji, diskreetti Fourier-muunnos (DFT). N näytettä pitkän sarjan $x[n]$ DFT on

$$y[k] = \sum_{n=0]}^{N-1} x[n] e^{2\pi i \frac{kn}{N}} \tag{4}$$

Tämä tuottaa sarjan, jossa puolivälin jälkeen taajuudet ovat negatiivisia. Nämä taajuudet voi yleensä jättää huomiotta (Fourier Transforms – scipy.fft 2022). Taajuudet jakautuvat siten, että sarjan puoliväli vastaa taajuutta $\frac{1}{2} f_s$, eli puolta näytteenottotaajuudesta.

Esimerkissä on tehty diskreetti Fourier-muunnos äänitetystä kitaralla soitetusta A-nuotista, jonka perustaajuus on (oikein viritettynä) 110 Hz (ISO 16:1975 2016). Logaritmisessa kuviossa 2 näkyy äänen perustaajuus sekä sarja harmonisia taajuuksia. Blackman-ikkunoinnin läpi ajetusta signaalista erottuvat taajuudet alkuperäistä korkeampina piikkeinä. Ikkunoinnista lisää luvussa 2.4.



Kuvio 2: DFT kitaran A-nuotista

Etsimällä huiput löydetään nuotin perustaajuus ja harmoniset taajuudet. Edeltävästä on etsitty huiput Pythonin `scipy.signal.find_peaks()`-funktiolla ja lajiteltu voimakkuusjärjestykseen. Viisi voimakkainta taajuutta on esitetty taulukossa 1. Harmonisen voimakkuus on ilmaistu desibeleinä

perustaajuuteen nähden. Tulos vahvistaa teorian, jonka mukaan värähtelyllä on perustaajuus ja harmoniset taajuudet ovat

$$f_n = n \cdot f_0 \mid n \in \mathbb{N}, \tag{5}$$

jossa f_0 on perustaajuus (Fundamental Frequency and Harmonics n.d.).

Taulukko 1: A-nuotin voimakkaimmat taajuudet

taajuus	dB
110.0	0.00
220.2	-1.18
330.6	-8.47
552.8	-11.62
441.6	-11.79
996.6	-13.44

2.4 Ikkunointi

Kun tarkastellaan pitkästä signaalista otettua näytettä, siten että leikataan signaalista osa käsittelyyn, signaali alkaa ja päättyy yhtäkkisesti. Nämä kohdat aiheuttavat signaalia käsitellessä vääristymiä, ylimääräisiä taajuuksia, joita ei alkuperäisessä signaalissa ole. Tätä sanotaan taajuusvuodoksi (frequency leakage). Pelkkä näyte vastaa **ikkunointifunktiota**, joka on suorakaiteen muotoinen, 1 ajalta, jolta näyte on, 0 muutoin. (Prabhu 2014, 69-71)

Näitä vääristymiä vähennetään ikkunointifunktiolla, joka ”liu’uttaa” signaalin sisään ja ulos ”pehmeästi”. Ikkunafunktion arvo ikkunan ulkopuolella on nolla ja ikkunan keskellä 1, tai lähellä sitä. Kertomalla signaali ikkunafunktiolla saadaan ikkuna.

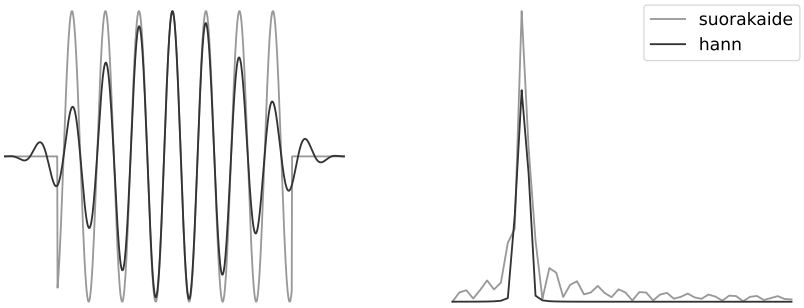
Eräs yleisesti käytetty ikkunointifunktio on Hann, jonka Prabhu (2014) kirjoittaa muotoon

$$f(t) = \begin{cases} 0.5 + 0.5cos(\frac{\pi t}{\tau}), & |t| \leq \tau \\ 0, & : \; muulloin \end{cases} \tag{6}$$

SciPy:n manuaalista funktion scipy.signal.windows.hann (2022) ohjeesta löytyy diskreetti määritelmä

$$w(n) = 0,5 - 0,5 \cdot cos(\frac{2\pi n}{M-1}) \;\; | \;\; 0 \leq n < M-1 \tag{7}$$

Kuviossa 3 nähdään vasemmalla sinin muotoinen signaali suorakaiteen muotoisella ikkunalla ja Hann-ikkunalla, sekä oikealla näistä tehty Fourier-muunnos.

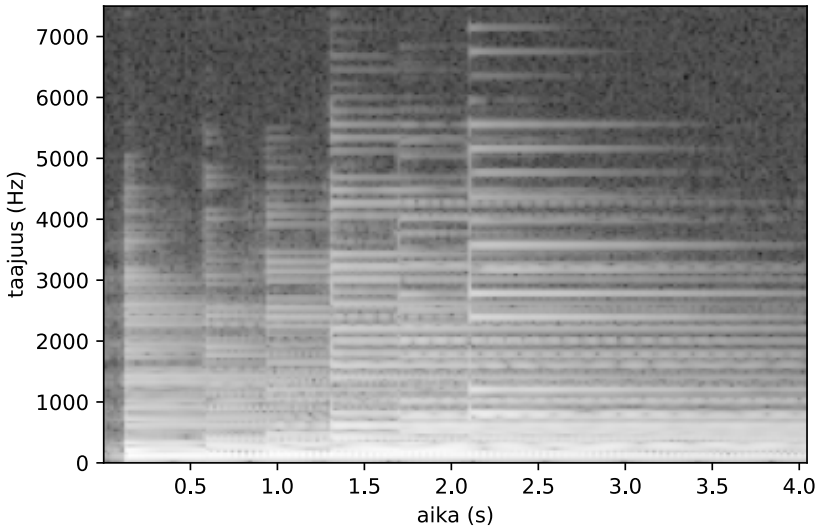


Kuvio 3: Ikkunointi ja vaikutus Fourier-muunnokseen

Puhtaassa sinisignaalissa esiintyy vain yhtä taajuutta. Suorakaideikkunassa signaali katkaistaan mielivaltaisesta kohdasta, joka näkyy muunnoksessa taajuusvuotona. Ikkunoinnin avulla taajuusvuoto on saatu lähes häivytettyä.

2.5 Spektrogrammi

Spektrogrammissa sarjasta Fourier-muunnoksia koostetaan kuva siten, että siitä voidaan havainnoida signaalin taajuusjakauma eli spektri kullakin ajan hetkellä.



Kuvio 4: spektrogrammi kitaran G-soinnusta

Kuviossa 4 spektrogrammi kitaralla arpeggiotyyliin soitetusta G-duurisoinnusta. Sointu koostuu nuoteista G_2, H_2, D_3, G_3, H_4 ja G_5. Kuviossa näkyy nuottien harmonisten taajuuksien kohoaminen soitettaessa kieliä alimmasta ylimpään.

2.6 Autokorrelaatio

Korrelaatio on monille tuttu ainakin käsitteenä. Suureena korrelaatio kertoo, onko asioilla jotain yhteyttä. Matemaattinen korrelaatio määritellään suhteiden kertoman odotusarvolla. Jos määritellään, että $g(h)$ on funktio sille, paljonko henkilö h pelaa väkivaltaisia videopelejä ja funktio $c(h)$ kertoo kuinka usein ja vakaviin väkivaltarikoksiin hän on syyllistynyt. Tällöin korrelaatio on

$$R_{gc} = E[g(h)c(h)]. \tag{8}$$

Jos haluaa tietää vielä korreloivatko h(h) ja c(h), pitää tietää, mikä ei korreloi. Jos

$$E[g(h)c(h)] = E[g(h)] \cdot E[c(h)], \tag{9}$$

Suureet eivät korreloi. Tästä poikkeavat arvot osoittavat yhteyttä eli korrelaatiota.

Autokorrelaatio kertoo suureen jaksollisuudesta. Signaalin $v(t)$ keskiarvo ajalla t on sama, kuin sen odotusarvo.

$$\overline{v(t)} = E[v(t)]. \tag{10}$$

Huomaa, että odotusarvo $E[v(t)]$ on ”kokoonpanomuuttuja” ja se saa keskiarvoa suuremman merkityksen yhdistelemällä, kuten tässä tapauksessa **autokorrelaation** määritelmässä

$$R_v(t_1, t_2) = E[v(t_1)v(t_2)] \tag{11}$$

(Carlson ym. 2002, 354).

Funktio kuvaa suhdetta funktion $v(t)$ arvoille ajan hetkillä t_1 ja t_2. Useimmiten jaksollisuuksia etsiessä käytetään muuttujana viivettä, eli aikaeroa τ.

$$R_{xx}(\tau) = E[v(t)v(t+\tau)] \tag{12}$$

(Aumala, Ihalainen, Jokinen & Kortelainen 1998, 72)

Suurin korrelaatio saadaan, jos merkitään $\tau = 0$ kaavassa 12 tai $t_1 = t_2$ kaavassa 11. Tällöinhän tekijät otetaan samasta kohtaa, joten

$$E[v(t)v(t)] = E[v^2(t)] = \overline{v^2(t)}. \tag{13}$$

Jos taas

$$R_{xx}(\tau) = \overline{v(t)}^2, \tag{14}$$

korrelaatiota ei ole. Kaavan 13 arvo on signaalissa samalla sen keskimääräinen teho. Jos signaalin keskiarvo on 0 (johon se usein kannattaa kalibroida), on myös kaavan 14 arvo 0. (Aumala ym. 1998; Carlson ym. 2002)

Diskreetille signaalille (kuten PCM) voidaan käyttää estimaattia

$$r_{xx}[k] = \frac{1}{N-k} \sum_{n=1}^{N-k} x[n]x[n+k], \tag{15}$$

jossa N on näytteiden määrä, k viive. (Aumala ym. 1998, 78)

2.7 Kiihtyvyys-, nopeus- ja siirtymäsignaali

Kiihtyvyys a on nopeuden muutosta ajan suhteen ja nopeus v on siirtymän x muutosta ajan suhteen, jolloin

$$\begin{gathered} a = \frac{\partial v}{\partial t} = \frac{\partial^2 v}{\partial^2 t} \\ \int (a)dt = v = \frac{\partial a}{\partial t} \\ \int \int (a)dt^2 = \int (v)dt = x. \end{gathered} \tag{16}$$

Jos nopeus alkuhetkellä v_0 tiedetään, saadaan kiihtyvyydestä nopeus ajan hetkellä t määrätyllä integraalilla

$$v_t = \int_0^t (a)dt + v_0. \tag{17}$$

(Wren 2010)

Diskreetissä muodossa integraali näytteen n kohdassa saadaan summaamalla:

$$v_n = \sum_{i=0}^{n} \frac{a_i}{f} + v_0 = \frac{\sum_{i=0}^{n}}{f} + v_0, \tag{18}$$

jossa f on näytteenottotaajuus ja v_0 alkunopeus.

3 Koneoppiminen

3.1 Tekoälyn ja koneoppiminen käsite

3.1.1 Tekoäly

Fry (2018) kertoo tarinan Claude Shannonin vierailusta Alan Turingin luona. Turing suunnitteli ohjelmaa, joka pelaa shakkia. Shannon tuumasi, että jos kone joskus voisi pärjätä, tai jopa voittaa maailman parhaimman shakinpelaajan, oltaisiin lähempänä singulariteettia, hetkeä jolloin kaikki muuttuu. 1996 IBM:n Deep Blue -supertietokone voitti shakin maailmanmestari Garri Gasparovin. Oliko Deep Blue älykäs? Se pystyi laskemaan 200 miljoonaa mahdollista asetelmaa sekunnissa ja valitsi siirron, joka takasi parhaat asetelmat jatkossa. Gasparov pelasi älyllä, Deep Blue raa'alla laskennalla. Tapahtumaa ei pidettykään suurena epookkina – shakinpeluutaitoa ei enää pidettykään merkkinä ihmismielen suuruudesta. Tietojenkäsittelytieteilijä Larry Tesler summasi ilmiön jo noin vuonna 1970: "Älykkyys on sitä mitä koneet eivät ole vielä tehneet". Tätä sanotaan Teslerin teoreemaksi. (Fry 2018; Girardin 2018)

Merriam-Webster määrittelee tekoälyn *tietojenkäsittelyntieteen haaraksi joka käsittelee älykkään toiminnan mallintamista tietokoneilla* (Artificial intelligence 2022a), American Heritage dictionary *tietokoneen tai muun laitteen kyvyksi suorittaa toimintoja joita normaalisti pidetään älykkyyttä vaativina* (artificial intelligence 2022b), MOT Kielitoimiston sanakirja *tietokoneen toiminnoista, jot-*

ka jäljittelevät ihmiselle tyypillisiä älykkyyttä vaativia toimintoja (tekoäly n.d.). Toisin sanoi tekoäly on käsite, johon kuuluu tekniikat, joissa kone, tietokoneohjelma tai laite vaikuttaa siltä että että se älyäisi jotain. Määritelmä ei riitä tarkoin rajaamaan, mitä tekoälyyn kuuluu ja ei kuulu. Yleisesti käsitteeseen kuuluu esimerkiksi shakkia ja muita ”älypelejä” pelaavat ohjelmat, tietokoneen ohjaamat hahmot videopelissä, kohdennettu markkinointi henkilöstä kerätyn datan perusteella, itsestään ajavat autot ja kohteen tunnistaminen kuvasta.

Aiemmin tekoälyä on pidetty mystisenä asiana. SciFin ja hypen innoittamina tulevaisuuden koneilta odotettiin oikeaa älykkyyttä ja jopa tietoisuutta. Sitä mukaa, kun sovelluksista on tullut todellisuutta, niitä on vähätelty ja sanottu että ”ei tuo ole älyä, se on vain laskentaa”, puhuttiin AI:n talvesta. (Kahn 2002)

Nyt kun tekoäly on ympärillämme ja käytämme sitä arjessamme voimme hyväksyä, että vaikka se on vain laskentaa, se on silti tekoälyä. Tiedostamme mitä jo Lady Lovelace sanoi Turingin (1950) mukaan – kone voi tehdä vain mitä sen käsketään tekemään, se ei voi saada uutta ideaa.

3.1.2 Koneoppiminen

Koneoppimista pidetään tekoälyn osa-alueena. Siinä ohjelman tai koneen saama syöte vaikuttaa siihen millaisen toiminnan, mallin tai funktion se tuottaa. Aiemman määritelmän mukaan shakkia pelaava ohjelma on tekoälyä, mutta jos se pelaa aina saman ohjelman mukaan se ei ole koneoppimista – se ei opi mitään aiemmista peleistä. Jos aikaisemmat pelit vaikuttavat seuraaviin peleihin – kone voi esimerkiksi tallentaa siirron, joka johti sen tappioon ja välttää vastaisuudessa – se ”oppii” pelaamaan paremmin.

Koneoppimisen menetelmät vaihtelevat tilastotieteen mallien laskemisesta tietokoneilla monimutkaisiin hermoston toimintaa matkiviin malleihin, joiden toteuttaminen missään muodossa ei ole ollut käytännössä mahdollista ennen tehokkaita tietokoneita.

3.2 Regressio

Yksinkertaisessa lineaarisessa regressiossa pyritään löytämään muuttujien $x_1 ... x_n$ lineaariselle funktiolle,

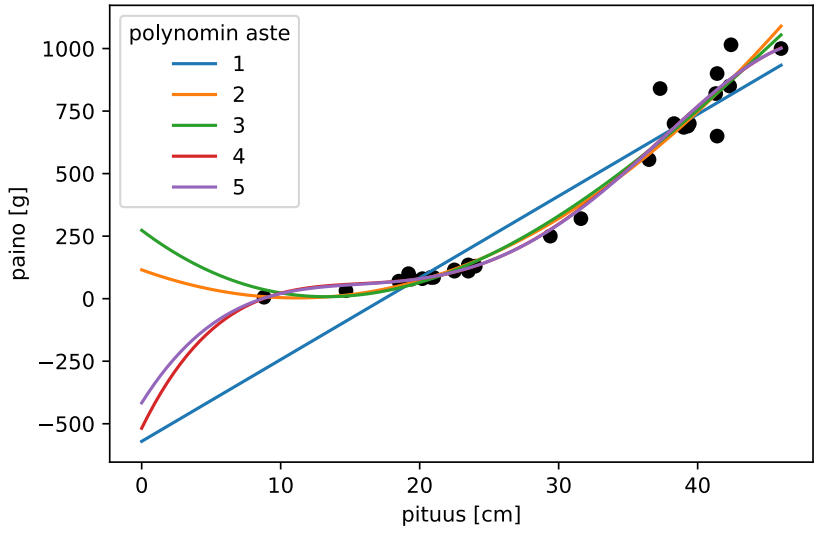
$$\hat{y}(w, x) = w_0 + w_1 x_1 + ... + w_p x_p \tag{19}$$

vakio w_0 ja kertoimet $w_1, w_2 ... w_p$ siten, että keskimääräinen virhe funktion arvon ja opetusdatan välillä on mahdollisimman pieni. Virheen mittarina käytetään tyypillisesti keskihajontaa, mutta jotain muutakin voi käyttää. Tämä menetelmä on kuitenkin rajallinen ja edellyttää melko hyvin lineaarisesti korreloivia muuttujia. Mallia voi pyrkiä täsmentämään korvaamalla pelkkä muuttuja x funktiolla $\phi(x)$. (Bishop 2006)

Polynomisessa regressiossa pyritään löytämään polynomille parhaat kertoimet. Kuviossa 5 näemme täplinä ahvenia[l] koordinaatistolla pituuden ja painon mukaan. Käyrissä näemme polynomisen regression kuvaajan polynomin asteen ollessa yhdestä viiteen, kun polynomille

$$y(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n, \tag{20}$$

jossa y on kalan paino, x on pituus ja n polynomin aste. Ensimmäinen aste on lineaarinen, eli kuten kaava 19 yhdellä muuttujalla. Huomaamme, että lineaarinen malli ennustaa pienille kaloille negatii-



Kuvio 5: Regressio: kalat

[l] Kalat ovat ote kagglen datasetistä *Fish Market* https://www.kaggle.com/datasets/aungpyaeap/fish-market

vistä painoa. Toisen asteen polynomi näyttäisivät olevan tässä tapauksessa paras, kun suuremmilla asteilla malli menee **ylioppimisen** puolelle.

Regressioanalyysin matematiikkaa löytyy mm. Bishopin (2006, luku 3) ilmaiseksi netistä ladattavassa kirjassa.

3.3 Syväoppiminen

Näissä edellisissä malleissa oppiminen on matalaa – mallille annetaan dataa ja ennalta määrätyn kaavan mukaan muodostetaan regressio tai luokittelu ja parametrit ja muuttujat ovat näkyviä. Kun mallille annetaan lisää parametreja, kokonainen verkosto niitä, data ja tulokset muokkaavat parametreja, parametrit muokkaavat toisiaan, ja toiminta tapahtuu ”piilossa” käyttäjältä, oppiminen on syvää.

3.3.1 Johdatus

Kärpäsen aivoissa on aivosoluja noin sata tuhatta (Servick 2018). Tämä on miljoonasosa siitä mitä ihmisellä, mutta silti kärpäsellä on monia taitoja – se löytää syötävää, osaa väistää vaaroja, puhdistaa itseään, löytää kumppanin, lisääntyä ja lentää. Jos pystyisimme mallintamaan pieniäkin aivoja, voisimme saada aikaan vaikka mitä.

Ja mehän pystymme. Keinotekoiset neuroverkot ovat tulleet jäädäkseen myös omaan arkeemme. Voimme kuvata kukkaa kännykän kameralla ja Google Lens kertoo meille mikä kasvi on kyseessä. BirdNet-sovellus tunnistaa lintulajin laulun perusteella. BirdNet:in neuroverkossa on parametrejä, noita neuroneita ja synapseja vastaavia olioita noin 27 miljoonaa – vähemmän kuin kärpäsellä, mutta hyvästä ääninäytteestä tunnistaa 984 lintulajia melko suurella varmuudella, näytteen spektrogrammin perusteella (Kahl, Wood, Eibl & Klinck 2021).

Keinotekoiset neuroverkot ovat laskentamalleja, jotka pyrkivät mallintamaan hermostoja, kuten aivoja, joskin karkeasti. Niissä muodostetaan verkosto, joissa suhteellisen yksinkertaisia laskutoimituksia tekevät, toisiinsa yhdistetyt yksiköt toimivat kuin neuronit ja synapsit aivoissa.

Perinteisessä ohjelmoinnissa ihminen, ohjelmoija, pyrkii miettimään miten ongelma ratkaistaan aritmeettis-loogisesti ja koodaamaan ratkaisun ohjeiksi koneelle. Neuroverkko sen sijaan pyrkii luomaan ratkaisun itse. Neuroverkko ei vain opi, se syväoppii. Lopulta käy niin kuin aivojen kanssa käy, neuroverkko toteuttaa toiminnon, mutta kukaan ei oikein tiedä miten. Se onkin neuroverkon etu. Jos ongelmaan on selkeä looginen, deterministinen ratkaisutapa, käytettäköön niitä. Mutta jos ratkaisua ei ihan tiedetä, tai siitä tulisi järjettömän monimutkainen, ovat syväoppivat neuroverkot paikallaan.

3.3.2 Neuroverkon rakenne ja toiminta

Neuroverkko koostuu toisiinsa yhteydessä olevista yksiköistä eli **neuroneista** ja niiden välisistä yhteyksistä eli **synapseista**. Jokainen neuroni on yhteydessä jokaiseen seuraavan kerroksen neuroniin synapsin välityksellä. Kullakin synapsilla on oma **paino**, kerroin jolla syöttävän synapsin antama arvo kerrotaan. Kunkin neuronin saama arvo on siihen edellisestä kerroksesta synapsien välityksellä saatujen signaalien painotettu summa. Neuroneilla on **aktivaatiofunktio**, joka määrittää minkä arvon se antaa seuraavalle kerroksille johtaville synapseille. Neuroverkon ensimmäinen kerros on **syötekerros**, jolle annetaan sitä dataa, jota halutaan prosessoida. Neuroverkon laskennan tulos, eli **ennuste** luetaan sen viimeiseltä kerrokselta, eli **ulostulokerrokselta**. Näiden välissä olevat kerrokset ovat **piilokerroksia**. (Kelleher 2020, luku 3)

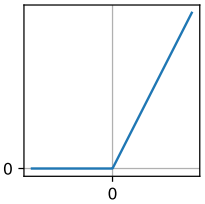
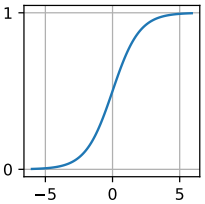
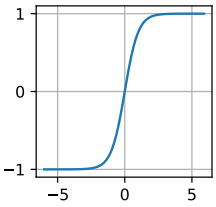
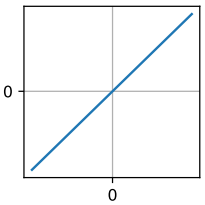
Aktivaatiofunktioista (taulukko 2), jotka siis määrittävät neuronin seuraavalle kerrokselle antaman arvon, on yleisessä käytössä muun muassa ReLU (rectified linear, positiivinen lineaarinen, ”sarana”), sigmoid (logistinen), tanh ja lineaarinen. Nykyään yleisin näistä on ReLU. Yksinkertaisena sen laskenta on nopeampaa kuin esimerkiksi sigmoidin, ja sillä on todettu saatavan hyviä tuloksia (Kelleher 2020, 72).

Luokittelevan neuroverkon ulostulokerroksella käytetään yleensä **softmax**-funktiota

$$\phi(x) = \frac{e^x}{\sum_j e^x}, \tag{21}$$

jossa $\sum_j e^x$ on kerroksen neuroneneiden e^x summa. Tällöin neuroneiden ulostulojen summa on 1 ja arvot kertovat kategorian todennäköisyyden. (Bishop 2006, 209; tf.keras.activations.softmax 2022)

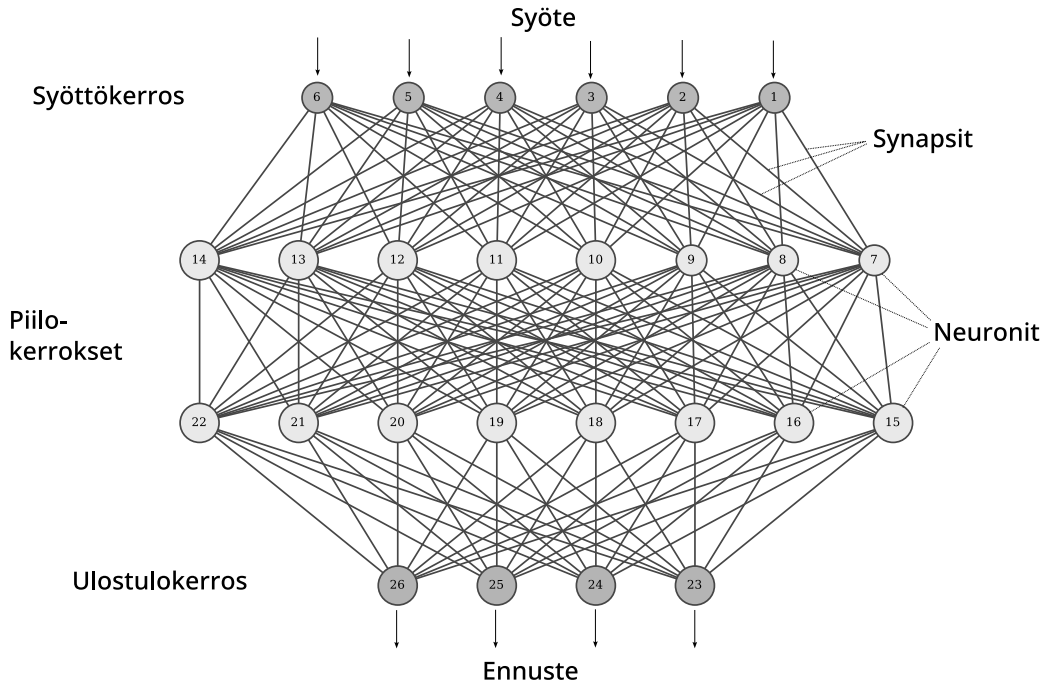
Taulukko 2: Aktivaatiofunktioita

	$\phi(x) =$	kuvaaja
ReLU	$\begin{cases} x & : x \geq 0 \\ 0 & : x < 0 \end{cases}$	
sigmoid	$\frac{1}{1+e^{-x}}$	
tanh	$\frac{2}{1+e^{-2x}} - 1$	
lineaarinen	x	

Esimerkkineuroverkossa (kuvio 6) on syötekerros, jossa on 6 sisääntuloyksikköä, kaksi piilotettua kerrosta joissa 8 neuronia kussakin, ja neljän neuronin ulostulokerros.

Merkitään neuroneiden antamia arvoja X_n, neuronien vakiotermejä b_m ja synapsien painoja W_{n_m} sekä neuroneiden aktivaatiofunktioita $\phi_n()$, joissa n on lähettävän ja m vastaanottava neuroni. Esimerkiksi neuronin 7 saama arvo on

$$\sum_{n=1}^{6} X_n \cdot W_{n\_7} + b_7. \tag{22}$$



Kuvio 6: Neuroverkko

Kun tämä syötetään vielä neuronin aktivaatiofunktiolle, neuronin seuraavalle kerrokselle antava arvo, eli

$$X_7 = \phi_7(\sum_{n=1}^{6} X_n \cdot W_{n\_7} + b_7). \tag{23}$$

Samoin voidaan laskea ensimmäisen piilokerroksen muiden neuroneiden arvot, ja näiden ollessa tiedossa edelleen seuraavan kerroksen arvot. (Koko kerroksen laskenta voidaan esittää myös matriisien pistetulona.)

Kun neuroverkko on kouluttamaton, sen antamat ennusteet ovat mitä sattuu. Uudet verkot ovat yleensä alustettuja satunnaisluvuilla. Siksi verkko koulutetaan datalla, jonka oikeat tulokset ovat tiedossa. Tulosten perusteella lasketaan kuinka paljon neuroverkko oli väärässä ja mihin suuntaan synapsien painoarvoa pitäisi muuttaa, jotta tulos olisi parempi. Tätä kutsutaan vastavirtaan syöttämiseksi.

3.3.3 Vastavirtaan syöttö

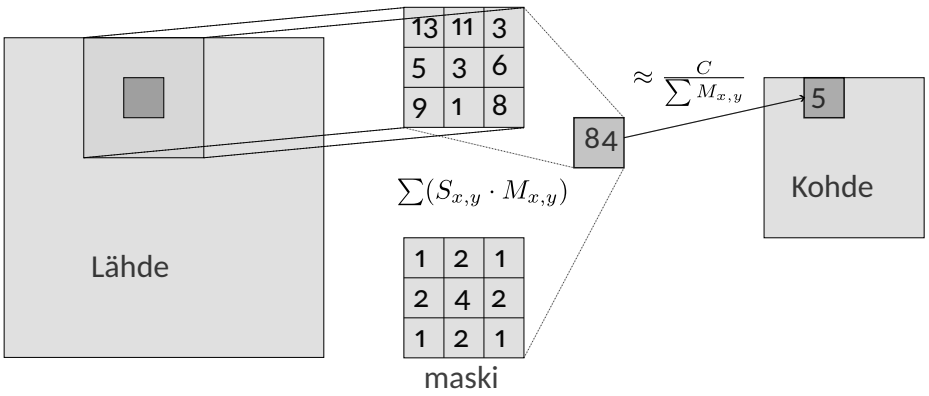
Vastavirtaan syötössä pyritään säätämään parametreja (painoja ja vakiotermejä) takaisinpäin edeten siten että verkon tuottama ennuste paranisi. Ennusteelle lasketaan virhe, eli `hävikki`. Hävikki voi olla ulostulojen keskihajonta, varianssi tai muu funktio – pääasia on, että mitä pienempi se on, sen parempi. Jokaiselle alimpaan kerrokseen johtavalle synapsille lasketaan virhegradientti δ, jo-

ka kertoo mihin suuntaan sitä pitäisi muuttaa, että hävikki pienenisi, esimerkiksi osittaisderivaatan avulla. Sitten jokaisen synapsin painoarvoa muutetaan hieman tähän suuntaan.

Seuraavan ylemmän kerroksen synapsien virhegradientit lasketaan summaamalla siihen yhteydessä olevien neuronien δ:t painotettuna niihin johtavien synapsien painoilla. Seuraavan kerroksen synapsit voidaan nyt säätää ja sitten taas laskea ylemmän kerroksen gradientit. Tätä jatketaan, kunnes päästään syöttökerrokselle. (Kelleher 2020, luku 6)

3.3.4 Konvoluutio

Erityisesti kuvia käytettäessä syötteenä käytetään konvoluutiota. Konvoluutiolla pyritään etsimään datasta piirteitä samalla vähentäen datan määrää. Konvoluutiota hyödyntäviä neuroverkkoja sanotaankin konvoluutioneuroverkoiksi.



Kuvio 7: Konvoluutio

Konvoluutiossa otetaan kuvasta maskin kokoinen alue. Alueen pikselit kerrotaan maskin vastaavilla arvoilla ja summataan. Tämä jaetaan maskin arvojen summalla. Tulos on siis kertomien painotettu keskiarvo. Maskattavaa aluetta siirretään yhden (jolloin kohteesta tulee saman kokoinen kuin lähde) tai useamman pikselin verran (jolloin kohdekuvan resoluutio pienenee). Maskin arvot vaikuttavat siihen, millaisia muunnoksia konvoluutio aiheuttaa. (Pound 2015, 2016)

Esimerkissä (kuvio 7) maskin koko on 3x3 ja siirtymä 2x2 pikseliä. Lähde on 8x8 pikselinen 16 bittinen kuva ja kohteesta tulee 4x4 pikselinen.

3.3.5 Käsitteitä

Suomenkieliset termit tässä ovat K. Pietiläisen käännöksestä Kelleherin (2020) kirjaan. Termit suomeksi eivät välttämättä ole vakiintuneet. Taulukossa 3 on englanninkieliset vastineet. Samalla esitellään joitakin käsitteitä, joita ei ole edellä mainittu.

Taulukko 3: Termistöä

suomi	englanti	kuvaus
vastavirtaan syöttö	backpropagation	
hävikki	loss	
ennuste	prediction	koneoppimismallin antama tulos syöttestä
oppimisnopeus	learning rate	painojen säätökerroin vastavirtaan syötössä
ylioppiminen	overfitting	malli noudattaa opetusdataa liian tarkasti
epookki	epoch	jakso, jossa opetusdata on syötetty mallille kertaalleen

4 Suunnitelma

4.1 Työkalut

Ohjelmointi suunniteltiin toteutettavaksi pääsääntöisesti Python-ohjelmointikielellä ja sen datankäsittelyyn, signaalinkäsittelyyn ja koneoppimiseen soveltuvilla kirjastoilla, kuten Pandas, NumPy, SciPy ja TensorFlow.

Python on viime vuosina kohonnut suosituimmaksi tulkattavaksi ohjelmointikieleksi. Syynä tähän lienee sen helppo omaksuttavuus ja monipuolinen valikoima kirjastoja monenlaiseen käyttöön. Pythonin huonoja puolia ovat hitaus ja puutteellinen moniajomahdollisuus. Useimpien kirjastojen rutiinit ovat kuitenkin käännettävällä kielellä kirjoitettuja ja konekielisiä, joten niiden suoritus oletettiin riittävän nopeaksi. Joitain rutiineja pidettiin mahdollisena kirjoittaa käännettävällä ohjelmointikielellä, jos niiden toteutus Pythonilla osoittautuu liian hitaaksi.

Pandas on tehokas datankäsittelykirjasto. Sen keskeinen tietotyyppi on DataFrame, taulukko, jossa voi olla useita tietotyyppejä. Pandasissa on monipuolisesti toimintoja datan muokkaukseen ja valintaan sekä tilastolliseen laskentaan.

NumPy tarjoaa tietorakenteita, datatyyppejä ja funktioita matemaattiseen laskentaan.

SciPy on tieteelliseen laskentaan kehitetty kirjasto. Sen sisältää toimintoja muun muassa signaalinkäsittelyyn ja Fourier-muunnokseen.

TensorFlow on syväoppimiskirjasto neuroverkkomallien luomiseen, kouluttamiseen ja ajamiseen. Sen kehittäjä on Google, joka julkaisi sen avoimena lähdekoodina vuonna 2015 (Metz 2015). TensorFlown Python versio sisältää mallien koostamista helpottavan *Keras*in. TensorFlow tukee myös nVidian CUDA ja AMD:n ROCm -rajapintoja grafiikkasuorittimien (GPU) käyttämiseksi neuroverkkoja kouluttaessa, jolloin suoritus on moninkertaisesti nopeampaa, kuin keskusprosessoria (CPU) käytettäessä.

4.2 Datan käsittely

Datasta aiottiin muodostaa ainakin Fourier-analyysi, spektrogrammi ja autokorrelaatio eri aika-arvoilla syötteeksi koneoppimismalleille. Fourier-analyysistä tallennetaan syötteiksi lista näytteen voimakkaimmista taajuuksista ja niiden vastaavat amplitudit. Tätä varten kiihtyvyysdata on muutettava nopeusdataksi integroimalla.

JSON-tiedostoista suunniteltiin koostettavaksi taulukko, johon kootaan relevantti informaatio näytteistä. Tätä taulukkoa käytetään perustana näytteiden jatkokäsittelylle ja koulutettavan datasetin valinnalle.

4.3 Neuroverkot ja niiden kouluttaminen

Neuroverkkomalleja aiottiin sekä koostaa itse, että käyttää TensorFlown valmiita malleja. Valmiita malleja tutkimalla voidaan myös löytää piirteitä, joita voi hyödyntää omissa malleissa. Parhaita malleja aiottiin löytää ja muodostaa kokeilemalla eri malleja, kerroksia ja aktivaatiofunktioita sekä analysoimalla niiden tuloksia. Myös koulutuksen edistymistä suunniteltiin seurattavan, jotta voidaan arvioida sopiva koulutuksen määrä, oppimisnopeus ja mahdollinen ylioppiminen.

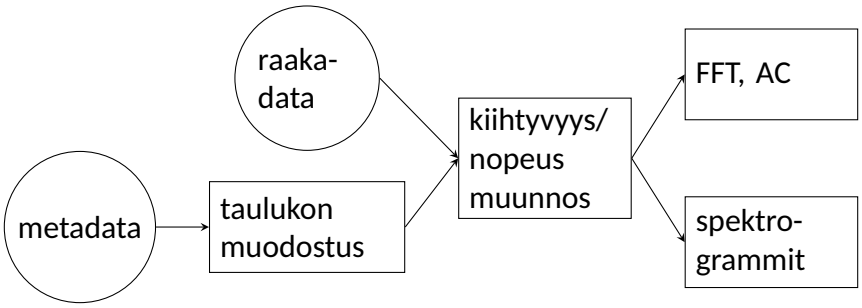
5 Toteutus

5.1 Datan muoto

Lähdedatana oli 88768 näytettä laitteiden värinästä. Laitteet olivat sähkömoottoreita, puhaltimia, pumppuja ja muita laitteita. Värinä oli koodattu 24 bittiseksi pulssikoodimodulaatioksi, joka oli tallennettu luetteloksi kymmenjärjestelmän lukuja tekstitiedostoon. Tiedostot oli jaoteltu useisiin alihakemistoihin.

Jokaista näytettä vastaa JSON-tyyppinen tiedosto, joka sisältää tietoja näytteestä: minkä tyyppinen kone on kyseessä, mihin anturi on kiinnitetty sekä datan parametreja kuten näytteenottotaajuus ja ennustettava suure, eli laitteen käyntinopeus (RPM).

5.2 Esikäsittely



Kuvio 8: Esikäsittelyn työnkulku

5.2.1 Näytelistan koostaminen

Metadatatiedostoissa olevan datan koostamiseksi taulukkoon tehtiin ohjelma, joka käy rekursiivisesti läpi kaikki datahakemistot ja kerää JSON-muotoisista metadatatiedostoista tarpeellisiksi tiedetyt ja mahdollisesti käytettävät tiedot Pandasin taulukkoon, eli *DataFrame*een. Koneen tyyppi (sähkömoottori, tuuletin tai pumppu) haettiin saadusta referenssitaulukosta. Listasta poistettiin näytteet, jotka oli merkitty virheelliseksi, lyhyet (alle puoli sekuntia pitkät) ja koneet jotka eivät ollee käynnissä lainkaan (RPM = 0). Karsinnan jälkeen näytteitä jäi taulukkoon 79055. DataFrame talletettiin levylle Apache Parquet –muodossa. Talletetut tiedot esitetty taulukossa 4. Taulukon generoiva ohjelma liitteessä 1.

Taulukko 4: DataFrameen talletetut tiedot

nimi	selite
terminal	laitteen sarjanumero
datafile	raakadatatiedoston nimi
metadata	metadatatiedoston nimi
directory	hakemisto
samplecount	kuinka monta lukua (PCM näytettä) näytteessä on
mVscaling	kerroin lukeman muuttamiseksi jännitteeksi
frequency	PCM näytteenottotaajuus hertseinä
RPM	käyntinopeus kierroksina minuutissa
meas	koneen tyyppiin ja anturin kiinnityspaikkaan viittaava tunniste
time	näytteenoton aika unix-aikana
type	koneen tyyppi

5.2.2 Muunnos nopeusdataksi

Antureiden kiihtyvyysdatasta saadaan muunnos nopeusdataksi integroimalla (Wren 2010). Ennen käsittelyä datasta on syytä vähentää sen keskiarvo, jolloin sen arvo liikkuu tasaisesi nollan molemmin puolin ja muunnoksen arvot eivät ”karkaa” suuriin lukuihin (Aumala ym. 1998, 130). Muunnokseen käytettiin SpiPyn funktiota `scipy.integrate.cumulative_trapezoid`. Muunnokset talletettiin sarjoina 32 bittisiä liukulukuja Parquet-tiedostoihin. Muunnettu data on mielivaltaista yksikköä, kuitenkin keskenään yhdenmukaista. Muunnokset tekevä Python-ohjelma on liitteessä 2.

5.2.3 Spektrogrammien muodostaminen

Nopeusdatasta muodostettiin spektrogrammit käyttämällä Matplotlibin `specgram`-funktiota ja talletettiin kuvina PNG-muodossa, joka muunnettiin harmaasävykuvaksi tallennustilan säästämiseksi[II]. Kuvien resoluutioksi tuli 334 kertaa 217 pikseliä. Resoluutioon ei pysty vaikuttamaan Matplotlibin, eikä minkään muunkaan kokeillun spectrogrammeja luovan ohjelman tai tai moduulin asetuksissa. Matplotlibillä on mahdollista tuottaa eri resoluutioinen kuva, mutta silloin se on skaalattu mainitusta, eikä täten sisällä enempää informaatiota. Tämä huomattiin, kun kokeiltiin säätää kuvan kooksi 100x1000 ja tarkasteltiin tulosta.

Alussa spektrogrammit oli tehty kiihtyvyysdatasta virheellisesti. Joskus käy kuitenkin niinkin, että virhe osoittautuu hyödylliseksi, kuten myöhemmin luvussa 6.1.2 kerrotaan.

[II] Informaatiota ei arvioitu tässä menetettävän, värikuva on vain ”mukavampi katsella”

5.2.4 Ominaisuuksien erottaminen

Nopeusdata normalisoitiin välille -1...1, suodatettiin pois hyvin matalat taajuudet (< 10 Hz) lisättiin Blackman-ikkuna ja tehtiin nopea Fourier-analyysi (DFT). DFT:n tulos on kompleksinen. Se muutettiin reaaliluvuiksi ottamalla itseisarvo. Realimuotoisesta DFT:stä etsittiin huippukohdat, joita aluksi löytyi liikaa. Vierekkäisten huippujen tallentumisen välttämiseksi ajettiin DFT-datalle alipäästösuodatus (mikä ei tässä tarkoita mitään todellista, koska tässä vaiheessa data ei ole aikamuotoista, vaan taajuusmuotoista). Näin saatiin pois datan rippeliä ja samojen huippujen ”löytymistä” useampaan kertaan. 64 voimakkaimman huipun taajuudet ja niiden vastaavat amplitudit tallennettiin DataFrameen. Jos näytteestä löytyi tätä pienempi määrä huippuja, talletettiin kohtaan puuttuvaa tarkoittava NaN.

Autokorrelaatio otettiin datasta väleillä 1...16384, joten yläraja on noin kolmasosasekunti näytteenottotaajuuden ollessa noin 48 kHz. Autokorrelaatiosta etsittiin myös huiput, jotka talletettiin voimakkuusjärjestyksessä. DFT:n laskemiseen, suodattimien toteuttamiseen ja huippujen löytämiseen käytettiin SciPyn funktioita, autokorrelaatioon puolestaan Statsmodelsin `acf`-funktiota. Suurin amplitudi (point to point) ja datan varianssi (suhteellinen teho) talletettiin myös. Nämä toiminnot toteuttava ohjelma on liitteessä 3.

5.2.5 Jako opetus-, validointi- ja testidataan

Data jaettiin kolmeen osaan:

- Opetusdataan (70% näytteistä), jota käytetään neuroverkon varsinaiseen koulutukseen, jota siis syötetään verkolle ja jonka perusteella oppiminen tapahtuu.
- Validointidataan (15%), jota käytetään opetuksen aikana seuraamaan mallin laatua. Tätä seurantaa käytetään päätökseen siitä, jatketaanko mallin opetusta tai pienennetäänkö sen oppimisnopeutta.
- Testidataan (15%), jolla testataan mallia opetuksen jälkeen.

Testi- ja validointidatana voidaan käyttää myös samaa dataa, mutta autenttisuuden vuoksi haluttiin käyttää testaamiseen dataa, jota malli ei ole opetuksen aikana lainkaan ”nähnyt”. Jako tehtiin käyttämällä `sklearn`-kirjaston `train_test_split`-funktiota. Käytännössä jako toteutetaan aina

uudestaan luvussa 5.3 esitellyssä ohjelmassa. Toteutunut jako on sama joka kerralla käytettäessä samaa satunnaisjuurta funktiossa, edellyttäen, että näytteiden lista ei muutu.

5.3 Opetusohjelma

Muutamien ensimmäisten mallien ajon jälkeen havaittiin, että projektista tulee melko sotkuinen, jos ottaa samasta mallin luovasta ja kouluttavasta ohjelmasta kopioita eri malleja ja koulutettavaa dataa varten. Päätettiin luoda Python-ohjelma, joka ajaa kaikkien mallien koulutuksen ja mallit olisivat erillisinä moduuleina. Ohjelma oli kehityksen alaisena lähes koko projektin ajan ja siihen lisättiin myös evaluointitoimintoja. Ohjelma sai nimekseen `trainer-general.py` (liite 5).

Trainer-general ottaa komentorivin argumentteina koulutettavan mallin, halutun datan (meas tai laitteen tyyppi) ja ajettavien epookkien määrän. Sillä voi myös listata saatavilla olevat neuroverkkomallit, meas:it tai tyypit ja ajaa halutessaan pelkän evaluaatiofunktion. Neuroverkkomallit olivat omassa hakemistossaan moduuleina, jotka sisälsivät tiedon siitä, mitä dataa se ottaa syötteenä. Moduulien lataamisen dynaamisesti siten, että sen nimeä ei tarvise ”kovakoodata” (`import` käskyn argumenttina) ohjelmaan mahdollistaa `importlib`-moduuli.

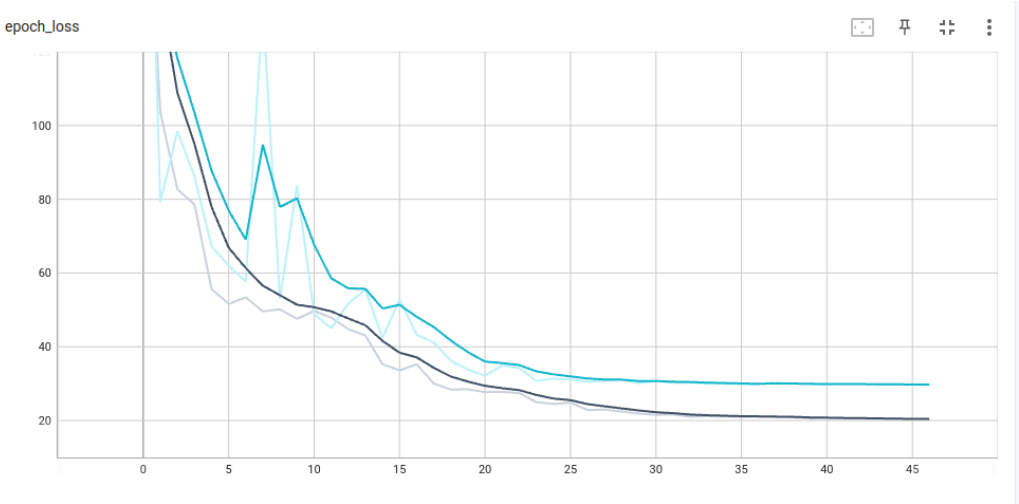
5.4 Takaisinkutsut

Takaisinkutsut (Callbacks) ovat työkaluja, joita ajetaan epookkien välillä. TensorFlowssa nämä löytyvät kirjaston `tf.keras.callbacks`-kirjastosta. Näistä käytettiin seuraavia:

- *EarlyStopping* pysäyttää opetuksen, kun hävikki on lakannut paranemasta. Tällöin opetusta käynnistäessä voi antaa ajettavien epookkien lukumääräksi reilusti suuremman luvun, kuin mikä arvioidaan tarpeelliseksi. Samalla vältetään ylioppimista.
- *ReduceLROnPlateau* pienentää oppimisnopeutta, kun validointihävikki ei ole enää parantunut. Näin opetus voidaan aloittaa suurehkolla oppimisnopeudella, jolloin oppimisen alkuvaihe tapahtuu nopeammin. Oppimisnopeuden ollessa liian suuri hävikki alkaa usein sahaamaan edestakaisin, kun verkon parametreja säädetään suurin askelin. Nopeutta pienennettäessä asteittain alkaa hävikki tasaantua siihen, mikä kyseisellä mallilla ja datalla on saavutettavissa.

- *TensorBoard* tallentaa tietoja oppimisen edistymistä. Tietoja luetaan erillisellä TensorBoard -ohjelmalla, jonka käyttöliittymää käytetään WWW-selaimella. Tietoja voi tarkastella niin opetuksen aikana, kuin sen jälkeenkin. Tensorboardista kaapatussa kuviossa 9 havainnollistuu hävikin heilahtelu ennen kuin ReduceLROnPlateau vähentää oppimisnopeutta.

ReduceLROnPlateauta ja EarlyStoppingia voi käyttää samanaikaisesti, kun asetetaan jälkimmäisen ”kärsivällisyys” (patience) – optiota, kuinka monta epookkia funktio odottaa ennen aktivoitumista – suuremmaksi. ReduceLROnPlateaun kärsivällisyytenä käytettiin arvoa 2 ja EarlyStoppingin aluksi 10. Huomattiin, että useimmissa tapauksessa EarlyStopping toimii, mutta joskus opetus jatkui loputtomiin, vaikket hävikki parantunut. Pienentämällä oppimisnopeuden vähentäjän minimiarvoa, sekä pienentämällä EarlyStoppingin kärsivällisyyttä ja asettamalla sille vähimmäisdelta-arvon, tilanne parani.



Kuvio 9: Kuvakaappaus TensorBoardin oppimiskäyrästä

5.5 Syötteiden syöttö

Spektrogrammien syöttämiseen malleille käytettiin Kerasin `ImageDataGenerator`-generaattoria optiolla `rescale=1./255`, eli pikseleiden arvot skaalattiin liukuluvuksi välille 0…1. Tästä muodostettiin erilliset generaattorit opetus-, validointi- ja testidatalle. FFT ja autokorrelaatiot voitiin antaa mallille suoraan DataFramesta.

Järjestelmässä mallin sisältävä moduuli määrittää sen, mitä syötteitä – spektrogrammi, Fourier-analyysin tulokset, autokorrelaatiomaksimit tai signaalin voimakkuutta indikoivat luvut – `trainer-`

`general` niille antaa. Muita syötteitä kuin spektrogrammi voitiin myös yhdistellä. Spektrogrammit syötetään neuroverkkomallille generaattorilla ja sen yhdistämiseen taulukko-muotoiseen dataan ei ole itsestäänselvää keinoa.

5.6 Omien mallien muodostus

Malleja kehiteltiin vaiheittain kokeilemalla erilaisia muutoksia - lisäämällä kerroksia, muuttamalla niiden parametreja ja aktivaatiofunktiota.

Kokeiluissa havaittiin mm. seuraavaa:

- Tavallisia piilokerroksia (Dense) ei tarvita montaa. Yleensä riittää yksi tai kaksi ennen ulostulokerrosta.
- Spektrogrammi osoittautui selkeästi parhaimmaksi syöttödataksi. Fourier analyysin taajuuksista ja amplitudeista, tai autokorrelaation huippukohdista neuroverkko ei oppinut ”ymmärtämään” erityisen hyvin niiden yhteyttä käyntinopeuteen.
- Spektrogrammin ollessa syötteenä konvoluutiokerrokset olivat erittäin hyödyllisiä.
- Maxpooling-kerrokset eivät parantaneet tuloksia tässä tapauksessa.
- Dropout-kerroksia käytetään välttämään mallin ylioppimista. Huomattiin, että vaikka sen käyttö kaventaa opetus- ja testidatan hävikkien eroa, se ei kuitenkaan yleensä johtanut parempaan testitulokseen. Dropouttia ei käytetty säännönmukaisesti.

5.7 Valmiiden mallien soveltaminen

Kerasin `applications`-kirjastossa on esikoulutettuja malleja, joita testattiin. Useat näistä malleista ovat huomattavan suuria – niissä on kerroksia jopa satoja ja parametreja (painoja ja vakiotermiarvoja) kymmeniä miljoonia (Keras Applications nd.). Suuremmat osoittautuivat käytettävän laitteiston kuuden gigatavun näyttömuistille liian suuriksi. CSC:n tai AWS:n palveluissa olisi ollut ehkä mahdollisuutta saada käyttöön suurempi kapasiteetti. Tähän ei kuitenkaan ryhdytty, koska katsottiin että muutakin kokeiltavaa riittää.

Valmiit mallit ovat suunniteltu etupäässä kuvien kategorisointiin. Niiden ulostulokerroksella on siten useita neuroneita ja aktivaatiofunktiona `softmax`. Ulostulokerros poistettiin ja lisättiin tilalle yksineuroninen lineaarinen kerros ja mahdollisesti jokin kerros ennen sitä.

5.8 Lineaarinen regressio

Yksinkertaisin koneoppimismalli käyntinopeuden ennustamiseen on Lineaarinen regressio. Häiriöttömistä signaaleista yhdenmukaisista lähteistä se olisi todennäköisesti luotettava malli. Seteistä laskettiin myös vertailukohdaksi lineaarinen regressio käyttämällä syötteenä Fourier-analyysin voimakkainta taajuutta. Malli toteutettiin Sklearnin `LinearRegression`-moduulilla.

5.9 Käytetty ohjelmisto

Taulukossa 5 on listattuna ohjelmistoja, jotka olivat olennaisia työn toteutuksessa. Lista ei ole kattava. Kaikki käytetyt sovellukset, kirjastot ja työkalut ovat avoimen lähdekoodin ohjelmistoja.

Taulukko 5: Käytetyt ohjelmistot

ohjelmisto	versio	tyyppi	tarkoitus
Manjaro Linux	21.2.6	käyttöjärjestelmä	työskentely
Python	3.10.4	tulkattava ohjelmointikieli	ohjelmointi
TensorFlow	2.8.0	syväoppimiskirjasto	neuroverkot
SciPy	1.8.0	kirjasto tieteelliseen laskentaan	signaalin matemaattinen analyysi
Matplotlib	3.5.2	graafikirjasto	visuaalinen havainnollistaminen
Pandas	1.4.2	datankäsittelykirjasto	datan käsittely
Scikit-learn	1.1.2	koneoppimiskirjasto	regressio, datan jako

6 Tulokset

Taulukoissa 6 ja 7 esitetyt testidatalla tehdyn ennusteen keskihajonnat on tehty valikoidulla joukolla neuroverkkomalleja. Mallin nimestä voi päätellä verkon koostumuksesta: ennen alaviivaa syötteiden tyyppi selittyy siten, että 'a' merkitsee autokorrelaatiota, 'f' Fourier-analyysin tuloksia, 'p' signaalin suurinta amplitudia ja varianssia, sekä 's' spekrogrammia. Useampi kirjain tässä merkitsee siis useamman syötteen mallia. Alaviivan jälkeen kerrokset ja niiden lukumäärä selittyy seuraavasti: 'd' on yhtä kuin `Dense`, eli tavallinen piilokerros, 'c' konvoluutiokerros, 'm' `MaxPooling` ja 'n' normalisointi. Esimerkiksi mallissa s_d1c2n2 on syötteenä spektrogrammi, yksi Dense-kerros sekä konvoluutio- ja normalisointikerroksia kaksi. 'Mobile' ja 'efficient' ovat Kerasin valmiita malleja: Mobilenet ja EfficientNet. Lopussa mahdollisesti oleva kirjain tarkoittaa eri versiota mallista (jotain on muutettu, kuten neuronien määrää tai aktivointifunktiota). 'LR' on lineaarinen regressio. Jos tulos on "NaN", ei ajossa ole saatu tulosta, vaan se on mahdollisesti kaatunut virheeseen, kuten muistin loppumiseen. 's_mobi-accel'-mallissa on käytetty kiihtyvyysdatasta tehtyjä spektrogrammeja MobileNet-mallissa. Paras tulos kullekin datasetille on merkitty tummennettuna.

Kunkin setin testidatan ennusteen keskihajonnan perusteella parhaat tulokset on esitetty taulukossa 8. Siinä "malli" on neuroverkkomalli, jolla tulos on saatu, "epokkeja" montako epookkia on ajettu ennen kuin EarlyStopping on pysäyttänyt opetuksen[III], ρ_{train} keskihajonta opetusdatalla, ρ_{test} keskihajonta testidatalla, "< 5" kuinka suuri osa ennusteista poikkeaa alle 5 RPM mitatusta ja vastaavasti "10...30", '30...100" ja "> 100" poikkeamia kyseisillä väleillä ja yli 100 RPM. "Paras" sarakkeessa on kaikkein lähimpänä mitattua ollut poikkeama ja "huonoin"-sarakkeessa kaikkein kauimpana ollut.

[III] Opetukset ajettu maksimiarvolla 100 epookkia

Taulukko 6: Ennusteiden keskihajonnat: osa 1

	demo	flakt-de	flakt-nde	motor-de	motor-nde	pump-de
a_d1	538.1	222.2	197.4	282.7	281.2	115.9
a_d2	474.9	213.4	168.6	272.4	269.9	100.4
a_d3	457.1	206.3	170.0	276.0	277.4	99.9
ap_d1	500.7	218.3	188.0	281.2	281.2	104.0
ap_d3	487.5	205.9	164.3	273.1	275.8	97.2
ap_d4	535.3	208.1	168.0	243.9	235.1	100.7
fap_d2	334.5	147.4	163.4	206.0	184.1	37.1
fp_d1	331.3	198.7	215.0	270.2	218.9	33.5
fp_d2	318.8	183.5	198.5	265.4	209.5	34.0
fp_d3	335.1	176.4	192.6	255.5	205.2	39.5
fp_d4	293.8	156.1	198.3	215.8	177.2	37.5
s_d1b	139.1	177.5	137.6	117.5	85.1	35.7
ap_d2	475.5	206.4	177.5	273.0	270.3	99.3
s_d1c1bd1	117.7	110.4	76.0	45.4	42.8	42.5
s_d1c1b	88.3	96.4	85.2	60.9	43.5	36.0
s_d1c1c	149.6	113.0	145.3	75.2	88.8	35.1
s_d1c1d	1117.7	NaN	84.1	55.5	66.4	48.9
s_d1c1m1	82.5	95.5	77.7	51.9	46.8	34.9
s_d1c1n1	78.6	102.2	87.2	41.0	44.9	52.1
s_d1c1	83.5	91.5	56.6	50.7	41.5	34.3
s_d1c2n2	**36.3**	53.7	49.1	34.8	30.5	14.3
s_d1c3n2	1500.1	34.1	31.8	**27.5**	24.6	**14.0**
s_d1	125.5	169.3	139.6	128.9	131.0	35.0
s_d2b	1194.1	320.9	307.3	319.3	310.9	36.3
s_d2c2n2	1443.1	56.8	42.4	33.7	25.7	15.3
s_d2	94.7	153.0	144.3	180.1	127.0	32.4
s_efficient	1116.3	320.9	307.4	NaN	310.9	36.3
s_mobile2	481.6	**27.3**	307.6	319.3	310.8	36.7
s_mobi-accel	1112.6	320.9	**26.3**	320.5	**20.7**	37.3
LR	1192.1	307.6	314.6	325.1	331.1	34.0

Taulukko 7: Ennusteiden keskihajonnat: osa 2

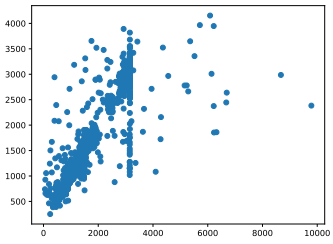
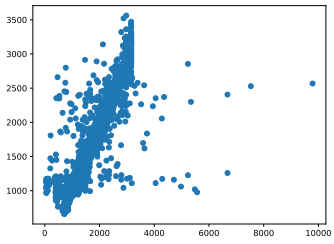
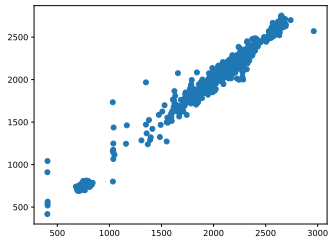
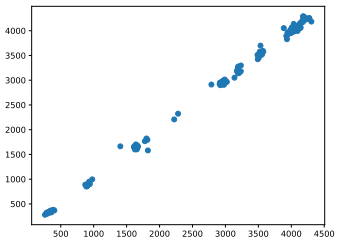
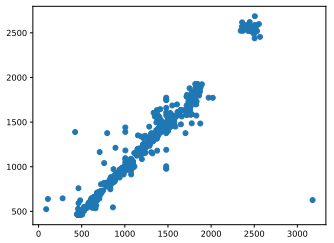
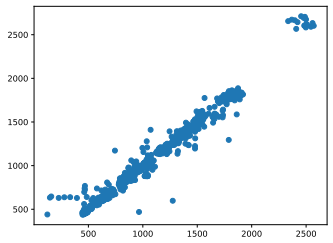
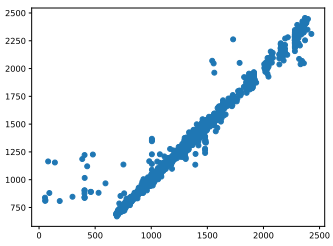
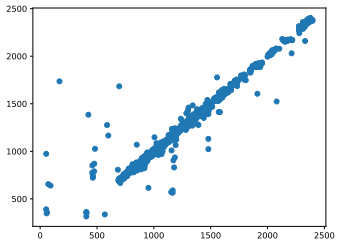
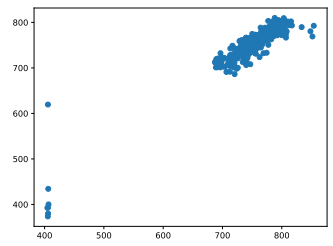
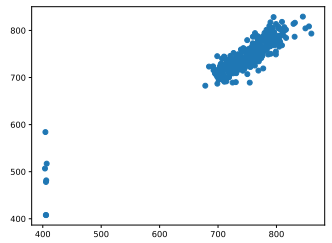
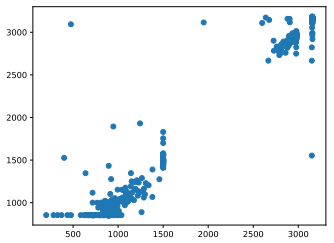
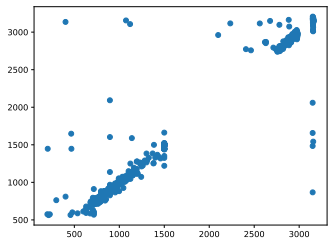
	pump-de-axial	vib0	vib1	vib2	Pump	Blower	Motor
a_d1	39.3	370.3	298.4	398.5	481.7	420.3	462.5
a_d2	35.7	377.7	287.5	383.6	419.7	382.8	446.2
a_d3	38.2	481.9	283.8	393.0	435.3	387.4	440.4
ap_d1	37.8	358.9	321.0	362.9	443.8	406.9	462.3
ap_d3	38.1	382.2	276.2	388.6	422.9	371.6	447.3
ap_d4	37.8	438.1	300.9	279.5	441.8	366.6	422.1
fap_d2	32.6	239.7	239.0	222.6	160.3	337.3	337.8
fp_d1	52.4	322.3	376.6	302.4	213.7	314.9	373.6
fp_d2	54.8	267.1	312.3	258.4	195.0	281.7	335.9
fp_d3	56.9	263.8	284.2	278.3	194.7	283.1	341.7
fp_d4	53.3	155.7	235.6	151.5	171.7	267.4	317.7
s_d1b	34.7	156.7	186.0	151.5	115.0	383.1	283.8
ap_d2	37.4	376.2	285.4	364.8	444.3	384.7	436.6
s_d1c1bd1	36.2	116.7	157.5	127.0	85.7	140.2	94.5
s_d1c1b	37.3	118.0	133.3	135.0	97.1	160.4	102.3
s_d1c1c	35.6	178.3	800.0	152.6	95.8	222.0	168.3
s_d1c1d	NaN	123.8	147.6	152.6	120.3	201.3	121.4
s_d1c1m1	32.7	109.0	116.9	109.9	85.3	154.5	93.5
s_d1c1n1	93.2	118.6	132.9	108.9	66.2	174.1	92.7
s_d1c1	33.6	109.2	110.8	97.3	71.8	167.9	95.8
s_d1c2n2	**16.1**	87.2	96.2	79.6	52.1	113.4	66.0
s_d1c3n2	144.2	76.3	78.2	76.9	**45.8**	**87.8**	59.3
s_d1	699.3	168.4	188.3	152.3	99.5	375.8	276.6
s_d2b	35.9	947.7	800.0	552.5	584.5	495.5	472.2
s_d2c2n2	177.6	81.2	93.5	70.6	88.8	115.4	62.9
s_d2	34.6	156.7	174.9	134.8	96.5	378.7	284.9
s_efficient	36.0	950.1	526.4	376.9	51.3	NaN	472.3
s_mobile2	35.9	103.5	800.3	**59.9**	NaN	NaN	**50.2**
s_mobi-accel	34.2	**70.6**	**55.5**	157.1	NaN	NaN	NaN
LR	36.9	828.9	889.6	561.5	616.4	523.8	506.1

Taulukko 8: Parhaat mallit seteittäin

setti	malli	epookkeja	ρ_{train}	ρ_{test}	< 5	5…10	10…30	30…100	>100	paras	huonoin
Blower	s_d1c3n2	34	28.06	87.75	10.52	11.04	30.62	32.34	10.52	0.02	7402.11
Motor	s_mobile2	44	40.24	50.24	18.06	16.98	42.50	17.37	18.06	0.02	7518.36
Pump	s_d1c3n2	35	36.85	45.80	10.65	10.77	30.72	37.21	10.65	0.11	700.87
demo	s_d1c2n2	58	32.08	36.27	10.69	9.43	35.22	38.36	10.69	0.13	258.87
flakt-de	s_mobile2	33	21.03	27.30	27.67	22.06	33.39	11.21	27.67	0.00	2549.89
flakt-nde	s_mobi-accel	54	23.68	26.30	25.14	20.19	34.24	15.49	25.14	0.01	677.36
motor-de	s_d1c3n2	47	18.98	27.53	21.33	22.04	38.49	15.04	21.33	0.03	1083.46
motor-nde	s_mobi-accel	36	16.58	20.69	30.27	23.77	36.48	7.19	30.27	0.00	1566.07
pump-de	s_d1c3n2	38	9.71	14.00	21.80	25.94	45.86	6.02	21.80	0.09	213.78
pump-de-axial	s_d1c2n2	34	11.29	16.12	21.67	19.20	47.37	10.84	21.67	0.09	179.63
vib0	s_mobi-accel	39	69.76	70.63	9.97	7.62	24.63	41.64	9.97	0.02	2617.86
vib1	s_mobi-accel	43	42.52	55.49	32.56	15.83	26.51	18.15	32.56	0.00	2734.03
vib2	s_mobile2	43	48.20	59.91	29.31	12.32	25.42	24.64	29.31	0.02	2663.66

Taulukossa 9 on kuvattu mallien ennusteiden suhdetta mitattuun arvoon. Kukin piste kuvaa näytettä, sijoittuen X-akselilla mitatun arvon ja y-akselilla ennustetun arvon kohdalle.

Taulukko 9: Mallien ennusteiden kuvaajia

		
Blower s_d1c3n2	Motor s_d1c1bd1	Pump s_d1c3n2
		
demo s_d1c2n2	flakt-de s_mobile2	flakt-nde s_mobi-accel
		
motor-de s_d1c3n2	motor-nde s_mobi-accel	pump-de s_d1c3n2
		
pump-de-axial s_d1c2n2	vibo s_mobi-accel	vib1 s_mobi-accel

6.1 Tulosten analysointi

6.1.1 Syötedatat

Autokorrelaatiomalleilla ei saavutettu kovin hyviä tuloksia. Niillä seteillä, joissa jotkin näistä malleista vaikuttavat suhteellisen hyviltä (pump-de-axial), saavutetaan jo lineaarisella regressiolla saman tasoinen tulos. Toisella matalan lineaarisen regression virheen setillä LR oli huomattavasti parempi. Muilla keskihajonta oli parhaimmillaan noin 31…75 % LR:stä.

Fourier-muunnoksia pidettiin etukäteen olennaisena syöttödatana. Voidaan siten pitää pettymyksenä, etteivät niiden ennusteet olleet tämän parempia. Useimmilla seteillä FFT-mallien tulokset olivat hieman parempia, kuin autokorrelaatiolla. Parhaan AC-mallin ennusteiden keskihajontojen keskiarvo oli 175 RPM, LR:n suhteutettuna vaihteluväli oli 25…141 %.

Edellisten yhdistelmällä ei myöskään saavutettu mitään, merkittävää. Tulokset olivat samaa kokoluokkaa, kuin edellä. Joissain hieman parempi, joissain hieman huonompi.

Spektrogrammi osoittautui ylivoimaisesti parhaaksi syöttödataksi. Virheen keskihajonnaksi saatiin tyypillisesti alle 40 RPM.

6.1.2 Parhaiden ennusteiden ja mallien lähempi tarkastelu

Yhtä mallia, joka olisi paras kaikille seteille ei ollut, neljä eri mallia osoittautui parhaaksi ainakin jossakin setissä. Tosin itse asiassa s_mobi-accel ja s_mobile2 ovat sama malli, mutta eri datalla syötettyjä. Eri koneilla on erilainen värähtely, jolloin niille toimivat erilaiset mallit. Itse tehdyistä malleista parhaiten toimivissa oli vain yksi Dense-kerros ulostulokerroksen lisäksi.

Jokaisessa setissä on ennusteita, jotka poikkeavat mitatusta huomattavasti, jopa tuhansia kierroksia minuutissa. Joitain huonon ennusteen spektrogrammeja tarkasteltiin. Syitä virheeseen voi olla monia. Osassa kuvista näyttäisi olevan lähinnä kohinaa, anturi ei ole ehkä toiminut, osa taas näyttää normaalilta. Jos virheen syy on laitteessa oleva vika, on se tuotantosovelluksen kannalta ongelmallista: vika aiheuttaa virheen, jonka takia taas vikaa ei löydetä.

Demo-setti on simuloitu Distencen laboratoriossa, jotta saataisiin dataa, jossa koneen malli on varmasti sama kaikissa näytteissä. Tämän hajontakuvaaja taulukossa 9 on kaikkein lineaarisin. Tästä voidaan johtaa hypoteesi, että samantyyppisten, mutta ei aivan samanlaisten koneiden joukko aiheuttaa malliin epävarmuutta, joka näkyy ennusteiden hajontana lineaarisen käyrän molemmin puolin. Setin laitteiston samankaltaisuus ilmeisesti parantaa oppimistulosta. Blower- ja Motor-seteissä, joissa on hajanainen laitekanta, hajoaa myös kuvaaja.

Huomattavaa on, että useiden settien kohdalla paras malli oli s_mobi-accel, eli Mobilenet ajettuna kiihtyvyysdatasta tehdyillä spektrogrammeilla. Tämä oli alun perin virhe, muunnos nopeusdataksi oli määrä tehdä ensin. Muiden mallien kohdalla – mukaan lukien muut Kerasin valmiit mallit – kiihtyvyysdata ei toiminut hyvin, vaan nopeusdatalla saatiin huomattavasti parempia tuloksia.

6.2 Menetelmän tarkkuus ja luotettavuus

Joidenkin mallien ja laitekonfiguraatioiden yhdistelmällä päästiin melko hyviin tuloksiin, keskihajonnan ollessa pienimmillään hieman yli 20 RPM. Silti näissäkin on liikaa ennusteita, jotka poikkeavat todellisesta huomattavan paljon. Tämä, ja kuvaaja kielivät siitä, että häiriöiden vaikutusta ei ole täysin onnistuttu saamaan tuloksista pois parhaallakaan yhdistelmällä.

Distencen edustajalta saadun tiedon mukaan datan käyntinopeuslukemissa saattaa olla virhettä, joka on suurimmillaan 30 RPM ja johtuu siitä, että värähtelynäyte ei ole tarkalleen samanaikainen käyntinopeuslukeman kanssa. Virhe toteutuu etenkin kiihtyvässä tai hidastuvassa vaiheessa olevan koneen mittauksissa. Tämä saattaa aiheuttaa malliin epätarkkuutta, mutta ei selitä huomattavia, yli 100 RPM virheitä ennusteissa.

7 Yhteenveto

7.1 Pohdintaa

Työtä aloitettaessa ei voinut tietää millaisia tuloksia saadaan aikaiseksi. Tavoite oli kehittää mahdollisesti jopa tuotantoon sovellettava tuote, mutta toisaalta kysymyksessä oli tutkimus, jossa tutkittiin, voidaanko sellainen tehdä. Vastaus tutkimuskysymykseen lienee, että todennäköisesti kyllä.

Nopeuspektrogrammien toimiminen yhdellä mallilla suhteellisesti erinomaisen hyvin osoittaa, että ei voida ennalta tietää, mikä neuroverkoissa toimii ja mikä ei. Siksi villitkin ideat ovat kokeilemisen arvoisia.

7.2 Jatkokehitysmahdollisuudet

Jatkokehityksessä on syytä tutkia tarkemmin syitä vikaan meneviin ennusteisiin ja pohtia niiden pohjalta parannusmahdollisuuksia. Neuroverkoille annettavista syötteistä voidaan parantaa ainakin spektrogrammin tarkkuutta kasvattamalla. Tämä voitaisiin toteuttaa kehittämällä oma ohjelma tai käyttämällä joitain avoimen lähdekoodin ohjelmaa pohjana. Spektrogrammin tueksi voidaan kokeilla antaa muuta signaalista erotettua dataa syötteeksi.

Syötteestä erotetuksi dataksi voidaan keksiä uusia keinoja, myös sellaisia jotka eivät kuulu perinteisiin signaalinkäsittelyn oppeihin. Ennalta ei tiedä, mitkä piirteet osoittautuvat toimiviksi neuroverkkomallaissa.

Kun koneoppimismallin ennusteesta saadaan riittävän tarkka ja luotettava, on mahdollista tehdä esimerkiksi lähes reaaliajassa toimiva pyörimisnopeuden kertova sovellus. Valmis koulutettu malli ei tarvitse enää huomattavaa laskentatehoa, ja se voidaan siirtää käytettäväksi *TensorFlow Lite*llä myös mobiililaitteeseen tai Raspberry Pi -minitietokoneeseen. Näin voitaisiin toteuttaa vaikka laitekohtainen käyntinopeusmittari.

Lähteet

Artificial intelligence. 2022a. Sanakirjamääritelmä Merriam-Webster verkkosanakirjassa. Päivitetty 13.6.2022. Viitattu 20.6.2022.
https://www.merriam-webster.com/dictionary/artificial%20intelligence

artificial intelligence. 2022b. Sanakirjamääritelmä American Heritage Dictionary -verkkosanakirjassa. Viitattu 20.6.2022.
https://www.ahdictionary.com/word/search.html?q=artificial+intelligence

Aumala, O., Ihalainen, H., Jokinen, H. & Kortelainen, J. 1998. Mittaussignaalien käsittely. Tampere: Pressus.

Bishop, C. 2006. Pattern Recognition and Machine Learning. New York: Springer.
https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/

Carlson, B., Crilly, P. & Rutledge, J. 2002. Communication systems: an introduction to signals and noise in electrical communication. New York: McGraw Hill.

Fourier Transforms – scipy.fft. 2022. SciPy User Guide. Viitattu 3.5.2022.
https://scipy.github.io/devdocs/tutorial/fft.html

Fry, S. 2018. The Iotatron that Wasn't. Great Leap Years -podcast, jakso 6. Viitattu 11.6.2022.
https://www.stephenfry.com/greatleapyears/

Fundamental Frequency and Harmonics. n.d. Physics Tutorial, the Physics Classroom verkkosivusto. Viitattu 16.5.2022.
https://www.physicsclassroom.com/class/sound/Lesson-4/Fundamental-Frequency-and-Harmonics

Girardin, L. 2018. What's the Difference Between Automation, Artificial Intelligence and Machine Learning? Artikkeli GovLoop-verkkosivustolla. Julkaistu 29.11.2018. Viitattu 11.6.2022.
https://www.govloop.com/community/blog/whats-difference-automation-artificial-intelligence-machine-learning/

ISO 16:1975. 2016. Acoustics — Standard tuning frequency. Standard ISO 16:1975, International Organization for Standardization, Geneva.
https://www.iso.org/standard/3601.html

Kahl, S., Wood, C.M., Eibl, M. & Klinck, H. 2021. BirdNET: A deep learning solution for avian diversity monitoring. Ecological Informatics, 61:101236.

Kahn, J. 2002. It's Alive. Artikkeli Wired-verkkolehdessä. Julkaistu 1.3.2002. Viitattu 12.6.2022.
https://www.wired.com/2002/03/everywhere/

Kelleher, J.D. 2020. Syväoppiminen: Kuinka tekoäly toimii. Suom. K. Pietiläinen. Helsinki: Terra Cognita.

Keras Applications. nd. Keras API reference. Viitattu 12.10.2022.
https://keras.io/api/applications/

Metz, C. 2015. Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine. Artikkeli Wired-verkkolehdessä. Julkaistu 9.10.2015. Viitattu 14.10.2022.
https://www.wired.com/2015/11/google-open-sources-its-artificial-intelligence-engine/

Pound, M. 2015. How Blurs & Filters Work - Computerphile. Kuvaus, editointi S. Riley. Video Youtubessa. Julkaistu 2.10.2015. Viitattu 23.10.2022.
https://youtu.be/C_zFhWdM4ic

Pound, M. 2016. Convolutional Neural Networks Explained - Computerphile. Kuvaus, editointi S. Riley. Video Youtubessa. Julkaistu 20.5.2016. Viitattu 23.10.2022.
https://youtu.be/C_zFhWdM4ic

Prabhu, K. 2014. Window Functions and Their Applications in Signal Processing. Boca Raton: CRC Press.

scipy.signal.windows.hann. 2022. SciPy v1.8.1 Manual. Viitattu 3.5.2022.
https://scipy.github.io/devdocs/reference/generated/scipy.signal.windows.hann.html

Servick, K. 2018. In a 'tour de force,' researchers image an entire fly brain in minute detail. Artikkeli Science-verkkolehdessä. Julkaistu 19.6.2018. Viitattu 17.5.2022.
https://www.science.org/content/article/tour-de-force-researchers-image-entire-fly-brain-minute-detail

tekoäly. n.d. Kielitoimiston sanakirja sanakirja.fi-verkkopalvelussa. Viitattu 20.6.2022.
https://www.sanakirja.fi/kotus/finnish-finnish/teko%C3%A4ly

tf.keras.activations.softmax. 2022. TensorFlow API reference. Muokattu 8.9.2022. Viitattu 2.10.2022.
https://www.tensorflow.org/api_docs/python/tf/keras/activations/softmax

Turing, A. 1950. Computing machinery and intelligence. Mind, LIX:433–460. Verkkoversio. Viitattu 12.6.2022.
https://doi.org/10.1093/mind/LIX.236.433

Wren, J. 2010. Converting Acceleration, Velocity & Displacement. Artikkeli Prosig-yhtiön verkkosivustolla. Julkaistu 16.12.2010. Viitattu 14.10.2022.
https://blog.prosig.com/2010/12/16/methods-of-conversion-between-acceleration-velocity-and-displacement/

Liitteet

Liite 1. Näytelistan generoiva ohjelma

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Program to create a table from json metadata files
"""

import os
import glob
import pandas as pd
import json

DATADIR = "../data/"
SAVE_TO = "work_data/samplelist-stage1.parquet.gz"

gdadir = glob.glob('T2*', root_dir=DATADIR)
samplelist = pd.DataFrame(columns=['terminal','datafile','f_exists'
    ])
machineTypes = pd.read_excel('../data/machinetypes.ods', usecols
    =[0,1,2]) # requires odfpy

for sampledir in gdadir:
    jsons = glob.glob('*.json', root_dir=os.path.join(DATADIR,
        sampledir))

    for sonni in jsons:
        with open (os.path.join(DATADIR,sampledir,sonni)) as fp:
            metadata = json.load(fp=fp)

        itsthere = os.path.exists(os.path.join(DATADIR,sampledir,
            metadata['fileName']))

        newsample = pd.DataFrame({'terminal':[metadata['terminal'
            ]], 'datafile':[metadata['fileName']], \
                                  'metadata':[sonni], 'directory':[
                                      sampledir], \
                                  'f_exists':[itsthere], '
                                      samplecount':[metadata['
                                      actualSampleCount']], \
                                  'mVscaling':[metadata['mVScaling'
                                      ]], \
                                  'frequency':[metadata['
                                      actualSamplingFrequency']], \
                                  'RPM':[metadata['encoderRpm']], '
                                      meas':[metadata['tags'][0].
                                      split(':')[1]],\
```

```
                                    'encSource':[metadata['
                                       encoderRPMSource']['name']],\
                                    'time':[metadata['time']]})
        try:
            newsample['type'] = machineTypes[ (machineTypes['Serial
               ␣number']==metadata['terminal']) & \
                   (machineTypes['#meas']==metadata['tags'][0].split
                      (':')[1]) ].Type.values[0]
        except:
            newsample['type'] = 'unknown'
        samplelist = pd.concat([samplelist, newsample])

# Some raw data files are missing, drop those lines.
samplelist = samplelist[samplelist.f_exists == True]
# RPM 1 is an error and stopped macines are also irrelevant
samplelist = samplelist[samplelist.RPM > 1]
# Drop fixed RPMs, these machines were tested without RPM sensor
samplelist = samplelist[samplelist.encSource.str.contains('fixed')
   == False]
# Drop the ones with low sampling frequency
samplelist = samplelist[samplelist.frequency >= 48000]
# Drop shor ones
samplelist = samplelist[samplelist.samplecount > 20000]
# Drop columns no longer needed
samplelist = samplelist.drop(['encSource', 'f_exists'], axis='
   columns')
# Samples are now consistent.
samplelist.reset_index(inplace=True, drop=True)
# Save it
samplelist.to_parquet(SAVE_TO, compression='gzip')
```

Liite 2. Kiihdyvyysdatan nopeusdataksi muuntava ohjelma

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Program to convert acceleration data to velocity data
"""

import pandas as pd
import numpy as np
import scipy
import os
import signal
import sys
import pyarrow as pa
import pyarrow.parquet as pq
import time

PROGRESSFILE = 'make_velocity_parquets_progress.txt'
DATADIR = os.path.join('..','data')
SAMPLELISTFILE = 'work_data/samplelist-current.parquet.gz'
# This assumes the data is 24 bit and samplingrate as most samples,
    all other dropped
ADC_MAX = 8388608
SAMPLINGRATE = 48828.125

# In case of an interrupt:
def handler(signum, frame):
    msg = "Interrupt (signal {}). Writing progress state and
       exiting."
    print(msg.format(signum), end="\n", flush=True)
    with open(PROGRESSFILE, 'w') as pf:
        pf.write(str(i-1))
    sys.exit(2)

signal.signal(signal.SIGINT, handler)
signal.signal(signal.SIGTERM, handler)

samplelist = pd.read_parquet(SAMPLELISTFILE)

if os.path.exists(PROGRESSFILE):
    with open(PROGRESSFILE, 'r') as pf:
        initial = int(pf.read())
else:
    initial = 0

n_o_s = len(samplelist)
s_t = time.monotonic()

for i in range(initial, n_o_s):
```

```
    path = os.path.join(DATADIR, samplelist.directory.iloc[i],
       samplelist.datafile.iloc[i])
    data = np.loadtxt(path, max_rows=65536)
    accel = data/ADC_MAX
    accel = accel - accel.mean()

    velo = scipy.integrate.cumulative_trapezoid(y=accel, dx=(1/
       SAMPLINGRATE)).astype(np.float32)
    velofile = path.replace('Raw.gz', 'Velo.parquet')
    pa_velo = pa.table({"data": velo})
    pq.write_table(pa_velo, velofile)
    if (i % 1000 == 0):
        print("%s␣of␣%s,␣time␣%s" % (i, n_o_s, (time.monotonic()-
           s_t)))
```

Liite 3. Ominaisuuksia erottava ohjelma

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Make list fourier analysis frequencies, autocorrelation peaks
and other parameters for input.
It's a long run, so give a chance to interrupt and continue later.
"""

import pandas as pd
from os.path import join
import numpy as np
import scipy.signal
from scipy import fft
import time
import signal
from os.path import exists
import sys
import warnings
import pyarrow.parquet as pq
import statsmodels.api as sm

MAKE_FFT = True
MAKE_AC = True

DATADIR = '../data/'
WORKDIR = 'work_data'
OLD_SAMPLELIST = 'work_data/samplelist-stage1.parquet.gz'
NEW_SAMPLELIST = 'work_data/samplelist_w_features.parquet.gz'
PROGRESSFILE = 'make_some_inputs_progress.txt'

MVSCALING = 10545.645/8388608

# we will handle this issue
warnings.simplefilter(action='ignore', category=pd.errors.
   PerformanceWarning)

# In case of an interrupt:
def handler(signum, frame):
    msg = "Interrupt␣(signal␣{}).␣Writing␣samplelist␣and␣state␣and␣
       exiting."
    print(msg.format(signum), end="\n", flush=True)
    lines.to_parquet(NEW_SAMPLELIST)
    with open(PROGRESSFILE, 'w') as pf:
        pf.write(str(i-1))
    sys.exit(2)

signal.signal(signal.SIGINT, handler)
signal.signal(signal.SIGTERM, handler)
```

```
# Keep the original samplelist
if exists(NEW_SAMPLELIST):
    samplelist = pd.read_parquet(NEW_SAMPLELIST)
    continuing = True
else:
    samplelist = pd.read_parquet(OLD_SAMPLELIST)
    continuing = False

if exists(PROGRESSFILE):
    with open(PROGRESSFILE, 'r') as pf:
        initial = int(pf.read())
else:
    initial = 0

new_features = ['variance', 'PTP']
new_columns = new_features.copy()
ftF_columns = []
ftA_columns = []
sc_columns = []
if MAKE_FFT:
    ftF_columns = ["ftF{:02X}".format(x) for x in range(64)]
    ftA_columns = ["ftA{:02X}".format(x) for x in range(64)]
    new_columns += (ftF_columns+ftA_columns)
if MAKE_AC:
    ac_columns = ["ac{:02X}".format(x) for x in range(64)]
    new_columns+= ac_columns

if not continuing:
    for column in new_columns:
        if column not in samplelist.columns:
            samplelist[column] = np.NaN

lines = samplelist.copy()   # Clean dataframe fragmented by many
   inserts
del samplelist

start_time = time.monotonic()  # for monitoring

for i in range(initial, len(lines)):
    # load the Velocity sample
    path = join(DATADIR, lines.directory.iloc[i], lines.datafile.
       iloc[i].replace('Raw.gz', 'Velo.parquet'))
    data_arrow = pq.read_table(path)
    data = np.array(data_arrow)[0]
    # substract average
    data_norm = data - data.mean()
    # normalize between 1 and -1
    scalefactor = np.max([data_norm.max(),abs(data_norm.min())])
    data_norm = data_norm/scalefactor
```

```
sRate = lines.frequency.iloc[i]
# filter out very low frequences
b, a = scipy.signal.butter(3, Wn=9, btype='highpass', fs=sRate)
data_norm = scipy.signal.filtfilt(b, a, data_norm)
# write variance, point-to-point (max amplitude), scalefactor,
   and power
feats = dict(zip(new_features, [data.var(), data.ptp()]))
for column in new_features:
    lines.at[lines.index[i], column] = feats[column]

if MAKE_FFT:  # Calculate top frequencies with fourier
   transform
    # Make window
    N = len(data)
    W = scipy.signal.blackman(N)
    ftBlac = fft.fft(data_norm*W)
    ftReal = np.abs(ftBlac)
    freqs = fft.fftfreq(N, 1/sRate)

    # Low pass filter the FFT to reduce ripple (although we're
       in f domain now)
    b, a = scipy.signal.butter(3, 100, fs=sRate)
    ftButter = scipy.signal.filtfilt(b, a, ftReal)

    peak_indexes, peak_features = scipy.signal.find_peaks(
       ftReal[:N//2], height=20, distance=100, threshold=10)
    peak_frame = pd.DataFrame()
    peak_frame['index'] = peak_indexes
    peak_frame['height'] = peak_features['peak_heights']
    peak_frame['frequency'] = peak_frame['index'].apply(lambda
       x: freqs[x])
    # Sort frequencies by amplitude
    peak_frame.sort_values(by=['height'], ascending=False,
       inplace=True)
    peak_frame.reset_index(drop=True, inplace=True)
    peak_frame = peak_frame.reindex(range(64), fill_value=np.
       NaN)

    # Put the features into the dataframe
    for l,col in enumerate(ftF_columns):
        lines.at[lines.index[i], col] = peak_frame.iloc[l].
           frequency
    for l,col in enumerate(ftA_columns):
        lines.at[lines.index[i], col] = peak_frame.iloc[l].
           height

if MAKE_AC:
    ac = sm.tsa.acf(data_norm, nlags=0x3FFF)
```

```
        peak_indexes, peak_features = scipy.signal.find_peaks(ac,
            height=0.01, distance=10)
        peak_frame = pd.DataFrame()
        peak_frame['index'] = peak_indexes
        peak_frame['height'] = peak_features['peak_heights']
        peak_frame.sort_values(by=['height'], ascending=False,
            inplace=True)
        peak_frame.reset_index(drop=True, inplace=True)
        peak_frame = peak_frame.reindex(range(64), fill_value=np.
            NaN)
        for l,col in enumerate(ac_columns):
            lines.at[lines.index[i], col] = peak_frame.iloc[l]['
                index']

    # Occasinally tell status
    if (i % 100 == 0) :
        print("Progress: {} of {}, time: {}".format(i, len(lines),
            time.monotonic()-start_time))

lines.to_parquet(NEW_SAMPLELIST)
with open(PROGRESSFILE, 'w') as pf:
    pf.write(str(i))
print('All done!')
```

Liite 4. Spekrogrammit generoiva ohjelma

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Generate spectrograms from velocity data
"""

import pandas as pd
from os.path import join, exists
import matplotlib.pyplot as plt
from PIL import Image
import time
import pyarrow.parquet as pq
import numpy as np

DATADIR = join('..','data')

samplelist = pd.read_parquet('work_data/samplelist-stage1.parquet.
    gz')
if exists('last_spectrogram_index.txt'):
    with open('last_spectrogram_index.txt', 'r') as faili:
        last = int(faili.read())
else:
    last = 0

length=len(samplelist)

time_start = time.monotonic()
time_now = time_start
time_100start = time_start

for i in range(last, length):
    datapath = join(DATADIR, samplelist.directory.iloc[i],
        samplelist.datafile.iloc[i]).replace('Raw.gz', 'Velo.parquet
        ')
    data = pq.read_table(datapath)
    data = np.array(data[0])
    sRate = samplelist.frequency.iloc[i]
    plt.specgram(data, NFFT=1024, Fs=sRate, scale='dB', cmap='gray'
        )
    ax=plt.gca()
    plt.axis(ymin=0, ymax=(sRate/2))
    plt.axis('off')
    ax.set_frame_on(False)
    out_filepath = datapath.replace('.parquet', 'Spectro.png')
    # Save FFT without borders
    plt.savefig(out_filepath, format='png', bbox_inches='tight',
        pad_inches = 0)
    plt.cla()  # Clear the figure
```

```
    # Make a Grayscale for smaller size
    Image.open(out_filepath).convert('L').save(out_filepath)
    with open('last_spectogram_index.txt', 'w') as faili:
        faili.write(str(i))

    if (i % 100 == 0): # Show info every 100
        time_now = time.monotonic()
        time_last100 = time_now - time_100start
        time_100start = time_now
        time_sinceStart = time_now - time_start
        print("%s␣of␣%s,␣time:%s,␣last␣100␣%s" % (i, length,
            time_sinceStart, time_last100))
```

Liite 5. Ohjelma mallien koulutukseen ja evaluaatioon

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Program to dynamically load neural network model specified on
command line, prepare data and train the model.
"""

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping,
   ReduceLROnPlateau, TensorBoard
from sklearn.model_selection import train_test_split
import os
import sys
import getopt
import importlib
import matplotlib.pyplot as plt
import time
import signal

IMAGE_WIDTH=334
IMAGE_HEIGHT=217
IMAGE_SIZE=(IMAGE_WIDTH, IMAGE_HEIGHT)
IMAGE_CHANNELS=1
PATH = '../data'
MODELSAVEDIR = 'saved_models'
TENSORBOARDDIR = 'tensorlog'
WORKDIR = 'work_data'
SAMPLELIST = 'samplelist_w_features.parquet.gz'
SAMPLEPATH = os.path.join(WORKDIR, SAMPLELIST)

sys.path.append("./model_modules")

def handler(signum, frame):
    msg = "Interrupt (signal {}). Saving model and exiting."
    print(msg.format(signum), end="\n", flush=True)
    model.save(os.path.join(MODELSAVEDIR, modelname, kind))
    print("Cannot save epochs run. Edit %s/%s/%s/epochs_run.txt
       manually." \
          % (MODELSAVEDIR, modelname, kind), 'w')
    sys.exit(2)

signal.signal(signal.SIGINT, handler)
signal.signal(signal.SIGTERM, handler)

def usage():
```

```
    print("usage: {} [-e epochs] [-t type(s)] [-M meas(es)] -m
       model".format(sys.argv[0])+
           "\n       {} [--list-models] [--list-meas] [--list-types]
              [--evaluate-only]".format(sys.argv[0])+
           "\n          [--dont-evaluate]")

def evaluate():
    if 'spectogram' in modul.inputs:
        mae_train = model.evaluate(generator_train)[1]
#        messedupresults = model.predict(generator_test)
        # model.predict with generator mixes the results
        # workaround predicts one sample at a time
        guesses = []
        for _, line in samples_test.iterrows():
            linedf = pd.DataFrame({'path_to_file':[line['
               path_to_file']], 'RPM': [line['RPM']]})
            generator_one = datagen.flow_from_dataframe(
                linedf,
                directory = PATH,
                x_col='path_to_file',
                y_col='RPM',
                target_size=IMAGE_SIZE,
                class_mode='raw',
                batch_size=1,
                color_mode='grayscale'
                )
            guesses.append(model.predict(generator_one)[0,0])
    else:
        if len(shape) == 1:
            mae_train = model.evaluate(data_train[0])[1]
        else:
            mae_train = model.evaluate(data_train)[1]
        guesses = model.predict(data_test)

    evalDf = pd.DataFrame(samples_test.RPM)
#    if 'messedupresults' in locals():
#        evalDf['messedupresults'] = messedupresults
    evalDf['guesses'] = guesses
    evalDf['diff'] = evalDf.RPM - evalDf.guesses
    evalDf['absdiff'] = evalDf['diff'].abs()
    mae = evalDf['absdiff'].mean()
    hist = np.histogram(evalDf['absdiff'], bins
       =[0,5,10,30,100,1000000])[0]
    hist = hist/len(evalDf)*100
    best = evalDf['absdiff'].min()
    best_idx = evalDf[evalDf['absdiff'] == best ].index.values[0]
    worst = evalDf['absdiff'].max()
    worst_idx = evalDf[evalDf['absdiff'] == worst ].index.values[0]
    now = int(time.time())
```

```
    try:
        epoch = history.epoch[-1] + 1
    except NameError:
        epoch = epochs_run

    is_acsp = ('--accel-spectro' in optdict.keys())
    with open(os.path.join(WORKDIR, 'Trainer-eval.asv'), 'a') as
       asv: # "ampersand separeted values"
        # unix_time & meas/type & model & epochs run & train mean
           absolute error & test mean absolute error
        # & error in RPM 0...5 & 5...10 & 10...30 & 30..100 & over
           100 & best & worst sample & accel spectrogram used
        print(f"{now}&{kind}&{modelname}&{epoch}&{mae_train}", end=
           '', file=asv)
        print(f"&{mae}&{hist[0]}&{hist[1]}&{hist[2]}&{hist[3]}␣",
           end='', file=asv)
        print(f"&␣{hist[0]}&{best_idx}:{best}&{worst_idx}:{worst}&{
           is_acsp}", file=asv)

    plt.scatter(samples_test['RPM'], guesses)
    plt.savefig(os.path.join(WORKDIR, "plot{}.svg".format(now)),
       format='svg')

    savefilename = "predictDf{}.parquet.gz".format(kind)
    try:
        predictDf = pd.read_parquet(os.path.join(WORKDIR,
           savefilename) )
        predictDf[str(now)] = guesses
    except FileNotFoundError:
        predictDf = pd.DataFrame(guesses, columns=[str(now)])
    predictDf.to_parquet(os.path.join(WORKDIR, savefilename),
       compression='gzip')

    return evalDf
# ------  End of evaluation ------------------------------------


try:
    opts = getopt.getopt(sys.argv[1:], "e:m:t:M:", ['list-models','
       list-types','list-meas','evaluate-only','accel-spectro'])
except getopt.GetoptError as e:
    print(e.msg)
    usage()
    sys.exit(2)

optdict = dict(opts[0])

if '--list-models' in optdict.keys():
    import glob
    for f in glob.glob('*.py', root_dir='model_modules'):
```

```
        print(os.path.splitext(f)[0])
    sys.exit(0)

if '--list-meas' in optdict.keys():
    samplelist = pd.read_parquet(SAMPLEPATH)
    print(samplelist.groupby(['meas'])['meas'].count())
    sys.exit(0)

if '--list-types' in optdict.keys():
    samplelist = pd.read_parquet(SAMPLEPATH)
    print(samplelist.groupby(['type'])['type'].count())
    sys.exit(0)

if '-m' not in optdict.keys():
    print("Please␣specify␣model")
    usage()
    sys.exit(2)
else:
    modelname = optdict['-m']
if '-e' in optdict.keys():
    epochs_todo = int(optdict['-e'])
else:
    epochs_todo = 10

modul = importlib.import_module("model_modules.{}".format(optdict['
    -m']), './model_modules')

# Prepare sammple list and make split to train, validation  and
    test data
samplelist = pd.read_parquet(SAMPLEPATH).fillna(value=0)
if '-M' in optdict.keys():
    if '-t' in optdict.keys():
        print("Specify␣either␣type␣or␣meas,␣not␣both")
        usage()
        sys.exit(2)
    kinds = optdict['-M'].split(',')
    samplelist = samplelist[samplelist.meas.isin(kinds)]
    if len(kinds) > 1:
        kind = ','.join(kinds)
    else:
        kind = kinds[0]
elif '-t' in optdict.keys():
    kinds = optdict['-t'].split(',')
    samplelist = samplelist[samplelist.type.isin(kinds)]
    if len(kinds) > 1:
        kind = ','.join(kinds)
    else:
        kind = kinds[0]
else:
    kind = 'all'
```

```
if '--accel-spectro' in optdict.keys():  # Possibility to still use
    (wrong) accel 'grams
    samplelist['path_to_file'] = samplelist.directory+'/'+
       samplelist.datafile.apply(lambda x: x.replace('Raw.gz','
       RawAccelSpectro.png'))
    # saves in a different directory to avoid collisions with
       normal runs
    MODELSAVEDIR = 'saved_models_old'
    TENSORBOARDDIR = 'tensorlog_old'
else:
    samplelist['path_to_file'] = samplelist.directory+'/'+
       samplelist.datafile.apply(lambda x: x.replace('Raw.gz','
       VeloSpectro.png'))
samples_train, samples_rest = train_test_split(samplelist,
   train_size = .7, random_state=69)
samples_valid, samples_test = train_test_split(samples_rest,
   test_size=0.50, random_state=69)
del samples_rest
print('%s training %s validation, %s test samples' % (len(
   samples_train), len(samples_valid), len(samples_test)) )

data_train, data_valid, data_test, shape = [],[],[],[]
# Prepare image data generators if spectogram input
if 'spectogram' in modul.inputs:
    shape.append([IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_CHANNELS])
    datagen = ImageDataGenerator(rescale=1./255)
    generator_train = datagen.flow_from_dataframe(
        samples_train,
        directory = PATH,
        x_col='path_to_file',
        y_col='RPM',
        target_size=IMAGE_SIZE,
        class_mode='raw',
        batch_size=32,
        color_mode='grayscale' )
    generator_valid = datagen.flow_from_dataframe(
        samples_valid,
        directory = PATH,
        x_col='path_to_file',
        y_col='RPM',
        target_size=IMAGE_SIZE,
        class_mode='raw',
        batch_size=32,
        color_mode='grayscale' )
    generator_test = datagen.flow_from_dataframe(
        samples_test,
        directory = PATH,
        x_col='path_to_file',
        y_col='RPM',
```

```
        target_size=IMAGE_SIZE,
        class_mode='raw',
        batch_size=32,
        color_mode='grayscale' )

if 'fft' in modul.inputs:
    data_train.append(samples_train.filter(regex='^ftF..|^ftA..'))
    data_valid.append(samples_valid.filter(regex='^ftF..|^ftA..'))
    validation_data = (data_valid, samples_valid['RPM'])
    data_test.append(samples_test.filter(regex='^ftF..|^ftA..'))
    shape.append([data_train[-1].shape[1]])

if 'ac' in modul.inputs:
    data_train.append(samples_train.filter(regex='^ac..'))
    data_valid.append(samples_valid.filter(regex='^ac..'))
    data_test.append(samples_test.filter(regex='^ac..'))
    shape.append([data_train[-1].shape[1]])

if 'scale' in modul.inputs:
    data_train.append(samples_train[['variance','PTP']])
    data_valid.append(samples_valid[['variance','PTP']])
    data_test.append(samples_test[['variance','PTP']])
    shape.append([data_train[-1].shape[1]])

validation_data = (data_valid, samples_valid['RPM'])


# Prepare callbacks
tensorboard = TensorBoard(log_dir="{}/{}/{}".format(TENSORBOARDDIR,
    modelname, kind))
earlystop = EarlyStopping(patience=6, mode="min", min_delta=0.4,
   verbose=1)
learning_rate_reduction = ReduceLROnPlateau(monitor='val_loss',
   patience=2,
                                            verbose=1, factor=0.5,
                                              min_lr=0.000001)

# Load or make the model
if os.path.isdir('{}/{}/{}'.format(MODELSAVEDIR, modelname, kind) )
   :
    print("Loading␣saved␣model")
    model = tf.keras.models.load_model('%s/%s/%s' % (MODELSAVEDIR,
       modelname, kind) )
    with open('{}/{}/{}/epochs_run.txt'.format(MODELSAVEDIR,
       modelname, kind), 'r') as enumsave:
        try:
            epochs_run = int(enumsave.read())
        except ValueError:
            print("Epoch␣save␣empty,␣using␣0")
            epochs_run = 0
```

```
elif len(shape) > 1:  # give multi-input shapes as list of shapes
    model = modul.make(shape)
    epochs_run = 0
else:  # otherwise get the shape from the list
    model = modul.make(shape[0])
    epochs_run = 0
if '--evaluate-only' in optdict.keys():
    evalDf = evaluate()
    sys.exit(0)

# Train the model
if ['spectogram'] == modul.inputs:
    history = model.fit(
        generator_train,
        steps_per_epoch = len(samples_train) / 32,
        epochs = (epochs_todo+epochs_run),
        initial_epoch = epochs_run,
        validation_data = generator_valid,
        validation_steps=len(samples_test) / 32,
        callbacks=[tensorboard, learning_rate_reduction, earlystop]
    )
elif len(shape) == 1:
    history = model.fit(
        x = data_train[0],
        y = samples_train['RPM'],
        epochs = (epochs_todo+epochs_run),
        initial_epoch = epochs_run,
        validation_data = validation_data,
        validation_steps=len(samples_test) / 32,
        callbacks=[tensorboard, learning_rate_reduction, earlystop]
        )
else:  # Multi-input model
    history = model.fit(
        x = data_train,
        y = samples_train['RPM'],
        epochs = (epochs_todo+epochs_run),
        initial_epoch = epochs_run,
        validation_data = validation_data,
        validation_steps=len(samples_test) / 32,
        callbacks=[tensorboard, learning_rate_reduction, earlystop]
    )

# save the model or weights
model.save(os.path.join(MODELSAVEDIR,modelname, kind))
# save number of epochs run so far
with open("%s/%s/%s/epochs_run.txt" % (MODELSAVEDIR, modelname,
   kind), 'w') as s:
    s.write(str(history.epoch[-1] + 1))

#evaluate
```

```
if '--dont-evaluate' not in optdict.keys():
    evaldf = evaluate()
```

Liite 6. Neuroverkkomalli s_d1c2n2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D,
   BatchNormalization

inputs = ['spectogram']

def make(shape):
    model = tf.keras.Sequential(layers=[
        Input(shape=shape),
        Conv2D(filters=64, kernel_size=(3,3), strides=(3,3),
           activation='relu'),
        BatchNormalization(),
        Conv2D(filters=64, kernel_size=(3,3), strides=(3,3),
           activation='relu'),
        BatchNormalization(),

        Flatten(),
        Dense(196, activation='relu'),
        Dense(1, name='output') ], name='s_l1c1bd1')

    model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
                  loss=tf.keras.losses.MeanAbsoluteError(),
                  metrics=tf.keras.metrics.MeanAbsoluteError())
    return model
```

Liite 6. Neuroverkkomalli s_d1c3n2

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import tensorflow as tf
from tensorflow.keras.layers import Input, Dense, Flatten, Conv2D,
   BatchNormalization

inputs = ['spectogram']

def make(shape):
    model = tf.keras.Sequential(layers=[
        Input(shape=shape),
        Conv2D(filters=64, kernel_size=(3,3), strides=(3,3),
           activation='relu'),
```

```
        BatchNormalization(),
        Conv2D(filters=64, kernel_size=(3,3), strides=(3,3),
           activation='relu'),
        BatchNormalization(),
        Conv2D(filters=64, kernel_size=(3,3), strides=(3,3),
           activation='relu'),

        Flatten(),
        Dense(196, activation='relu'),
        Dense(1, name='output') ], name='s_l1c3n2')

    model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
                  loss=tf.keras.losses.MeanAbsoluteError(),
                  metrics=tf.keras.metrics.MeanAbsoluteError())
    return model
```

Liite 6. Neuroverkkomalli s_mobile2 / s_mobi-accel

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import tensorflow as tf

inputs = ['spectogram']

def make(shape):
    model_app = tf.keras.applications.MobileNet(
        weights = None,
        input_shape = shape
        )
    x = model_app.layers[-1].output
    x = tf.keras.layers.BatchNormalization(axis=-1, momentum=0.99,
       epsilon=0.001 )(x)
    prediction = tf.keras.layers.Dense(1, activation='linear')(x)
    model = tf.keras.Model(inputs=model_app.input, outputs=
       prediction, name='specto_mobilenet2')
    for layer in model.layers:
        layer.trainable = True
    model.compile(optimizer=tf.keras.optimizers.Adam(0.01),
                  loss=tf.keras.losses.MeanAbsoluteError(),
                  metrics=tf.keras.metrics.MeanAbsoluteError())

    return model
```