Tomi Heikkinen

Rinnakkaislaskennan hyödyntäminen neuro-verkkojen kouluttamisessa ja käytössä

Tieto- ja viestintätekniikan insinööri

Datasta tekoälyyn

Kevät 2023

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Heikkinen Tomi

Työn nimi: Rinnakkaislaskennan hyödyntäminen neuroverkkojen kouluttamisessa ja käytössä

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: rinnakkaislaskenta, neuroverkko, TensorFlow, Keras, Nvidia, Cuda, CPU, GPU, avoin data

Opinnäytetyössä hyödynnettiin rinnakkaislaskentaa ja kehyksiä neuroverkkojen kouluttamisessa ja käytössä. Opinnäytetyöhön koulutettiin kissa- ja koiraneuroverkko hyödyntämällä TensorFlow- ja Keras-tekniikkaa sekä PetImages-kuvia. Moniytimistä GPU:ta hyödyntävän neuroverkon koulutusprosessiin liitettiin lisäkehyksiä tai niiden ominaisuuksia. Saatuja koulutustuloksia vertailtiin keskenään ja saatiin vertailu erilaisten kehyksien vaikutuksessa neuroverkon oppimiseen.

Kaikissa neuroverkko koulutuksissa käytettiin samaa PetImages-kuvadataa. Kaikkien kehyksien kouluttamisessa käytettiin pohjana samaa TensorFlow- ja Keras-mallia. Opinnäytetyössä koulutettiin laskentayksikön kahdella GPU-suorittimella neuroverkko ja malliin lisättiin Horovod-, Sonnet-, TensorFlowOnSpark-kehykset tai niiden ominaisuuksia.

Parhaimmalla tarkkuudella saatiin kehyksistä koulutusprosessi toimimaan TensorFlowOnSpark-kehyksellä ja huonoiten Horovod-kehyksellä. Sonnetin-replikaattori vaihdettiin TensorFlow'n strategian tilalle ja sen vaikutusta mitattiin neuroverkon oppimiseen. Jokainen opinnäytetyössä käytetty kehys tai kehyksen ominaisuus on lisätty eri näkökulmasta MultiGPU:ta hyödyntävään TensorFlow- ja Keras-neuroverkkoon. Opinnäytetyön tulokset ovat vertailukelpoisia, koska eri neuroverkon kouluttamisessa on käytetty samaa neuroverkkomallia ja koulutusdataa sekä TensorFlow- ja Spark-tekniikoita.

Opinnäytetyö tuotti tietoa, miten ja millä tavalla lisäkehyksiä voidaan hyödyntää TensorFlow- ja Keras-pohjaisessa neuroverkon koulutusmallissa. Opinnäytetyö tuotti tietoa neuroverkon oppimisen tarkkuuden muutoksista, kun siihen lisätään kehyksiä tai niiden ominaisuuksia. Opinnäytetyössä on hyödynnetty Kajaanin ammattikorkeakoulun supertietokone Bullin laskentayksikköä. Opinnäytetyö edistää Kajaanin ammattikorkeakoulun MLOps-kehitystä. Opinnäytetyö tuotti avointa tietoa neuroverkkoon liitettävistä kehyksistä ja niiden hyödyntämisen mahdollisuuksista Kajaanin ammattikorkeakoulun koneoppimisympäristössä.

KAMK • University of Applied Sciences

Abstract

Author(s): Heikkinen Tomi

Title of the Publication: Utilization of Parallel Computing in the Training and Use of Neural Networks

Degree Title: Bachelor of Engineering, Information and Communication Technology

Keywords: parallel computing, Neural Network, TensorFlow, Keras, Nvidia, Cuda, CPU, GPU, Open Data

The thesis utilizes parallel computing and frameworks in training and using neural networks. For the thesis, a cat and dog neural network was trained using TensorFlow and Keras technology and PetImages. Additional frameworks or their properties were added to the training process of the neural network utilizing a multi-core GPU. The obtained training results were compared with each other, and a comparison was obtained in the effect of different frameworks on neural network learning.

All neural network trainings used the same PetImages image data. The same TensorFlow and Keras model was used as a basis for training all networks. In the thesis, a neural network was trained with the two GPU processors of the computing unit and the Horovod, Sonnet and TensorFlowOnSpark frameworks or their features were added to the model.

With the best accuracy, the training process was made to work with the TensorFlowOnSpark framework and the worst with the Horovod framework. The Sonnet replicator was replaced by the TensorFlow's strategy and its effect on neural network learning was measured. Each framework on feature of the framework used in the thesis is added form a different perspective to the TensorFlow and Keras neural network utilizing MultiGPU. The results of the thesis are comparable because the same neural network model and training data as well as TensorFlow and Spark technologies have been used to train different neural networks.

The thesis produced information on how and in what way additional frameworks can be utilized in a neural network training model based on TensorFlow and Keras. The thesis produced information about the changes in the learning accuracy of the neural network when additional frames or their properties are added to it. Kajaani University of Applied Sciences' supercomputer Bull's computing unit has been utilized in the thesis. The thesis promotes the MLOps development of Kajaani University of Applied Sciences. The thesis produced open information about the framework that can be connected to the neural network and the possibilities of their utilization in the machine learning environment of Kajaani University of Applied Sciences.

Alkusanat

Tämä opinnäytetyö on tehty Kajaanin ammattikorkeakoulun toimeksiannon pohjalta. Opinnäytetyössä käsitellään supertietokoneen rinnakkaislaskennan teknologiaa ja syväoppimisen parametreja. On olemassa sanonta, että yksi kuva on enemmän kuin tuhat sanaa. Lähtökohtaisesti se vaikuttaisi pitävän varsin yleisesti paikkaansa. On loogista ajatella, että laskutehon kasvaessa myös tiedon määrä kasvaa. Silloin datasta saadaan yhä enemmän informaatiota eli uutta tietoa. Neuroverkot oppivat paljon uutta tietoa juuri kuvien ja suurien datamassojen avulla.

Kyseessä on valtava tietomassa, jota voidaan kouluttaa neuroverkolla oikein toteutetulla mallilla. On toki totta, että kuvaa voidaan muokata ja näin vaikuttaa toivottuun lopputulokseen. Tällä voi olla myös vaikutusta neuroverkon oppimiskykyyn. Kuvien kuvankäsittely ja niiden muokkaaminen käyttötarkoitukseensa on nykyään varsin arkipäiväistä toimintaa. Neuroverkkoja koulutetaan suurilla datamassoilla ja niissä käytetään paljon parametrejä. Neuroverkko alkaa tunnistamaan ja oppimaan kuvadatan pohjalta oikealla tavalla.

Kuvien käsittelystä ja neuroverkon hyödyntämisen mahdollisuuksista keskusteltaessa aihe herättää kiinnostusta kuulijoissaan. Ihmiset etsivät yhä uusia käyttötarkoituksia neuroverkoille. Tällainen kehitys voi johtaa jossakin vaiheessa vahvan tekoälyn kehittymiseen. Näkisin myös, että rinnakkaislaskenta liittyy hyvin voimakkaasti supertietokoneiden energiatehokkuuteen ja niiden kustannuksien arviointiin. Kyseessä on teknologia, jonka kehityskaari voi yltää lähitulevaisuudessa yhä tehokkaampaan ihmiselämää edistävämmäksi tekijäksi maapallolla.

” I don’t want to believe, I want to know.”

- Carl Sagan

Sisällys

Symboliluettelo

Anaconda	Data science technology. Datatieteen tekniikka.
ANN	Artificial neural network. Neuroverkko.
BVLC	BACnet Virtual Link Control. BACnet-verkkokerros.
CPU	Central processing unit. Tietokoneen prosessori.
CUDA	Compute unified device architecture. Nvidian laskenta-arkkitehtuuri.
cuDNN	Cuda DNN. GPU:lla kiihdytetty primitiivikirjasto syvien hermoverkkojen kouluttamiseen.
CNN	Convolutional Neural Network. Konvoluutiohermoverkko.
DNN	Deep neural networks. Syvät hermoverkot.
Docker	Accelerated, Containerized Application Development. Nopeutettu, konttipohjainen sovelluskehitys.
GPU	Graphics processing unit. Tietokoneen grafiikkaprosessori.
GPGPU	General-purpose computing on graphics processing units. Yleiskäyttöinen laskenta graafisilla prosessoinneilla.
MLOps	Machine learning operations. Koneoppimistoiminnot.
RBFN	Radial Basis Function Network. Kantafunktioverkko.
TensorFlow	Free and open-source software library for artificial intelligence. Ilmainen ja avoimen lähdekoodin ohjelmistokirjasto tekoälylle.
TFDS	TensorFlow datasets. Tensorflow'n tietojoukko
TPU	Tensor Processing Units. Tensorin prosessointiyksiköt.

1. Johdanto

Opinnäytetyön tavoite on kouluttaa rinnakkaislaskentaa ja lisäkehystä hyödyntämällä TensorFlow- ja Keras-neuroverkkoa. Opinnäytetyön tutkimusongelma on selvittää, millaisella tarkkuuden muutoksella on mahdollista liittää neuroverkon kouluttamiseen TensorFlow'ta ja Kerasta tukeva kehys tai tietty kehyksen ominaisuus. Opinnäytetyön tutkimusmenetelmissä yhdistyy määrällinen ja laadullinen tutkimus.

Opinnäytetyö on tarkoituksena liittää osaksi Kajaanin ammattikorkeakoulun tekoäly- ja koneoppimiskehitystä. Opinnäytetyö tuottaa tietoa liittyen supertietokoneiden GPU-laskennasta ja kehyksien hyödyntämisestä neuroverkkojen kouluttamisessa. Opinnäytetyö toimii dokumentaationa liittyen Kajaanin ammattikorkeakoulun MLOps-ketjun kehittymiseen.

Rinnakkaislaskennan toteuttaminen grafiikkasuorittimilla (GPU) on selkeä vaikutus neuroverkkojen optimointiin ja niiden suorituskyvykkyyteen [1, s. 204]. Supertietokoneen energiatehokkuus on huomattavasti tehokkaampaa, kun käytetään CPU-laskennan sijasta GPGPU-laskentaa [2, s. 29]. Työympäristön käytössä oleva suoritin on yleisimpiä pullonkauloja DNN-verkkojen kouluttamisessa [3, s. 9957].

Laskennan rinnakkaistaminen valmiiden kirjastojen avulla voidaan yhä tehokkaammin analysoida tutkimusaineistoja. Näin saadaan tuotettua myös nopeammin oleellista tietoa analysoitavasta aineistosta. Cuda-algoritmeja voidaan luontevasti testata erilaisiin sovelluskehyksiin, koska niissä on useimmiten vain pieniä keskinäisiä eroja havaittavissa. [1, s. 223.]

Tutkimuskysymykset:

1. Miten TensorFlow- ja Keras-pohjaisen kissa- ja koira-neuroverkon oppimistarkkuus muuttuu, kun siihen lisätään kehys rinnakkaislaskennan näkökulmasta?

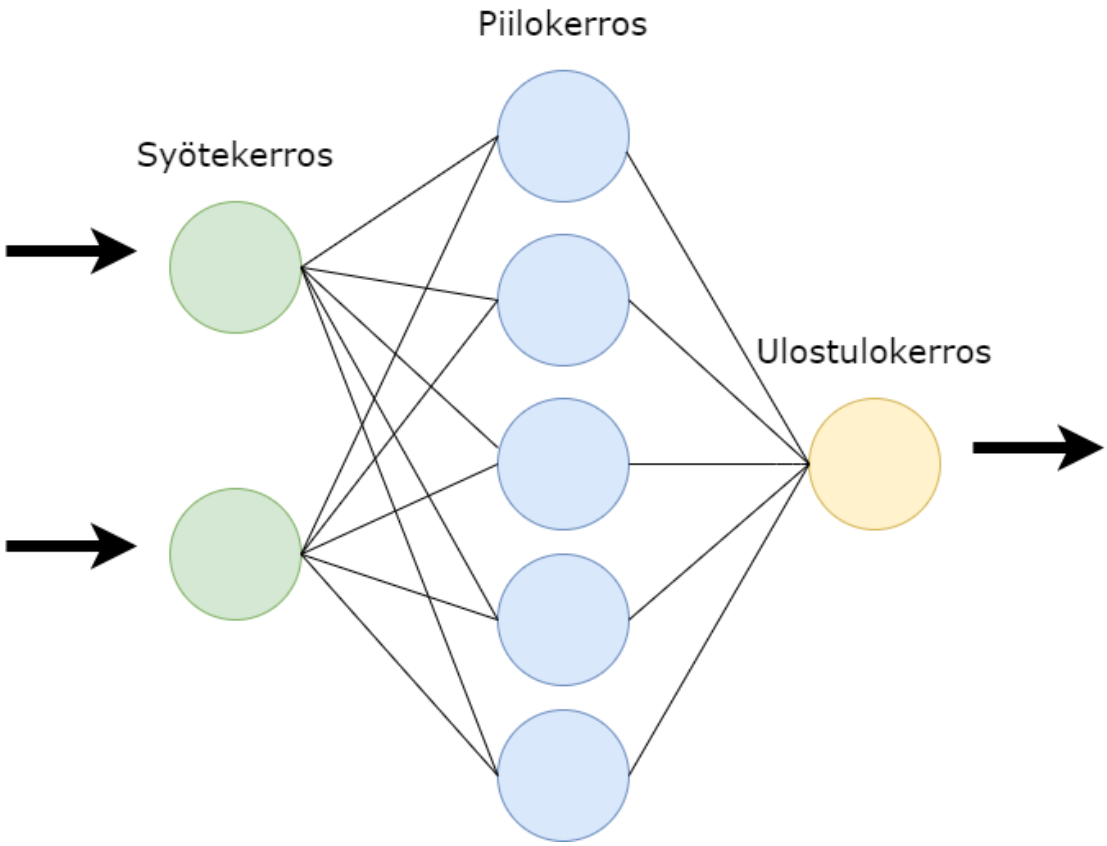
2. Millaisia vaikutuksia TensorFlow- ja Keras-neuroverkon kehyksien lisäämisellä on rinnakkaislaskentaan?

2. Rinnakkaislaskennan hyödyntämisen mahdollisuuksia

2.1 Neuroverkkojen toimintaperiaate

Neuroverkkoja on kehitetty 1940-luvulta lähtien. Kuvassa 1 on esitetty pelkistetty malli neuroverkosta. Syvä neuroverkko (DNN) pystyy oppimaan tehokkaammin uutta tietoa useampien piilotettujen kerroksien avulla kuin perinteinen neuroverkko (ANN). Neuroverkkoon piilotettu kerros kykenee muuttamaan ongelmadatan sellaiseen ominaisuustilaan, jolla voidaan ratkaista tehokkaasti tiedonkäsittelyn ongelma. Tällaista neuroverkkorakennetta voidaan hyödyntää regressioissa, luokittelussa, kuvien tunnistamisessa ja myös muissa tämän tyyppisissä ongelmissa. [4, s. 3.]

Toistuva epälineaaristen aktivointifunktioiden käyttäminen neuroverkoissa luo epälineaarisuutta. Neuroverkkojen toimintaan liittyvät tietyntyyppiset sääntöjoukot, joita kutsutaan datan luokitteluksi. Tällainen neuroverkon luokitteluun perustuva kouluttaminen voi helpottaa suunnittelua. [5, s. 8.]



Kuva 1. Neuroverkko [6]

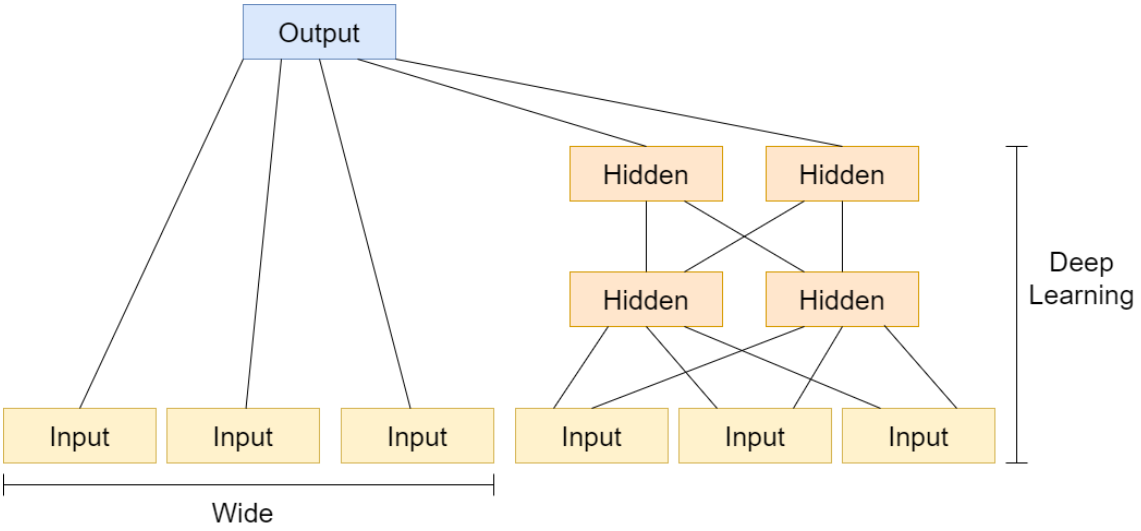
Matalarakenteisen neuroverkon oppimisen pullonkaula on yksi ja ainoa piilokerros. Tällaisen mallin tehokkuus ja laskentakyky ovat hyvin rajallisia, kun käsitellään monimutkaisia epälineaarisia funktioita. Syvä neuroverkkomalli on noussut juuri tehokkaampien näytedatan piirrevektorien myötä tutkijoiden suosioon. Lisäksi syvillä neuroverkoilla on saavutettu tehokkaampaa oppimistuloksia, ja datan luonne sekä sen ominaisuuspiirteet ovat kohdanneet luontevammin. Syviä neuroverkkomalleja on hyödynnetty yhä enemmissä määrin Big datan aikakaudella uusien tietojoukkojen myötä. Tällaisia tietojoukkoa ovat esimerkiksi ImageNet-kuvantunnistusaineisto ja Project Getenbergin kieleen liittyvät tietojoukot. [4, s. 3.]

2.2 Syväoppiminen ja neuroverkkojen opettaminen

Tämän opinnäytetyön käytännönosiossa hyödynnetään ja koulutetaan kissa- ja koirakuvilla CNN-neuroverkkoa. Kuvataiteen opettamisessa ja oppimisessa on esimerkiksi hyödynnetty tekoälyn todella nopeaa kehitystä. Syväoppiminen on ollut todella tehokasta, kun on hyödynnetty RBF-neuroverkkoa. Kyseinen neuroverkkomalli kykenee hyödyntämään epälineaarisia aktivointifunktioita, jolloin kuvien luokittelun tarkkuus on jo varsin tehokasta. [5, s. 8.]

Neuroverkkojen ja etenkin samanaikaisten neuroverkkojen kouluttamisessa on käytössä olevalla kehyksellä vaikutus lopputulokseen. Työtaakkaa voidaan keskittää myös vain tietylle prosessorille. Kuitenkin näihin edellisiin valintoihin vaikuttavat inputtien koko ja tarkkuuden erot. [3, s. 9958.]

DNN-neuroverkoissa on useita kerroksia, joiden lukumäärä vaikuttaa puolestaan laskentavaatimuksiin. Kun kasvatetaan kerroksien lukua ja kokoa, niin niillä on myös vaikutus neuroverkon suorituskykyyn. Erityyppisiä neuroverkkoja voidaan varsin tehokkaasti testata kehyksien avulla. Kuvassa 2 on esitetty useampi neuroverkko samassa kehyksessä - ajatusta. Monitasoiset kehykset soveltua todella hyvin moniin sovelluksiin. Berkleyn visio- ja oppimiskeskus on kehittänyt useita suosittuja ja avoimen lähdekoodin syväoppimiskehyksiä. [7.]

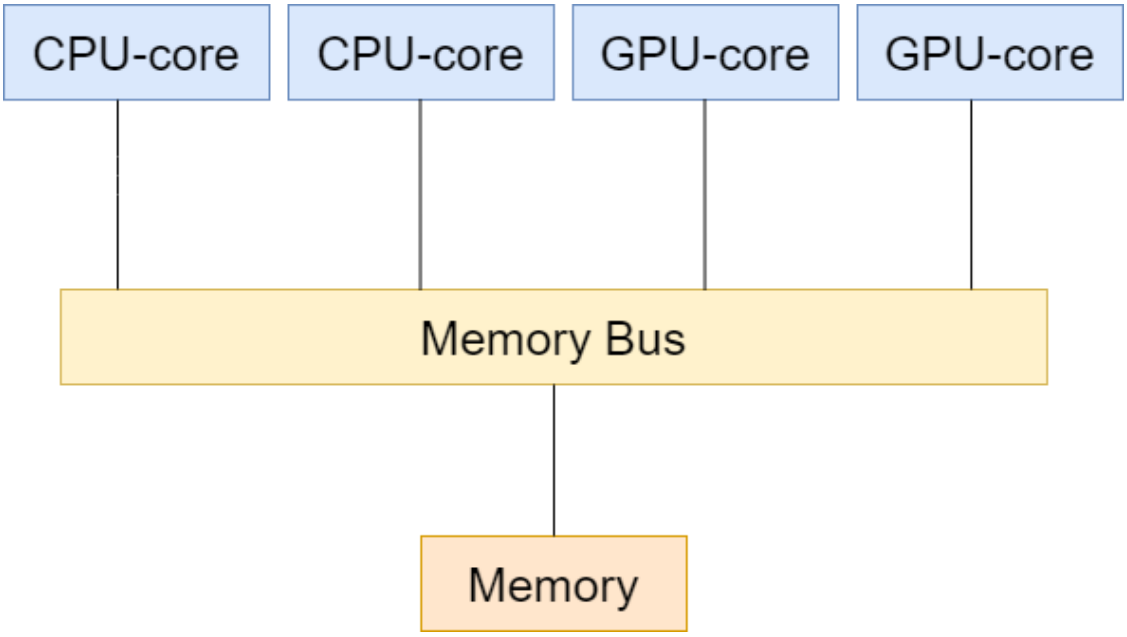


Kuva 2. Useampi neuroverkko samassa kehyksessä [7]

Nvidian-ympäristöissä voidaan hyödyntää useampien verkkojen rinnakkaiskouluttamisessa DIGITS-työkalua, joka soveltuu hyvin kuvien luokitteluun- ja objektientunnistustehtäviin [7].

2.3 GPU-prosessorien käyttö neuroverkkojen opetuksessa

Kuvassa 3 on kuvattu, miten rinnakkaisessa ongelman ratkaisemisessa prosessorit voivat jakaa yhteisen muistin. Tällaisen arkkitehtuurin haasteita ovat muistin nopeus, kapasiteetti ja muistin käytön prioriteettiin liittyvät yksityiskohdat. Yhteinen muisti yksinkertaistaa rinnakkaislaskenta-ohjelmointia ja sen prosesseja. [8.]



Kuva 3. Yhteinen muisti [8]

Ideaalisesti neuroverkon kouluttumisen pitäisi nopeutua siten, että käytössä olevien suorittimien määrä vaikuttaa suoraan rinnakkaislaskennan nopeutumiseen. Kyse on kuitenkin prosessorien välisestä yhteistyöstä ja ennen kaikkea niiden välisistä kommunikointiominaisuuksista. Rinnakkaislaskennan kouluttamisen nopeutuminen saadaan selvitettyä tarkasti mittaamalla nopeus ensiksi yhdellä prosessorilla ja sen jälkeen vasta useammalla prosessorilla. [2, s. 2.]

Muistin tekniset haasteet rajoittavat suurien prosessorimäärien käyttämisen samassa koneessa. Näin ollen supertietokoneet on toteutettu useammasta laitteesta, jotka on linkitetty nopeilla verkkoyhteyksillä. Näin supertietokoneiden laitteiden välinen tiedonsiirto tapahtuu vain verkon välityksellä. Tällainen arkkitehtuuri mahdollistaa valtavien loputtomien tietokonemäärien yhdistämisen yhteen ja samaan supertietokoneeseen. [8.]

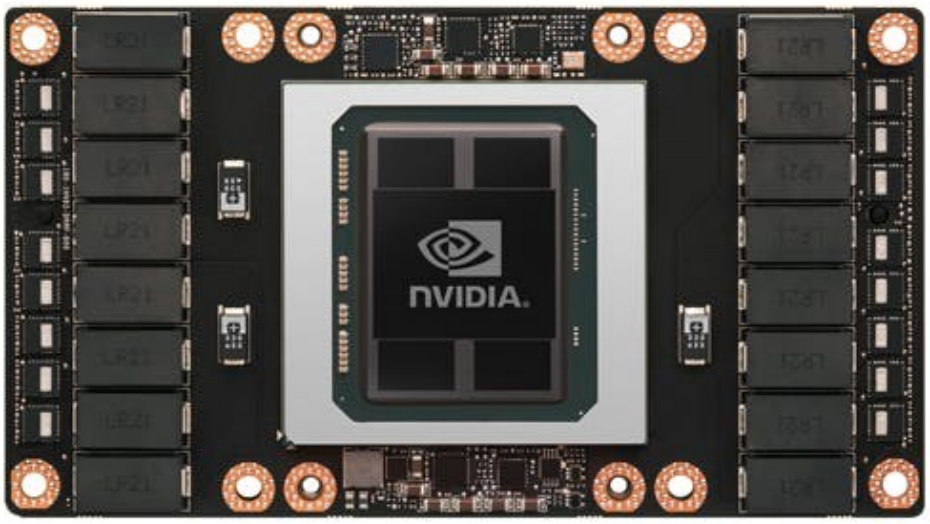
DNN-arkkitehtuuri vaatii tehokkaan ympäristön laskutoimituksille. Näin räätälöidyt laitteistoarkkitehtuurit nousevat keskiöön. Tällainen laitteistoarkkitehtuuri vaatii maksimaalisen kiihtyvyyden saavuttamiseksi oikein räätälöidyn kehyksen. Tiedemaailmasta ja teollisuudesta löytyy vertailuarvoja, joita tietoja tulisisi analysoida tarkemmin. [3, s. 9949–9950.]

2.4 Neuroverkkojen opetuksen hajauttaminen usealla GPU-prosessorille

GPU-laitteet ovat moniytimisiä, jolloin niistä on hyötyä tekoälyn syväoppimisessa. Syväoppiminen on yksi oleellisimpia tekoälyn kehittymisen osa-alueita. Moniytimisten GPU-laitteiden suorituskyky on kehittynyt huomattavan nopeasti. Tällainen kehitys on mahdollistanut kehyksien ja työkalujen yhä tehokkaamman hyödyntämisen, myös muilla ohjelmointikielillä kuin C-ohjelmointikielellä. Alusta valinnalla on ehkä tällä hetkellä suurin merkitys syväoppimisen kehittymisen kannalta. [7.]

Neuroverkkojen (DNN) hyödyntämisen mahdollisuudet ovat huikeat. Neuroverkkoja on hyödynnetty esimerkiksi valokuvien käsittelyssä ja oleellisena osana itseohjautuvien autojen toteutusta. Kyseessä on tosiaan vain yksi tekoälyn tutkimus- ja kehitysalusta, jonka suosio on kasvanut prosessoritehojen kasvun mukana. Nvidian tekniikka tukee CUDA-ohjelmistoympäristöä ja DNN-kehyksiä. Tällaisia Cuda DNN (cuDNN) numeraalisessa laskennassa käytettäviä kehyksiä ovat esimerkiksi TensorFlow ja avoimen lähdekoodin ohjelmistokirjastot. [9.]

Tunnetuimpia kehyksiä ovat OpenCL ja Nvidian Cuda DNN. Useasti tällaisilla DNN-alustoilla käytetään olemassa olevia tai uusia kehyksiä. OpenCL-ympäristöt pystyvät hyödyntämään useita alustoja moniytimisilläsuorittimilla ja myös GPGPU-ryhmiä. Nvidian tekninen ratkaisu on toteutettu sen omalla GPU-tekniikalla, joka soveltuu ja suoriutuu hyvin neuroverkkojen oppimistilanteissa. Kuvassa 4 on Nvidian GPU-prosessori. [7.]



Kuva 4. Nvidia’s Tesla P100 GPU [9]

DNN-sovelluksien kouluttaminen vaatii valtavia määriä dataa, jotta neuroverkon painot ja solmut saadaan oikein määritettyä. Puolestaan valmiiksi koulutettu neuroverkko voidaan toteuttaa käytännössä hyödyntämällä hyvinkin yksinkertaisia mikrokontrollereita. [7.]

Selkeä haaste neuroverkon kouluttamisessa on DNN-tutustuminen. Useasti kehykset ovat valmiiksi esikonfiguroituja kuvantunnistusta varten. Yleensä työkaluilla toteutettavaan uuteen konfiguraation virittämiseen ja sen kehittäminen tarvitaan asiantuntijaa [7.]. DNN-toteutukset eivät todellakaan ratkaise tekoälyn kaikkia ongelmia, mutta niiden käytettävyys ja käytännönläheisyys ovat tehneet niistä yleisessä käytössä hyvin suosittuja [9].

Koneoppiminen ja luova ongelmanratkaisu hyödyntävät hyvin ennalta määriteltyjä matemaattisia kompromisseja. Esimerkiksi Bayesian-agentti osaa hyödyntää todennäköisyyksien kannalta saatavilla olevaa tietoa optimaalisesti. Tällainen laskusuoritus on fyysisesti kovin haasteellinen yksittäiselle fyysiselle tietokoneelle. Tekoälyn tavoitteena voi olla löytää oikoteitä toteuttaa riittävän laadukasta ja optimaalista tarkkuuden laskentaa sekä samalla säilytetään kiinnostavan asian tutkimisessa korkea suorituskyky. [10, s. 11.]

3. Rinnakkaislaskennan hyödyntäminen käytännössä

Opinnäytetyöhön koulutettiin kissa- ja koirakuvilla neuroverkko Kajaanin ammattikorkeakoulun Bull-supertietokoneen laskentayksiköllä. Laskentayksikössä on hyödynnetty neuroverkon kouluttamisessa kahta GPU-prosessoria samanaikaisesti.

3.1 Data-aineistojen esittely

Opinnäytetyössä koulutettiin neuroverkkoja Kagglen PetImages-datalla [11], datasetistä on neuroverkon kouluttamiseen poistettu korruptioituneet kuvat. Kuvasta 5 voidaan todeta, että neuroverkko käytti kouluttamisessa 23410 kuvaa ja kuvat kuuluvat kahteen luokkaan.

```
Deleted 1590 images
Found 23410 files belonging to 2 classes.
Using 18728 files for training.
Found 23410 files belonging to 2 classes.
Using 4682 files for validation.
```

Kuva 5. Poistetut ja löydetyt kuvat

Kagglen koulutus- ja testikuvia on yhteensä 25000 kappaletta, joista on poistettu 1590 kappaletta. Kuvassa 6 korruptoitunutta dataa näkyy komentokehotteessa. Korruptioitunut data on poistettu datasetistä ohjelmistokoodissa. Opinnäytetyön kehyksien kouluttamisessa käytettiin PetImages-kuvia. Neuroverkolle syötettävien opetuskuvien koko skaalattiin 180 pikseliin. (Liite 8)

```
Corrupt JPEG data: 128 extraneous bytes before marker 0xd9
Corrupt JPEG data: 65 extraneous bytes before marker 0xd9
Corrupt JPEG data: 396 extraneous bytes before marker 0xd9
Corrupt JPEG data: 239 extraneous bytes before marker 0xd9
Corrupt JPEG data: 252 extraneous bytes before marker 0xd9
Corrupt JPEG data: 1153 extraneous bytes before marker 0xd9
Corrupt JPEG data: 162 extraneous bytes before marker 0xd9
Corrupt JPEG data: 214 extraneous bytes before marker 0xd9
Corrupt JPEG data: 99 extraneous bytes before marker 0xd9
Corrupt JPEG data: 1403 extraneous bytes before marker 0xd9
Corrupt JPEG data: 2226 extraneous bytes before marker 0xd9
Corrupt JPEG data: 228 extraneous bytes before marker 0xd9
Warning: unknown JFIF revision number 0.00
```

Kuva 6. Korruptoitunutta Jpeg-dataa

3.2 Laskentayksikön esittely

Supertietokone ei todellakaan tarkoita yhtä nopeaa tietokonetta (Liite 1). Supertietokone sisältää useamman nopean tietokoneen. Opinnäytetyössä rinnakkaislaskentaa käsitellään yhden laskentayksikön välityksellä. Kuvassa 7 laskentayksikköön on lisätty MiniConda-ympäristö, jonka kautta toimii Jupyter Notebook-ympäristö GPU-kehyksineen.

```
# conda environments:
#
base                  *  /home/tomihei/miniconda3
tf_analyticszoo          /home/tomihei/miniconda3/envs/tf_analyticszoo
tf_bigDL                 /home/tomihei/miniconda3/envs/tf_bigDL
tf_elephas               /home/tomihei/miniconda3/envs/tf_elephas
tf_horovod               /home/tomihei/miniconda3/envs/tf_horovod
tf_keras                 /home/tomihei/miniconda3/envs/tf_keras
tf_mesh                  /home/tomihei/miniconda3/envs/tf_mesh
tf_petastorm             /home/tomihei/miniconda3/envs/tf_petastorm
tf_singa                 /home/tomihei/miniconda3/envs/tf_singa
tf_sonnet                /home/tomihei/miniconda3/envs/tf_sonnet
tf_spark                 /home/tomihei/miniconda3/envs/tf_spark
```

Kuva 7. MiniConda-ympäristö

TensorFlow on mahdollista kouluttaa neuroverkko useammalla GPU-prosessorilla, koneella tai TPU:lla [12]. Olemassa olevia malleja ja niiden koulutustiedostoja pääsee TensorFlow:ssa hyödyntämään koodimuutoksilla. TensorFlow'n strategia on suunniteltu palvelemaan useita käyttäjäsegmenttejä. Voidaan yleisesti todeta, että kyseinen tekniikka sisältää lähtökohtaisesti hyvän suorituskyvyn (Liite 5). TensorFlow'ssa on helppoa vaihtaa tilanteen mukaan strategiaa ja jatkaa sillä neuroverkon kouluttamista. [11.]

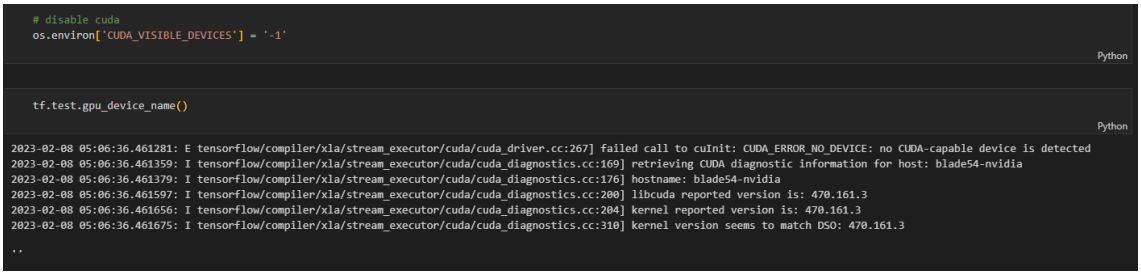
Laskentaympäristössä koulutettiin MiniConda- ja Docker-Compose-ympäristössä (Liite 6). Ohjaimena käytettiin Nvidian ajuria 470.141.03 ja Cuda-versiota 11.2 sekä hyödynnetään samanaikaisesti kahta GPU-prosessoria [11]. Opinnäytetyön syväoppimisen kehyksien koulutus on toteutettu MiniConda-ympäristössä.

MiniConda-ympäristössä käytettiin TensorFlow-GPU versiota 2.11.0. Lähtökohtaisesti kehyksien asennusohjeet on tehty yhteensopiviksi MiniConda-ympäristön kanssa [17]. Eri ympäristöjä hallittiin ja koulutettiin itsenäisissä enveissä. Tulevaisuudessa voidaan yhä enempi hyödyntää kehyksiä Docker-Compose-tyyppisissä ympäristöissä. MiniConda-ympäristön käyttöön ottaminen on helppoa.

Neuroverkkomallilla on liki neljämiljoonaa koulutusparametria käytössään (Liite 2). Käytössä oleva neuroverkon malli ja data toistuivat useissa Cat&Dog-neuroverkon syväoppimisen esimerkeissä. Tällainen yleisesti käytössä oleva malli kouluttamissa lisää tämän opinnäyteyön luotettavuutta ja toistettavuutta. [13; 14.]

3.3 CPU-prosessorilla nopeuden mittaaminen

Ohjelmakoodissa voidaan määrittää käytettävät laitteet ympäristömuuttujan avulla yksilöllisesti. Näin voidaan nopeasti neuroverkon kouluttamisessa valita CPU-prosessori tai yksittäinen GPU-prosessori erilaisissa solmurakenteissa (Kuva 9 & 11). Kuvassa 8 GPU-prosessori on kytketty pois päältä. Kyseistä menetelmää voidaan hyödyntää, jos on tarkoitus yhdellä koneella ajaa useampi ohjelma samanaikaisesti omissa ympäristöissään. [15.]



Kuva 8. GPU-suorittimen poiskytkentä

Kun Cuda on asetettu pois päältä, niin laskentayksikkö käyttää vain CPU-prosessoria TensorFlow'ssa. Kuvasta 9 on määritetty TensorFlow-koodiin, että käytetään vain CPU-prosessoria.

```
# Used only CPU
with tf.device('CPU'):
    history = model.fit(ds_pet_train, validation_data=ds_pet_test, epochs=epochs)
```

Kuva 9. Laskentaan määritetty CPU-prosessori

3.4 GPU-prosessorilla nopeuden mittaaminen

Laskentayksikön unified-muisti mahdollistaa useiden GPU- ja CPU-prosessorien yhteisen muistin jakamisen hallitusti. Kaikkien GPU-prosessorien tulee tukea vertaismuistin käyttöä (P2P memory access) [14]. Kuvasta 10 voidaan havaita, että laskentayksiköllä on kaksi GPU-prosessoria. Laskentayksikön laitteistoajuria ei voi enää päivittää uudempaan versioon (Liite 1).

```
# nvidia-smi
Mon Oct 17 17:12:01 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.141.03   Driver Version: 470.141.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K40t          Off  | 00000000:01:00.0 Off |                    0 |
| N/A   58C    P0   138W / 235W |  10938MiB / 11441MiB |     95%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K40t          Off  | 00000000:81:00.0 Off |                    0 |
| N/A   47C    P0    62W / 235W |     67MiB / 11441MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
+-----------------------------------------------------------------------------+
#
```

Kuva 10. Nvidia-smi

TensorFlow tukee GPU-suorittimella tieteellistälaskemista. TensorFlow-koodiin tulee määrittää GPU-prosessorille tarvittavat parametrit [12]. Kuvassa 11 on neuroverkon kouluttamiseen määritetty käytettäväksi ensimmäinen GPU-prosessori.

```
# Used only GPU
with tf.device('GPU:0'):
    history = model.fit(ds_pet_train, validation_data=ds_pet_test, epochs=epochs)
```

Kuva 11. Laskentaan määritetty GPU-prosessori

TensorFlow:n prosessorien määritysparametreja [12]:

”/device:GPU:0” : Laskentayksikön ensimmäinen GPU-prosessori.

”GPU:0” : Lyhyempi merkintä laskentayksikön ensimmäisestä GPU-prosessorista.

”/job:localhost/replica:0/task:0/device:GPU:1” : Toisen GPU-prosessorin täydellinen nimi, joka näkyy TensorFlow:lle.

3.5 Useammalla GPU-prosessorilla nopeuden mittaaminen

Opinnäytetyöhön koulutettiin kissa- ja koiraneuroverkko käyttäen kahta GPU-prosessoria ja hyödynnettiin Tensorflow:n hajautusohjeita. Tensorflow’n **”tf.distribute.Strategy”** on suunniteltu tukemaan tutkijoita ja insinööreitä. TensorFlow-tekniikka tarjoaa tehokkaan suorituskyvyn ja tekniikkaa voidaan skaalata erilaisten strategioiden välillä [16]. Kuvassa 12 kumpikin GPU-prosessori on käytössä.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.141.03   Driver Version: 470.161.03   CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla K40t          Off  | 00000000:01:00.0 Off |                    0 |
| N/A   76C    P0    74W / 235W |  11122MiB / 11441MiB |     67%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla K40t          Off  | 00000000:81:00.0 Off |                    0 |
| N/A   77C    P0    87W / 235W |  11122MiB / 11441MiB |     68%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A    315473      C   .../envs/tf_keras/bin/python    11117MiB |
|    1   N/A  N/A    315473      C   .../envs/tf_keras/bin/python    11117MiB |
+-----------------------------------------------------------------------------+
```

Kuva 12. Kaksi GPU-prosessoria käytössä

TensorFlow'n MirroredStrategy tukee hajautettua koulutusta useilla GPU-prosessoreilla samassa laskentayksikössä. Kuvassa 13 on esitetty, miten MirroredStrategy'a voidaan käyttää ohjelmistokoodissa. GPU-prosessorit replikoidaan ja näin muodostuu MirroredVariable. Laitteet kommunikoivat keskenään hyödyntämällä all-reduce-algoritmia. [16.]

```
strategy = tf.distribute.MirroredStrategy()
# strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
```

Kuva 13. MirroredStrategy

Kuvassa 14 on Scope-määritys ja se tulee tehdä ennen mallin sovittamista neuroverkolle. Fit-komentoa käytetään esimerkiksi Keras-pohjaisissa toteutuksissa. Scopen käyttö mahdollistaa strategian siirtämisen neuroverkon malliin. [17.]

```
with strategy.scope():
    model = build_and_compile_model()
```

Kuva 14. Strategy Scope

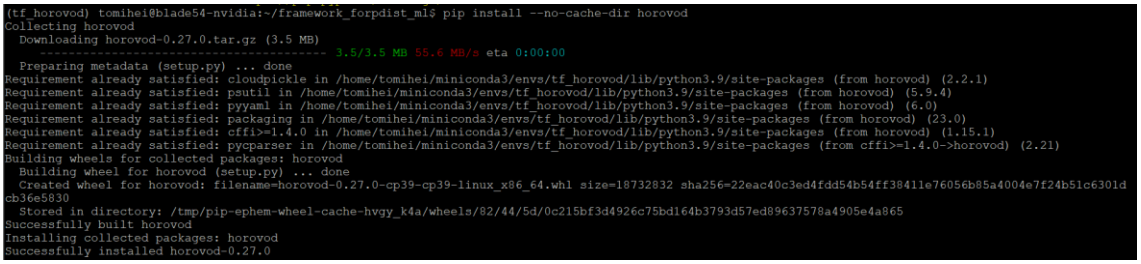
3.6 Ohjelmistokehyksien esittely

Mudugandla:n artikkelista on käsitelty ohjelmistokehyksiä, joista valikoitui Horovod ja TensorFlowOnSpark-kehykset tähän opinnäytetyöhön ja niitä on vertailtu GPU-koulutusympäristössä [18]. Kolmanneksi kehykseksi valikoitui suosittu Sonnet, jonka suunnittelu on toteutettu TensorFlow:n pohjalta [19].

Laskentayksikkö ja TensorFlow- ja Spark-neuroverkkomalli rajasivat käytettäviä kehyksiä. Valittavien kehysten tuli tukea TensorFlow:ta ja strategisesti useampaa GPU-prosessoria. Toisaalta laskentayksikön näytönohjaimien ajuri-versio rajasivat koulutettavia kehyksiä (Taulukko 2 ja Liite 1).

Horovod

Horovod koulutuskehystä voidaan hyödyntää TensorFlow-, Keras- ja PyTorch-ympäristöissä. Kuvassa 15 on esitetty Horovod-kehyksen pip-asennusprosessi. Kehyksen avulla syväoppimisesta muodostuu helppokäyttöistä ja nopeaa. Tekniikka toimii yhdellä tai useammalla grafiikkasuorittimella. [18.]



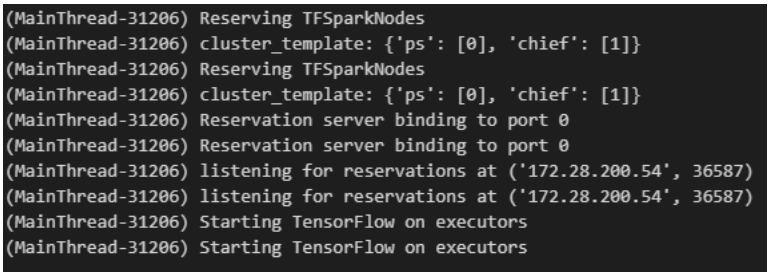
Kuva 15. Horovod-kehyksen asentaminen

Sonnet

Sonnet on rakennettu TensorFlow'n päälle. Koneoppimismenetelmä tarjoaa pelkistetyn syntaksin ja hyvän dokumentaation. Dokumentaation pohjalta voi rakentaa myös uuden neuroverkon kouluttamisen soveltuvan mallin. [19.]

TensorFlowOnSpark

Kuvassa 16 on esitetty TensorFlow:n ja Sparkin keskinäistä kommunikaatiota. TensorFlowOnSpark-kehyksessä yhdistyy syväoppimiskehyksien tärkeimmät ominaisuudet Apachen Hadoopin ja Sparkin kanssa kehys mahdollistaa syväoppimisen CPU- ja GPU-klustereissa. Tekniikka soveltuu hyvin hajautettuun TensorFlow-koulutukseen. Kyseistä teknologiaa on kehittänyt Yahoo hyödyntämällä Hadoop-klustereita pilvessä. [18.]



Kuva 16. Spark ja sen nodet

3.7 Hajautuskirjastojen testaaminen GPU-prosessoreilla

Opinnäytetyössä koulutettiin neuroverkkoja Keras- ja TensorFlow-ohjelmointikirjastoilla CPU- ja GPU-prosessoreilla MiniConda-ympäristössä. Olemassa olevaan neuroverkkoon tai neuroverkkomalliin voidaan lisätä kehyksiä ja näin päästään vertailemaan niiden keskinäisten tuloksien eroja moniytimisessä GPU-koulutusympäristössä [12]. Kehyksien tekniikan ymmärtäminen ja niiden ominaisuuksien sekä mahdollisuuksien oivaltaminen ovat aina oma prosessinsa.

MiniConda-ympäristössä kehyksiä on melko helppoa kouluttaa kahdella GPU:lla ja hyödyntämällä TensorFlow'n hajautusohjeita. Kehyksien työohjeiden mukaisesti valtaosa kehyksistä asennetaan pip-komennolla [18 ja 19]. Tulevaisuudessa todennäköisesti kehyksien Docker-kuvat yleistyvät ja niiden hyödyntäminen Docker-Compose-ympäristöissä helpottuu ja monipuolistuu. Laskentayksikön suorituskyvyssä ei ilmennyt juurikaan eroja Docker-compose tai MiniConda-ympäristöissä (Liite 5).

Horovod, Sonnet ja TensorFlowOnSpark ovat aivan erilaisia teknologioita ja kehyksiä kouluttaa neuroverkkoa. Jokainen kehys vaatii koodimuutoksia ja lisäyksiä toimiakseen. Jokaisen kehyksen pohjana käytettiin TensorFlow- ja Keras-MGPU-tiedostoa (Liite 8).

Horovod

Horovod-kehys tulee alustaa hvd.init() -komennolla Keras-pohjaiseen koodiin. Jokainen laskentayksikön GPU-prosessori kiinnitetään yhteen prosessiin. Malli tukee myös TensorFlow'n mirroredstrategiaa. Kuvassa 17 Horovod-kehykselle on määritetty distributedOptimizer-optimoija. [22.]

Optimoita delegoi gradienttien laskentaa alkuperäiselle optimoijalle hyödyntämällä allreduce- tai allgather-toimintoja [22]. Horovod vaati neuroverkon optimoijan muokkaamisen Keras legacyksi, jotta se alkoi toimimaan [23].

```
optimizer = tf.keras.optimizers.legacy.SGD()
optimizer = hvd.DistributedOptimizer(optimizer)
model.compile(optimizer=optimizer, loss="binary_crossentropy", metrics=["accuracy"])
return model
```

Kuva 17. Horovod-optimoija

Sonnet

Sonnetin kehykseen sisältyy replikaattori, joka on esitetty kuvassa 18. Replikaattorilla voidaan neuroverkon koulutusdata jakaa useammalle GPU-suorittimille. Replikaattori-kehys hyödyntää TensorFlow'n jakelustrategiaa ja malli on suunniteltu TensorFlow'n ohjelmointimalliin perustuen. Kyseessä on varsin tehokkaan oloinen tiedon jakaja, jonka ympärille voi myös ottaa käyttöön Sonnetin optimoija ja kehittää Keras-mallista Sonnet-malli. [19;20.]

```
strategy = snt.distribute.Replicator(
    ["/device:GPU:{}".format(i) for i in range(2)],
    tf.distribute.ReductionToOneDevice("GPU:0"))
```

Kuva 18. Sonnet Replicator

TensorFlowOnSpark

Olemassa oleva TensorFlow- ja Keras-malli voidaan muuttaa TensorFlowOnSparkin kautta hajautetuksi-sovellukseksi. Kehys toimii hyvin pienimuotoisilla dataseteillä ja on tehokas työväline TensorFlow'n spesifisiin vianmäärityksiin. Kehys toimii myös ilman hajautusta ja Keras-kirjastoa. [21.]

Kuvassa 19 kehysmalliin on määritetty inputModeksi TensorFlow. Num_executors on laskentayksikön suorittimien lukumäärä. Sovelluksen toimiessa voidaan alkaa kasvattamaan klusterin kokoa ja näin kouluttaa neuroverkko suuremmalla datamassalla. [21.]

```
if __name__ == '__main__':
    import argparse
    from pyspark.context import SparkContext
    from pyspark.conf import SparkConf
    from tensorflowonspark import TFCluster

    sc.stop()
    sc = SparkContext(conf=SparkConf().setAppName("Cat&Dog_keras"))
    executors = sc._conf.get("spark.executor.instances")
    num_executors = int(executors) if executors is not None else 2

    parser = argparse.ArgumentParser()
    parser.add_argument('-f')
    parser.add_argument("--batch_size", help="number of records per batch", type=int, default=32)
    parser.add_argument("--buffer_size", help="size of shuffle buffer", type=int, default=1000)
    parser.add_argument("--cluster_size", help="number of nodes in the cluster", type=int, default=num_executors)
    parser.add_argument("--num_ps", help="number of parameter servers", type=int, default=1)
    parser.add_argument("--tensorboard", help="launch tensorboard process", action="store_true")
    parser.add_argument("--epochs", help="number of epochs", type=int, default=20)
    args = parser.parse_args()
    print("args:", args)
    cluster = TFCluster.run(sc, main_fun, args, args.cluster_size, num_ps=1, tensorboard=args.tensorboard, input_mode=TFCluster.InputMode.TENSORFLOW, master_node='chief')
    cluster.shutdown()
```

Kuva 19. Spark-klusteri

4. Tulokset

Opinnäytetyöprosessissa on tärkeää hahmottaa hankkeen vaiheet, joista muodostuu tuloksia. Kehittämishankkeen tuotoksena muodostuu jotakin uutta tietoa ja jopa uusia parempia toimintatapoja. Hyvä tavoitetila olisi, että raportti sisältää kokonaiskuvauksen kehittämistoiminnan ymmärtämisestä, omasta oppimisesta sekä ammattikorkeakoulun innovatiivisuudesta. Kaikki tuotetut aineistot ja materiaalit ovat lähtökohtaisesti yhtä tärkeitä tuloksien tulkinnassa. [24, s. 23–25.]

4.1 Tuloksien luokittelua

Määrällisessä tutkimuksessa on tärkeää kyetä hahmottamaan asiaongelma ja tärkeätä on myös kyetä se nimeämään yksilöllisesti. Näin voidaan muodostaa tutkimusongelma ja siihen liittyvät kysymykset tai mahdollinen hypoteesi. Teoria, tutkimuskohteen luonne, kieli ja muuttujat vaikuttavat tutkimukseen. Tavoitteena on saada mahdollisimman täsmällistä tietoa tuotettua. [25.]

CPU- ja GPU-ympäristöjä koulutettiin 20 epochilla (Liitteet 3–4 ja 7). Taulukossa 1 on esitetty Mini-Anaconda ympäristössä TensorFlow- ja Keras-, Horovod-, Sonnet- ja TensorFlowOnSpark-kehyksien koulutuksien mittaustuloksia. Taulukosta voidaan havaita, että CPU-prosessorien neuroverkon kouluttamiseen käytetty aika on huomattavasti suurempi, kuin GPU-ympäristöissä.

Tulokset voidaan luokitella ja suositeltava luokkien määrä on n. 4–10 kappaletta, johon vaikuttaa tilastoaineiston laajuus. Luokittelun tulisi olla tasavälistä ja tunnusluvuista avoimia luokkia tulisi välttää. Luokkaväliksi tulisi valita sopiva kokonaisluku. [26, s. 48.]

Prosessori(t) **Kehykset**	**Aika**	**accuracy**	**val_** **accuracy**	**loss**	**val_loss**
CPU **+ TensorFlow + Keras** **(586/586)**	208s 355ms/step (mean: 209.7s)	0,8686	0,8646	0,3079	0,3271
GPU **+ TensorFlow + Keras** **(586/586)**	106s 159ms/step (mean: 119.85s)	0.8638	0.8590	0.3109	0,3204
2 x GPU **+ TensorFlow + Keras** **(293/293)**	109s 327ms/step (mean: 79.4s)	0.8605	0.8693	0.3184	0.3059
2 x GPU **+ TensorFlow + Keras** **+ Horovod** **(293/293)**	74s 251ms/step (mean: 75.05s)	0,7478	0,7384	0,5105	0,5269
2 x GPU **+ TensorFlow + Keras** **+ Sonnet Replicator** **(293/293)**	73s 248ms/step (mean: 80.2s)	0,8571	0,8522	0,3280	0,3532
2 X GPU **+ TF + Keras +** **TensorFlowOnSpark** **(293/293)**	107s 366ms/step (mean: 108.6s)	0,8628	0,8415	0,3143	0,3143

Taulukko 1. Mittaustuloksia Epoch 20 MiniConda + PetImages (Liitteet 3–4&7)

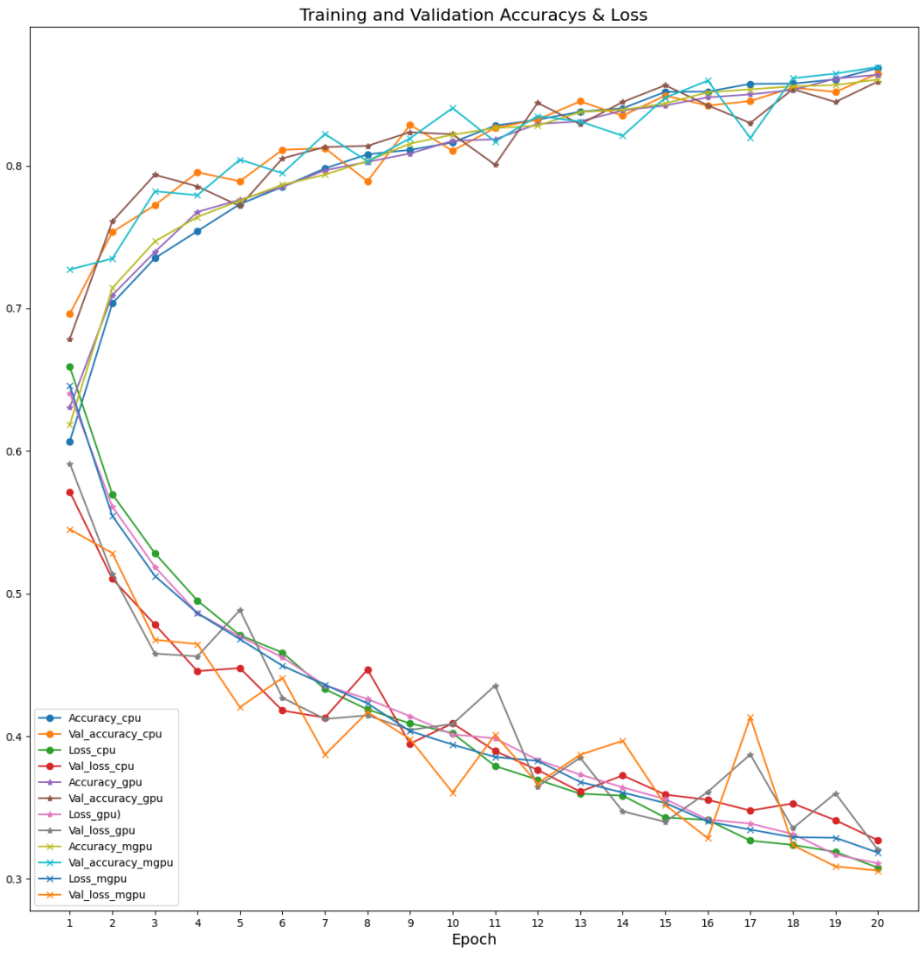
Tutkimuskysymyksiin saadaan vastauksia tutkimustuloksista ja näin niistä voidaan tehdä johtopäätöksiä [27, s. 8]. Taulukosta 1 voidaan havainnoida selkeät CPU- ja GPU-prosessorien koulutuksien ajalliset erot. Eri suorittimilla ja kehyksillä koulutuksen ja validoinnin arvot olivat saman suuntaisia. TensorFlow- ja Keras-kehyksellä ja CPU-prosessorilla saavutettiin paras tarkkuus.

Taulukosta 1 voidaan havaita, että kaikilla yhtälöillä neuroverkko saatiin koulutettu. Hyvin lähelle mittaustuloksissa päästiin yhdellä ja kahdella GPU-prosessorilla. Taulukosta voidaan havaita kehyksien integroinnilla olevan vaikutusta TensorFlow- ja Keras-neuroverkon oppimisen tarkkuuteen.

Taulukkoon 1 on lisätty aika-sarakkeeseen sulkuihin kaikkien 20 epochin koulutuksen ajallinen keskiarvo. Horovod- ja Sonnet-kehykset jopa nopeuttivat koulutusprosessia entisestään. TensorFlowOnSpark-kehyksen koulutus aika on hyvin saman suuntainen TensorFlow- ja Keras -kehyksen kanssa. (Liite 4)

4.2 TensorFlow- ja Keras-viivadiagrammina

Kuvassa 20 CPU- ja GPU-prosessorilla kouluttaminen tuottaa samannäköisiä käyriä. GPU-prosessori on ajallisesti nopeampi kouluttamaan neuroverkkoja kuin CPU-prosessori. Kuvassa 20 on käytetty viivadiagrammia, joka soveltuu hyvin erilaisten aikasarjojen tai steppien esittämiseen. Näin saadaan yleiskuva tarkkuuden muutoksesta [26, s. 56]. Taulukosta 1 voidaan havaita, että TensorFlow- ja Keras-neuroverkon kouluttamisen validointi- ja häviö- ja tarkkuusarvot ovat melko lähellä toisiansa. (Liite 3)



Kuva 20. TensorFlow ja Keras (CPU&GPU&2xGPU)

4.3 TensorFlow- ja Keras-kehyksien tunnuslukuja

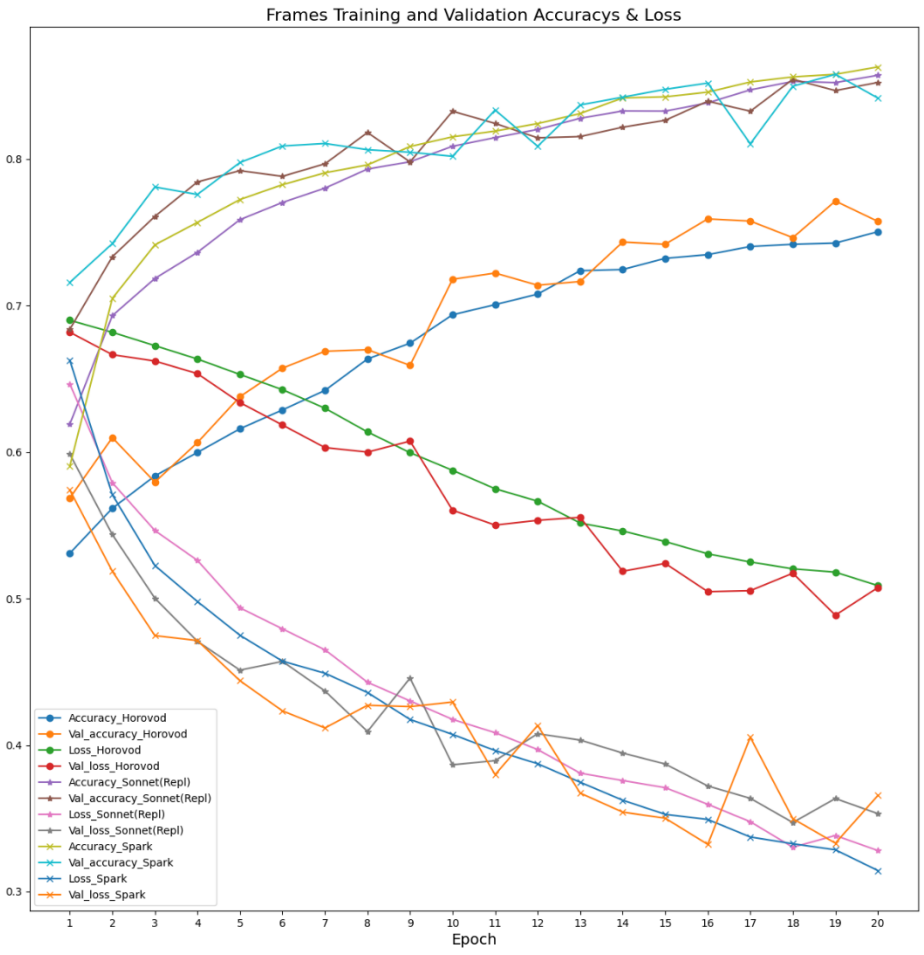
Kuvasta 21 voidaan nähdä, että 20 koulutetun epochin MultiGPU tarkkuuden (accuracy) keskiarvo on vain hieman parempi kuin yksittäisen GPU-prosessorin keskiarvotarkkuus. Puolestaan CPU-prosessorilla neuroverkon kouluttaminen tuotti parhaimman oppimistarkkuuden (Taulukko 1). Tarkkuuden keskihajonta (std) on pienin GPU-kouluttamisessa eli 0.057805 ja suurin CPU-prosessorilla kouluttamisessa 0.064634. Tähän yhtenä syynä voi olla neuroverkon kouluttamiseen käytetty aika. Kaikkien Tensorflow- ja Keras-neuroverkkojen keskiarvo on liki samansuuruinen. Keskihajonta on yleisin suhde- ja välimatka-asteikon mittaustulosten hajontaluku. Yleensä aritmeettista keskiarvoa käytetään myös yhdessä keskihajonnan kanssa. [26, s. 84.]

	Accuracy_cpu	Accuracy_gpu	Accuracy_mgpu
mean	0.803906	0.803503	0.804675
std	0.064634	0.057805	0.059534
min	0.606632	0.630553	0.618486
max	0.868593	0.863840	0.860476

Kuva 21. Tunnuslukuja (TensorFlow ja Keras)

4.4 Horovod,- Sonnet-, TensorFlowOnSpark-kehykset viivadiagrammina

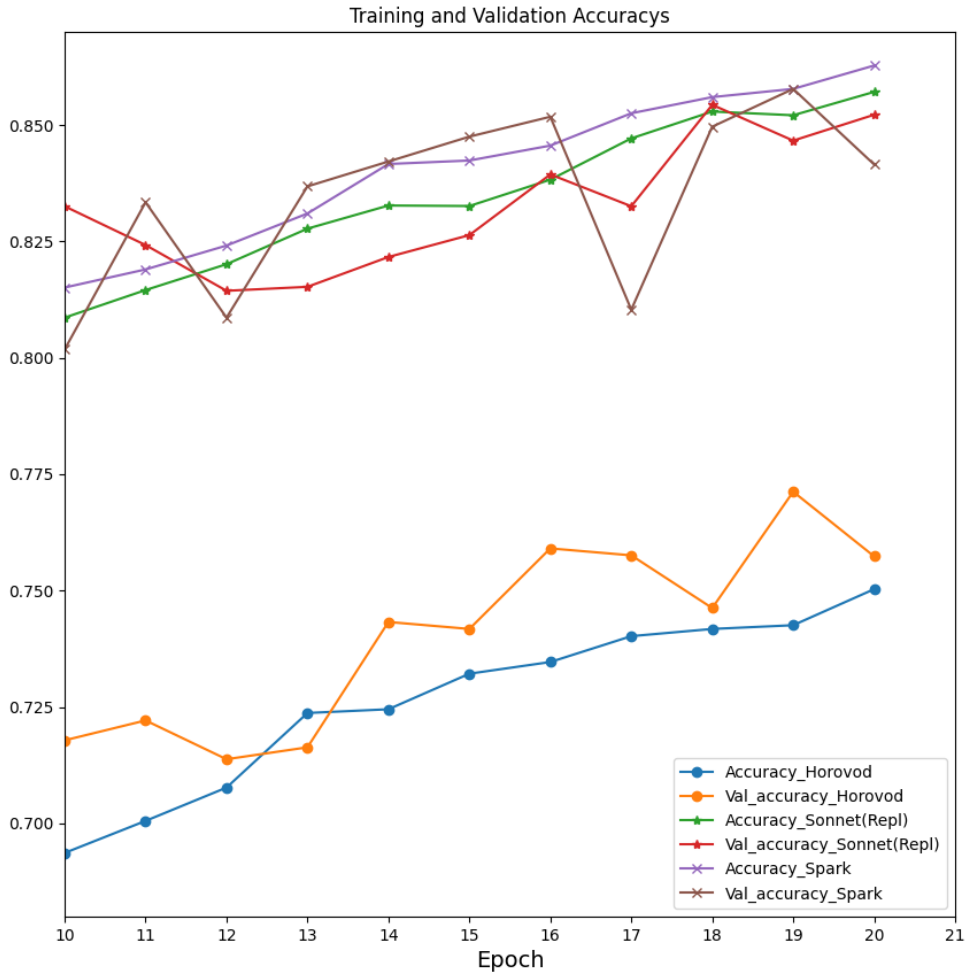
PetImages-kuvia koulutettiin Horovod-, Sonnet- ja TensorFlowOnSpark-kehyksissä tai hyödyntämällä niiden ominaisuuksia (Liite 4). Jokaista kehystä on koulutettu 20-epochilla. Taulukosta 1 voidaan havaita kehyksien välillä oppimisessa olevan eroavaisuuksia. TensorFlowOnSpark-kehys on saatujen mittauslukujen perusteella oppimistarkkuudeltaan huomattavasti parempi kuin Horovod-kehys. Kuvasta 22 voidaan nähdä eri kehyksien oppimiseen liittyvää tarkkuutta ja sen kehittymistä.



Kuva 22. Horovod- & Sonnet Replicator- & TensorFlowOnSpark -kehykset

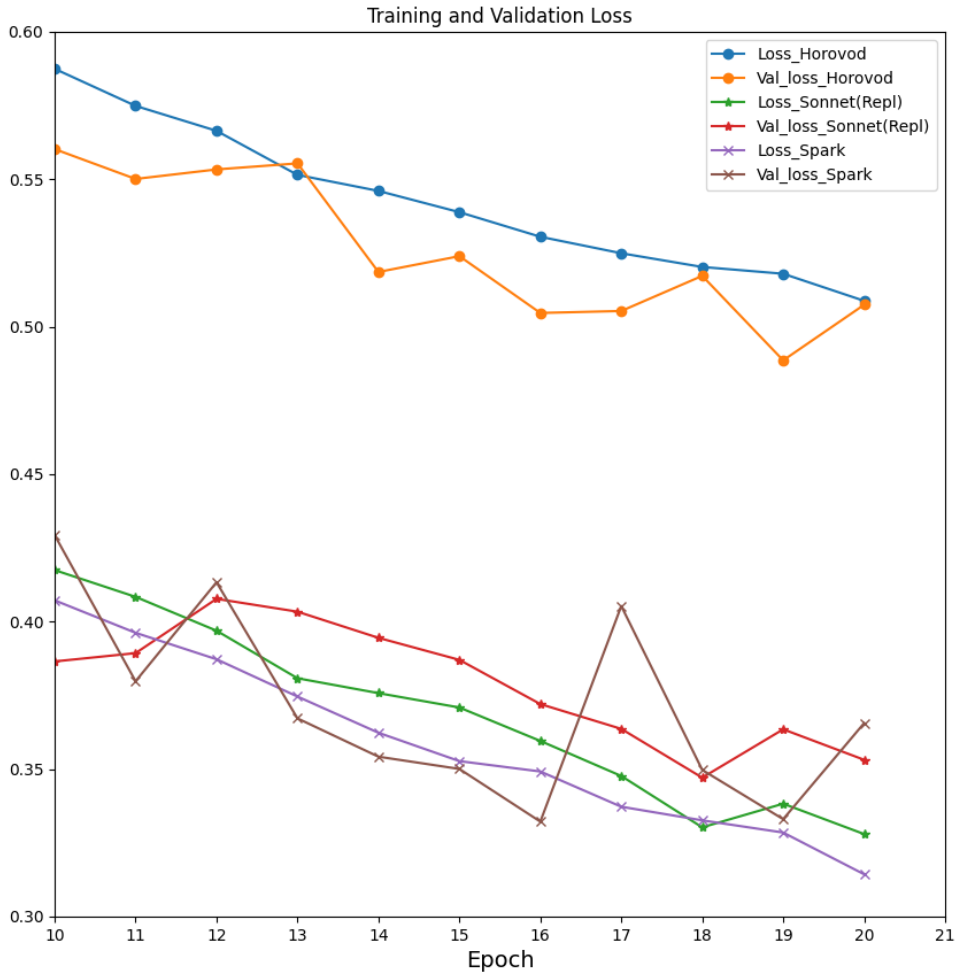
Kuvasta 23 voidaan havaita, että Horovod-kehyksen koulutus- ja validointitarkkuuden suorituskyky eroaa selkeästi Sonnetin- sekä TensorFlowOnSpark-kehyksien suorituskyvystä. Käytössä olevalla neuroverkon optimointitekniikalla on todennäköisesti vaikutusta tarkkuuteen tai tarkkuuksien kehittymiseen. Kaikkien neuroverkkojen koulutus- ja validointitarkkuus oli melko tasaisesti nousevaa.

Kuvista 22 ja 23 voidaan havaita, että tilastokuva auttaa tiivistämään tilastoaineiston keskeisiä seikkoja. Näin tilastotieteellinen fakta voidaan esittää totuudenmukaisesti ja samalla saadaan lukija ajattelemaan ja pohtimaan asiaa. Vertailu korostaa kuvion sanomaa ja samalla elävöittää opinnäytetyötä. [26, s. 64.]



Kuva 23. Epochien 10–20 tarkkuuden ja validoidun tarkkuuden arvot

Kuvassa 24 voidaan havaita neuroverkon häviön ja validointihäviön kehittyminen eri kehysten välillä. Koulutuksien oppimisen häviö- ja validointiarvojen erot ovat melko lähellä toisiansa. Horovod-kehys eroaa selkeästi muista testatuista kehyksistä. Kaikissa kehyksissä on havaittavissa neuroverkon oppimista.



Kuva 24. Epochien 10–20 häviö ja häviön validointi-arvot

4.5 Horovod,- Sonnet-, TensorFlowOnSpark-kehyksien tunnuslukuja

Kehyksien tarkkuuden tunnuslukuja on esitetty kuvassa 25. MultiGpu:lla mitattujen Sonnetin replikaattorin ja TensorFlowOnSpark-kehyksien tarkkuuden (accuracy) keskiarvo on huomattavasti parempi, kuin Horovod-kehyksellä. Horovod-kehyksen keskiarvo on välttävää tasoa. Neuroverkon oppimisen tarkkuuden (Accuracy) keskihajonta (std) on pienin Sonnetissa eli 0.061962 ja suurin Horovod-kehyksellä 0.067746. Horovod-kehyksen keskiarvo poikkeaa selkeästi myös muista kehyksistä. Kehykset ja niiden käyttäminen neuroverkon kouluttamisessa vaikuttaisivat kasvattavan oppimisen tarkkuuden keskihajontaa.

	Accuracy_Horovod	Accuracy_Sonnet(Repl)	Accuracy_Spark
mean	0.674600	0.792522	0.799522
std	0.067746	0.061962	0.065113
min	0.530863	0.618913	0.590186
max	0.750320	0.857112	0.862772

Kuva 25. Kehyksien tunnuslukuja

4.6 Kehyksien vinous ja huipukkuus

Kuvassa 26 on esitetty kehyksien Skew-arvoja. Tilastoanalyysissä voidaan tarkastella aineiston vinous- ja huipukkuusarvoja [28]. Pienin tarkkuuden vinousarvo on TensorFlowOnSpark:lla (Accuracy_Spark) eli -1.95646 ja suurin Horovod:lla eli -0.729059. Sonnetin replikaattori tarkkuuden Skew-arvo on -1.398, ja se poikkeaa selkeästi TensorFlowOnSpark:n ja Horovodin Skew-arvoista. Kaikkien tarkkuuksien vinous on samansuuntaista eli negatiivistä eli vasemmalle vinoa.

```
Accuracy_Horovod        -0.729059
Accuracy_Sonnet(Repl)   -1.398000
Accuracy_Spark          -1.956460
dtype: float64
```

Kuva 26. Skew-arvot

Kuvassa 27 on esitetty kehyksien Kurtosis-arvoja. Horovodin kehyksessä koulutuksen tarkkuuden kurtosis-arvo -0,663786 poikkeaa vertailun TensorFlowOnSparkin arvosta 4.797223 ja Sonnetin replikaattorin arvosta 1.912559. Horovod oli litteähuippuinen ja muissa kehyksissä voidaan havaita huipukkuutta. Opinnäytetyön liitteenä on kehyksien validointi-, häviö- ja tarkkuusarvot (Liite 6).

```
Accuracy_Horovod        -0.663786
Accuracy_Sonnet(Repl)    1.912559
Accuracy_Spark           4.797223
dtype: float64
```

Kuva 27. Kurtosis-arvot

4.7 Mallin todennäköisyys

Kaikilla neuroverkkokehyksillä saatiin oppimistuloksia ja todennäköisyyksiä. Horovod-kehys tuotti 57.4 % todennäköisyyden, että kuvassa on kissa. Kuvassa 28 on esitetty Horovod- neuroverkon satunnaisotoksen todennäköisyys.

```
1/1 [==============================] - 0s 190ms/step
This image most likely belongs to Cat with a 57.40 percent confidence.
```

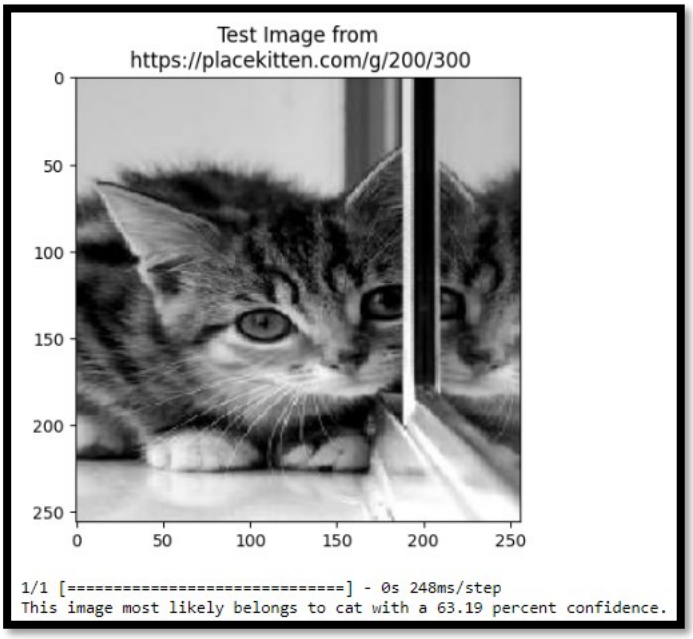
Kuva 28. Horovod-kehyksen todennäköisyys

Kuvassa 29 on esitetty Sonnetin replikaattori neuroverkon todennäköisyys. Sonnetin replikaattorilla koulutettu neuroverkko tuotti 50.05 % todennäköisyyden, että kuvassa on kissa.

```
1/1 [==============================] - 2s 2s/step
This image most likely belongs to Cat with a 50.05 percent confidence.
```

Kuva 29. Sonnet-kehyksen todennäköisyys

Kuvassa 30 näkyy kissa. TensorFlowSpark'lla koulutettu neuroverkko tuotti 63.19 % todennäköisyyden. Kehyksistä parhaimman todennäköisyyden saavutti TensorFlowOnSpark-kehys. Todennäköisyyksissä on hyödynnetty satunnaisesti valittuja Internetin kissa-kuvia [29].



Kuva 30. TensorFlowOnSpark-kehyksen todennäköisyys

5. Johtopäätelmä

Opinnäytetyössä koulutettiin neuroverkkoja yhdellä laskentayksiköllä, jossa on yksi CPU- ja kaksi GPU-prosessoria (Liite 1). Rinnakkaislaskennasta ja siihen liitettävistä neuroverkkokehyksistä voidaan tehdä tulkinta. Kehys-käsite ja sen liittäminen olemassa olevaan neuroverkkoon ei välttämättä ole aina yksiselitteinen kokonaisuus. Lisäkehyksiä voidaan lisätä ottamalla esimerkiksi käyttöön klustereita, uusi neuroverkkomalli, vaihtamalla neuroverkon optimoijaa tai replikaattori toiseen. Lähtökohtaisesti voidaan todeta, että sellaisenaan TensorFlow- ja Keras-neuroverkon koulutusprosessi toimi parhaiten.

Ensimmäinen tutkimus kysymys liittyi oppimistarkkuuden muutoksiin. Ehdottomasti suorituskykyisin vaihtoehto kehyksistä oli TensorFlowOnSparkin kehysympäristö. Opinnäytetyössä Horovodin optimoijan tai Sonetin GPU-replikoinnin lisääminen vaikuttivat jopa negatiivisesti neuroverkon oppimisen tuloksiin. Kehyksien kouluttamisen rajoittavana tekijänä oli koulutusympäristön GPU-laiteajurien vanhentuneet versiot ja laskentayksikön rauta (Liite 1 ja Taulukko 2).

Toinen tutkimuskysymys liittyi rinnakkaislaskentaan ja sen muutoksiin koulutuksien aikana. Pääsääntöisesti parhaiten toimi TensorFlow’n alkuperäinen rinnakkaislaskentamalli. Toisaalta mikäli Sonnetin malliin olisi kehitetty oma neuroverkkomalli, niin siitä olisi voinut tulla enempi kilpailija TensorFlow’n strategialle. Paljon lopputulokseen vaikuttaa myös se, että mistä lähtökohdasta lähdetään vertailemaan tarkkuuksia ja niiden muutoksien eroja. Tulevaisuudessa voi olla toki sellaisia toimintaympäristöjä yhä enempi tarjolla, joissa toimii ihan hyvin Horovod-, Sonnet- tai TensorFlowOnSpark-kehykset.

Tuloksien luotettavuutta ja toistettavuutta lisäävät, että ne on toteutettu ja mitattu samassa laskentayksikössä, samoilla koulutusdatoilla ja TensorFlow-versiolla sekä neuroverkko mallilla (Taulukko 2). Neuroverkkoja tuli jokaisella kehyksellä testattua useamman kerran. Useasti neuroverkon kouluttaminen lisäkehyksessä vaati jopa laskentayksikön uudelleenkäynnistämisen, jotta kehys alkoi toimimaan. Vanhentuneet ja stabiloituneet laitteiston GPU- ja Cuda-ajurit lisäsivät tuloksien luotettavuutta ja vertailukelpoisuutta. Tulokset ovat lähtökohtaisesti vertailukelpoisia, koska koulutus on tapahtunut hyödyntämällä samaa mallia, dataa ja TensorFlow- ja Keras-ympäristöä.

Taulukkoon 2 on koostettu tämän opinnäytetyön tuloksia. Opinnäytetyöhön olisi toki voinut kouluttaa myös enempi kehyksiä, mutta toisaalta kolmella kehyksellä sai aivan riittävän laajuisen tiedon kehyksien hyödyntämisen mahdollisuuksista neuroverkon ja rinnakkaislaskennan näkökulmasta katsottuna. Toisaalta kehyksien kouluttamista rajoittivat laskentayksikkö ja siihen yhteensopiva TensorFlow-GPU-versio.

Rinnakkaislaskenta + Kehys	**Yhteenveto(a)**
TensorFlow + Keras **(CPU & GPU & 2 x GPU)**	• Hyödynnetty pohja-templatena muille kehyksille (Optimoija Adam, Keras-malli (Liite 2), PetImages)
TensorFlow + Keras + Horovod **(2 x GPU)**	• hvd.init() • Optimoijana käytettiin Legacyn SGD()
TensorFlow + Keras + Sonnet **(2 x GPU)**	• Strategiana käytettiin Sonnetin replikaattoria
TensorFlow + Keras + TensorFlowOnSpark **(2 x GPU)**	• Koulutettu Spark-klusterissa if__name__ funktiorakenteella (kuva 19).
Testattu AnalyticsZoo, BigDL, Elephas, Mesh, Petastorm, Singa – kehyksiä (Kuva 7).	• Laskentayksikön näytönohjaimien stabiloitunut laitteistoajurin versio ja siihen yhteensopiva TensorFlow-GPU rajoittivat kehyksien kouluttamista ja koodaamista. • Opinnäytetyössä koulutettiin vain TensorFlow- ja Keras yhteensopivia kehyksiä samalla mallilla ja PetImages-kuvilla • Koulutettavan kehyksen tuli tukea useammalla GPU-suorittimella kouluttamista.

Taulukko 2. Yhteenvetotaulukko

Tämä opinnäytetyö auttoi ymmärtämään syvemmin rinnakkaislaskennan hyödyntämisen mahdollisuuksia laskentayksikön näkökulmasta katsottuna. Opinnäytetyö oli hyvin opettavainen kokonaisuus syväoppimiseen ja auttoi ymmärtämään neuroverkkojen toimintaa ja toimintaperiaatteita. Yleisesti ottaen rinnakkaislaskennasta on havaittavissa ainakin nopeudellinen hyöty ja samalla saavutetaan riittävän laadukkaita oppimistuloksia. Opinnäytetyö vaati perehtymistä neuroverkkojen koodaamiseen ja kehyksien lisääminen sekä niiden testaaminen olivat jokainen oma koodausprosessi. Koodi muutoksien määrä vaihteli käytettävästä kehyksestä tai kehyksen lisäominaisuuden mukaan. Opinnäytetyöhön tuli ohjelmoitua useita satoja rivejä koodia, joita on tiivistetty tämän työn liitteiksi (Liite 2-8).

Tästä opinnäytetyöstä saa hyviä vinkkejä toteuttaa ja kehittää hajautettua laskentaa erilaisissa kehyksissä ja toimintaympäristöissä. Opinnäytetyö lisää ymmärrystä supertietokoneen toiminnasta. Erilaisten supertietokoneiden ja niiden laskentayksiköiden välillä voisi tehdä lisää vertailua neuroverkon oppimisen tuloksissa. Kehyksiä voisi lisää tutkia esimerkiksi selvittämällä Keras-mallin muuttamista muihin malleihin ja kehyksiin paremmin yhteensopivammaksi.

Laskentayksiköiden välinen Infiniband-yhteys eli saman aikaisesti useampaa hajautettua laskentayksikön kouluttaminen jäi testaamatta, koska käytössä oli vain yksi laskentayksikkö. Supertietokoneella olisi voinut kouluttaa useammalla laskentayksiköllä ja vertailla niiden tuloksia esimerkiksi keskenään tai tarkemmin erilaisten kehyksien välillä.

Maailma on sen verran iso paikka, että isompien datamassojen kouluttamiseen tarvitaan todella paljon laskentatehoa. Rinnakkaislaskenta ja neuroverkon kehykset vaikuttavat yhä enemmissä määrin tulevaisuudessa neuroverkkojen oppimisprosessiin. Näkisin myös, että opinnäytetyö lähestyi aihetta kestävän kehityksen näkökulmasta. Opinnäytetyöstä saa hyviä vinkkejä pidentää supertietokoneiden käyttöikää ja jopa keksiä niille uusia käyttötarkoituksia erilaisissa muuttuvissa toimintaympäristöissä.

Lähteet

1 Miras, J. Fast diferential box-counting algorithm on GPU. [Internet]. 2020;76:204–225. Saatavilla: doi: 10.1007/s11227-019-03030-1

2 Ihonen, A. & University, T. (2022). Nvidian GPGPU-tekniikka: Tekninen katsaus CUDA-ohjelmointialustaan. https://urn.fi/URN:NBN:fi:tuni-202203162570

3 Wei Min, Z. Long, Z. Zheyu, Z. Mingjun, S. IBD1: The metrics and evaluation method for DNN processor benchmark while doing Inference task. [Internet]. 2021; Journal of Intelligent & Fuzzy Systems. 2021, Vol. 40 Issue 5, p9949-9961. Saatavilla: doi: 10.3233/JIFS-202552.

4 Fuli, B, Chong, W. A Method for Evaluating the Quality of Mathematics Education Based on Artificial Neural Network. [Internet]. 2022; Computational & Mathematical Methods in Medicine. 8/12/2022, p1-11. 11p. Saatavilla: doi: 10.1155/2022/6976654

5 Xiao, L. Luo, Y. The Application of RBF Neural Network Model Based on Deep Learning for Flower Pattern Design in Art Teaching. [Internet]. 2022; Article ID 4206857, 9 pages. Saatavilla: doi: 10.1155/2022/4206857

6 Neuroverkot. Saatavilla 4.3.2023. https://fi.wikipedia.org/wiki/Neuroverkot

7 A Deeper Look at Deep-Learning Frameworks. *Electronic Design*, vol. 64, no. 8, 2016.

8 Edukamu: Johdatus supertietokoneisiin – Rinnakkaislaskennan teknisempipuoli. [viitattu 20.10.2022]. Saatavilla: https://edukamu.fi/elements-of-supercomputing-fi/shared-memory-computer

9 GPUs and Deep Learning. Saatavilla 8.7.2022. https://www.electronicdesign.com/markets/robotics/article/21801698/gpus-and-deep-learning

10 Bostrom, N. (2016). Superintelligence: Paths, dangers, strategies (Paperback edition.). Oxford University Press.

11 Kaggle. Dogs vs. Cats. Create an algorithm to distinguish dogs from cats. Saatavilla 6.2.2023. https://www.kaggle.com/competitions/dogs-vs-cats/overview

12 TensorFlow. Use a GPU. Saatavilla 10.2.2023. https://www.tensorflow.org/guide/gpu

13 Kaggle. Cat vs Dogs Image Classification. Saatavilla 14.10.2022. https://www.kaggle.com/code/mauricioasperti/cats-vs-dogs-image-classification

14 Lindevs. Classify Images of Dogs and Cats using CNN and TensorFlow 2. Saatavilla 20.10.2022. https://lindevs.com/classify-images-of-dogs-and-cats-using-cnn-and-tensorflow-2

15 Nvidia Developer. CUDA Pro Tip: Control GPU Visibility with CUDA_VISIBLE_DEVICES. Saatavilla 13.2.2023. https://developer.nvidia.com/blog/cuda-pro-tip-control-gpu-visibility-cuda_visible_devices/#:~:text=To%20use%20it%2C%20set%20CUDA_VISIBLE_DEVICES,them%20in%20a%20specific%20order.

16 TensorFlow. Distributed training with TensorFlow. Saatavilla 6.6.2022. https://www.tensorflow.org/guide/distributed_training

17 TensorFlow. tf.distribute.Strategy. Saatavilla 13.2.2023. https://www.tensorflow.org/api_docs/python/tf/distribute/Strategy

18 Mudugandla, S. 10 Python Frameworks for Parallel and Distributed Machine Learning Tasks. Saatavilla 6.6.2022. https://medium.com/swlh/10-python-frameworks-for-parallel-and-distributed-machine-learning-tasks-c0215269cfd

19 Sonnet. Sonnet Documentation. Saatavilla 13.2.2023. https://sonnet.readthedocs.io/en/latest/index.html

20 Sonnet. Replicator. Saatavilla 13.2.2023. https://sonnet.readthedocs.io/en/latest/api.html#replicator

21 GitHub. Conversion Guide. Saatavilla 13.2.2023. https://github.com/yahoo/TensorFlowOnSpark/wiki/Conversion-Guide

22 GitHub. Horovod with Keras. Saatavilla 13.2.2023. https://github.com/horovod/horovod/blob/master/docs/keras.rst

23 TensorFlow. tf.keras.optimizers.legacy.Optimizer. Saatavilla 13.2.2023. https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/legacy/Optimizer

24 Salonen, K. (2013). Näkökulmia tutkimukselliseen ja toiminnalliseen opinnäytetyöhön: Opas opiskelijoille, opettajille ja TKI-henkilöstölle. Turun ammattikorkeakoulu.

25 Vilkka, H. (2021) Näin Onnistut Opinnäytetyössä: Ratkaisut Tutkimuksen Umpikujiin. PS-kustannus.

26 Holopainen, M., & Pulkkinen, P. (2002). Tilastolliset menetelmät. WSOY.

27 Kananen, J. (2019). Opinnäytetyön ja pro gradun pikaopas: Avain opinnäytetyön ja pro gradun kirjoittamiseen. Jyväskylän ammattikorkeakoulu.

28 Mv.Helsinki.fi. Tilastolliset tunnusluvut (lyhyt kertaus). Saatavilla 10.2.2023. https://www.mv.helsinki.fi/home/hotulain/Tilasto/TSTO221107.pdf

29 Kaggle. Transfer Learning using Keras. Saatavilla 28.2.2023. https://www.kaggle.com/code/selcukcan/dl-4-2-transfer-learning-using-keras

Bull-supertietokone
Sana supertietokone tarkoittaa monelle vain yhtä hyvin nopeaa tietokonetta. Todellisuudessa kyse ei kumminkaan ole yksittäisestä laitteesta, vaan kyseessä on usean nopean tietokoneen yhdistelmä.

Bull-supertietokoneen hyödyntäminen projekti- tai opetuskäytössä
Bull-supertietokoneen käyttö projekteissa, opinnäytetöissä ja opetuksessa on herättänyt kiinnostusta, mutta on ollut hieman epäselvää, kuinka superia pääsee hyödyntämään. Normaalissa tilassa pidämme vain toisen puoliskon supertietokoneesta käynnissä. Tämä johtuu siitä, että toinen räkeistä sisältää Intel Xeon Phi -laskentakortteja, joiden ajurituki on päättynyt. Lisäksi heikon varaosien saatavuuden takia on parasta pitää toinen laiteräkeistä varalla. Pyydettäessä myös Xeon Phi -kortillisia bladeja on mahdollista ottaa käyttöön projekteihin. Alla lyhyt kuvaus laitteistosta ja huomioitavista asioista, jos haluat käyttää supertietokonetta hyödyksi projekteissa/opinnoissa.

Blade sisältää
- 2x Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz
- 64GB RAM
- 2x NVIDIA Tesla K40t
- 200GB vapaata tallennustilaa
- Käyttöjärjestelmänä: Ubuntu 20.04
- NVIDIA ajurit (jos käytössä NVIDIA-blade)
- Etäyhteys: SSH

Mitkä asiat asiakkaan tulisi vähintään tietää
- Minkä hankkeen/projektin käyttöön blade tulee
- Tarvitsetko yli 200GB tallennustilaa datalle ja laskentatuloksille
- Tarvitsetko GPU-laskentaa (eli Tesla K40t-laskentakortteja)
- Kuinka pitkäksi aikaa blade on varattava käyttöösi
- Ylläpito ei tee töitä puolestasi, vaan tarjoaa alustan töiden tekoon
- Asiakas on itse vastuussa töidensä varmuuskopioinnista.
- Asiakas on vastuussa bladensa tietoturvasta (esim. vahvat salasanat etäyhteyksiin).

Mitkä asiat asiakkaan olisi hyvä tietää
- Onko jossakin vaiheessa tarkoituksena hajauttaa laskentaa useammalle bladelle
- Tarvitsetko jotain tiettyjä kirjastoja/ohjelmistoja valmiiksi asennettuna
- Tarvitsetko SSH-yhteyden lisäksi jotain muita porttiohjauksia ulkoverkkoon

Mitä me ylläpidossa teemme
- Asennamme käyttöjärjestelmän ja ajurit valmiiksi
- Ohjaamme SSH-yhteyden käyttämällesi bladelle
- Voimme antaa vinkkejä sopivista ohjelmista/kirjastoista projektiisi
- Autamme kirjasto/ohjelmistoyhteensopivuusongelmien kanssa
- Ohjaamme tarvittaessa muuta verkkoliikennettä bladelle

Opinnäytetyön neuroverkko malli:

PetImages (Kaggle) koulutus- ja testidata + kehykset (Liitteet 3&4)

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 180, 180, 3)       0

 rescaling_2 (Rescaling)     (None, 180, 180, 3)       0

 conv2d_6 (Conv2D)           (None, 180, 180, 16)      448

 max_pooling2d_6 (MaxPooling  (None, 90, 90, 16)       0
 2D)

 conv2d_7 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_7 (MaxPooling  (None, 45, 45, 32)       0
 2D)

 conv2d_8 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 22, 22, 64)        0

 flatten_2 (Flatten)         (None, 30976)             0

 dense_4 (Dense)             (None, 128)               3965056

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 3,988,769
Trainable params: 3,988,769
Non-trainable params: 0
_________________________________________________________________
```

Neuroverkon oppimistuloksia MiniConda (TensorFlow ja Keras / MiniConda(PetImages)[11]):

Koulutettu 3.2.2023 Kamk Bull / blade54-nvidia.

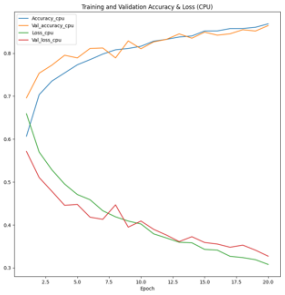
Train Dataset: https://www.kaggle.com/competitions/dogs-vs-cats/overview

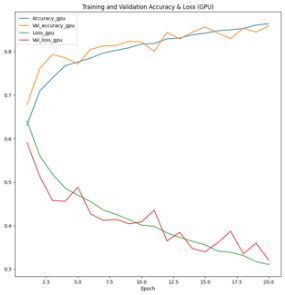
Mittaus- ja tilastotuloksia

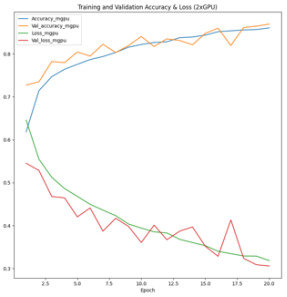
	Accuracy_cpu	Val_accuracy_cpu	Loss_cpu	Val_loss_cpu	Accuracy_gpu	Val_accuracy_gpu	Loss_gpu	Val_loss_gpu	Accuracy_mgpu	Val_accuracy_mgpu	Loss_mgpu	Val_loss_mgpu
0	0.606632	0.696070	0.659094	0.571397	0.630553	0.678556	0.640638	0.591337	0.618486	0.727253	0.645777	0.545127
1	0.703492	0.753524	0.569732	0.510414	0.709259	0.761000	0.561043	0.513703	0.714278	0.734942	0.554546	0.528376
2	0.735209	0.772319	0.528382	0.478372	0.739534	0.793678	0.518815	0.458003	0.746956	0.782144	0.512452	0.467712
3	0.754165	0.795387	0.495211	0.445705	0.767621	0.785562	0.486384	0.456067	0.764097	0.779368	0.486334	0.464668
4	0.773120	0.789193	0.470865	0.447834	0.776164	0.771679	0.469924	0.488505	0.775577	0.804357	0.468046	0.420368
5	0.785188	0.811192	0.458696	0.418031	0.785188	0.805211	0.455446	0.427009	0.786683	0.794746	0.449460	0.441027
6	0.798163	0.812260	0.433050	0.413084	0.796828	0.813114	0.435728	0.412123	0.793785	0.822298	0.436095	0.387110
7	0.808041	0.789193	0.418832	0.446790	0.802702	0.813968	0.426142	0.414707	0.803343	0.803076	0.423103	0.416965
8	0.811138	0.828706	0.409032	0.394809	0.808629	0.823580	0.414046	0.404474	0.815517	0.819308	0.403645	0.398070
9	0.816318	0.810551	0.402215	0.409085	0.817439	0.822085	0.401328	0.408620	0.821764	0.840453	0.394189	0.360601
10	0.828118	0.826570	0.379161	0.389693	0.818614	0.800726	0.398542	0.435602	0.826463	0.816959	0.385509	0.401076
11	0.832337	0.832337	0.369576	0.376425	0.829293	0.844084	0.383393	0.364818	0.828118	0.834900	0.382782	0.367032
12	0.837890	0.845152	0.359743	0.361201	0.831109	0.829133	0.373197	0.384970	0.837676	0.831055	0.368004	0.387060
13	0.840293	0.835113	0.358418	0.372548	0.838798	0.844724	0.364229	0.347393	0.839278	0.821017	0.360564	0.396857
14	0.851826	0.849210	0.343020	0.359123	0.842268	0.856472	0.355948	0.340025	0.843657	0.847715	0.353253	0.351910
15	0.851986	0.842161	0.341328	0.355498	0.847928	0.842375	0.341632	0.361028	0.851506	0.859675	0.340244	0.328506
16	0.857433	0.845365	0.326871	0.347950	0.850064	0.829987	0.338814	0.387295	0.853588	0.819522	0.334624	0.413264
17	0.857593	0.854763	0.323849	0.352925	0.853108	0.853695	0.331482	0.335527	0.855671	0.861384	0.329225	0.323655
18	0.860583	0.851773	0.318978	0.341166	0.861117	0.844724	0.317109	0.360031	0.856578	0.864588	0.328882	0.308755
19	0.868593	0.864588	0.307941	0.327143	0.863840	0.859035	0.310946	0.320431	0.860476	0.869287	0.318444	0.305880

	Accuracy_cpu	Val_accuracy_cpu	Loss_cpu	Val_loss_cpu	Accuracy_gpu	Val_accuracy_gpu	Loss_gpu	Val_loss_gpu	Accuracy_mgpu	Val_accuracy_mgpu	Loss_mgpu	Val_loss_mgpu
count	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000
mean	0.803906	0.815271	0.413700	0.405960	0.803503	0.813669	0.416239	0.410583	0.804675	0.816702	0.413759	0.400701
std	0.064634	0.040821	0.093279	0.062593	0.057805	0.042343	0.086186	0.067156	0.059534	0.039151	0.085655	0.065809
min	0.606632	0.696070	0.307941	0.327143	0.630553	0.678556	0.310946	0.320431	0.618486	0.727253	0.318444	0.305880
25%	0.782171	0.793838	0.342597	0.358217	0.782932	0.798964	0.352369	0.360779	0.783906	0.800993	0.350000	0.358429
50%	0.822218	0.827638	0.390688	0.392251	0.818026	0.822832	0.399935	0.406547	0.824114	0.820269	0.389849	0.397463
75%	0.851866	0.845205	0.461738	0.445976	0.843683	0.844244	0.459065	0.440718	0.845619	0.842268	0.454107	0.425533
max	0.868593	0.864588	0.659094	0.571397	0.863840	0.859035	0.640638	0.591337	0.860476	0.869287	0.645777	0.545127

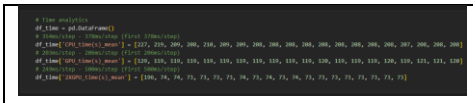
Plotteja TensorFlow ja Keras (CPU, GPU & 2xGPU)







Koulutuksien aika-analyysi



	CPU_time(s)_mean	GPU_time(s)_mean	2XGPU_time(s)_mean
mean	209.70	119.85	79.40

Neuroverkon kehyksien oppimistuloksia (MiniConda (PetImages)[11]):

Koulutettu 3–13.2.2023 (Kamk Bull / blade54-nvidia

Mittaus- ja tilastotuloksia Horovod / Sonnet Replicator/ TensorFlowOnSpark

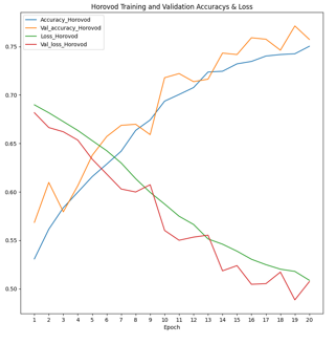
	Accuracy_Horovod	Val_accuracy_Horovod	Loss_Horovod	Val_loss_Horovod
0	0.530863	0.568560	0.689888	0.681735
1	0.561566	0.609782	0.681680	0.666413
2	0.583672	0.579453	0.672583	0.662084
3	0.599690	0.606365	0.663454	0.653478
4	0.615816	0.637762	0.652908	0.633749
5	0.628684	0.657198	0.642535	0.618400
6	0.642033	0.668731	0.629847	0.603010
7	0.663499	0.669799	0.613668	0.599938
8	0.674231	0.659120	0.599549	0.607422
9	0.693667	0.717856	0.587482	0.560253
10	0.700555	0.722127	0.574912	0.550130
11	0.707710	0.713798	0.566444	0.553361
12	0.723729	0.716361	0.551509	0.555407
13	0.724530	0.743272	0.546073	0.518587
14	0.732166	0.741777	0.538863	0.523950
15	0.734675	0.759077	0.530525	0.504688
16	0.740229	0.757582	0.524936	0.505355
17	0.741777	0.746262	0.520283	0.517266
18	0.742578	0.771252	0.517983	0.488617
19	0.750320	0.757369	0.508803	0.507468

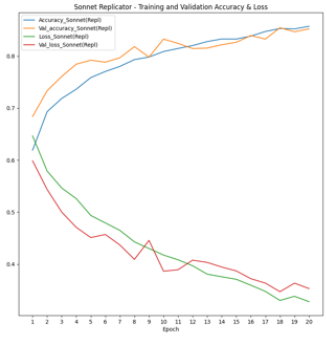
	Accuracy_Sonnet(Repl)	Val_accuracy_Sonnet(Repl)	Loss_Sonnet(Repl)	Val_loss_Sonnet(Repl)
0	0.618913	0.683682	0.646513	0.598589
1	0.693026	0.733234	0.579042	0.543660
2	0.718390	0.760786	0.546404	0.500204
3	0.736224	0.784280	0.526042	0.470820
4	0.758543	0.791969	0.493650	0.451137
5	0.770290	0.788125	0.479297	0.457064
6	0.780062	0.796668	0.464983	0.436889
7	0.793091	0.818026	0.442851	0.409248
8	0.798056	0.797950	0.430136	0.445697
9	0.808629	0.832550	0.417504	0.386525
10	0.814556	0.824220	0.408423	0.389340
11	0.820109	0.814396	0.396948	0.407750
12	0.827745	0.815250	0.380824	0.403396
13	0.832710	0.821657	0.375779	0.394529
14	0.832604	0.826356	0.370906	0.387054
15	0.838264	0.839385	0.359568	0.372070
16	0.847127	0.832550	0.347680	0.363652
17	0.852894	0.854336	0.330269	0.347115
18	0.852093	0.846647	0.338286	0.363507
19	0.857112	0.852200	0.327983	0.353170

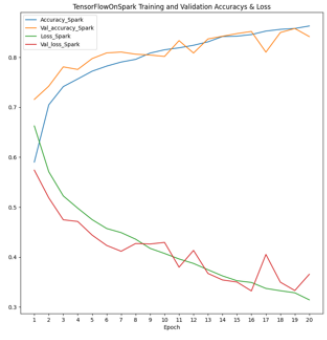
	Accuracy_Spark	Val_accuracy_Spark	Loss_Spark	Val_loss_Spark
0	0.590186	0.715720	0.662667	0.574085
1	0.704827	0.742204	0.570881	0.518636
2	0.741350	0.780863	0.522521	0.474702
3	0.756674	0.775737	0.497978	0.471300
4	0.772266	0.797522	0.474931	0.443930
5	0.782411	0.808842	0.457233	0.423323
6	0.790528	0.810551	0.449006	0.411663
7	0.795921	0.806279	0.435846	0.427162
8	0.808469	0.804571	0.417466	0.426222
9	0.815090	0.801794	0.407249	0.429362
10	0.818988	0.833405	0.396239	0.379807
11	0.824114	0.808629	0.387259	0.413369
12	0.831002	0.836822	0.374620	0.367217
13	0.841628	0.842161	0.362349	0.354231
14	0.842375	0.847501	0.352721	0.350071
15	0.845579	0.851773	0.349208	0.332209
16	0.852520	0.810337	0.337255	0.405308
17	0.855991	0.849637	0.332637	0.349744
18	0.857753	0.857753	0.328504	0.333038
19	0.862772	0.841521	0.314348	0.365617

	Accuracy_Horovod	Val_accuracy_Horovod	Loss_Horovod	Val_loss_Horovod	Accuracy_Sonnet(Repl)	Val_accuracy_Sonnet(Repl)	Loss_Sonnet(Repl)	Val_loss_Sonnet(Repl)	Accuracy_Spark	Val_accuracy_Spark	Loss_Spark	Val_loss_Spark
count	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000	20.000000
mean	0.674600	0.690175	0.590696	0.575565	0.792522	0.805713	0.433154	0.424071	0.799522	0.811181	0.421546	0.412550
std	0.067746	0.064175	0.061031	0.062367	0.061962	0.041914	0.089031	0.065669	0.065113	0.037026	0.090155	0.063186
min	0.530863	0.568560	0.508803	0.488617	0.618913	0.683682	0.327983	0.347115	0.590186	0.715720	0.314348	0.332209
25%	0.625467	0.652339	0.536778	0.518256	0.767354	0.791008	0.368071	0.382911	0.779875	0.800726	0.351843	0.362771
50%	0.697111	0.715079	0.581197	0.557830	0.811592	0.816638	0.412963	0.405573	0.817039	0.809590	0.401744	0.412516
75%	0.732793	0.744020	0.645129	0.622237	0.834099	0.832550	0.482885	0.452618	0.843176	0.841681	0.461658	0.433004
max	0.750320	0.771252	0.689888	0.681735	0.857112	0.854336	0.646513	0.598589	0.862772	0.857753	0.662667	0.574085

Plotteja Horovod / Sonnet Replicator / TensorFlowOnSpark







Koulutuksien aika-analyysi

```
# Time analytics
df_time = pd.DataFrame()
# 251ms/step - 354ms/step (first 250ms/step)
df_time['Horovod_time(s)_mean'] = [93, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 74, 75, 75, 74]
# 267ms/step - 250ms/step (first 481ms/step)
df_time['Sonnet_time(s)_mean'] = [218, 73, 73, 73, 73, 73, 73, 73, 73, 73, 72, 73, 73, 73, 73, 73, 73, 73]
# 365ms/step - 378ms/step (first 374ms/step)
df_time['Spark_time(s)_mean'] = [125, 109, 107, 107, 108, 107, 108, 107, 107, 108, 108, 107, 108, 108, 109, 108, 108, 108, 108, 107]
```

	Horovod_time(s)_mean	Sonnet_time(s)_mean	Spark_time(s)_mean
mean	75.05	80.20	108.60

Docker-compose vs. MiniConda

Docker Compose (TF+Keras)	HUOM! HUOM! Tämä tehty tfds-datalla.
Mitattu 17.1.2023	Dataset: tensorflow dataset
Datan lähde: https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_5340.zip	

Datan lähde: https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_5340.zip

TF+Keras+CPU	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7	Epoch8	Epoch9	Epoch10
loss	0,6568	0,5268	0,4246	0,334	0,2298	0,1171	0,0626	0,0351	0,0142	0,0345
val_loss	0,5686	0,4876	0,4901	0,4833	0,5409	0,7361	0,749	0,9452	1,2144	1,1104
accuracy	0,62	0,7402	0,8027	0,8552	0,9036	0,9549	0,9788	0,9885	0,9964	0,9877
val_accuracy	0,7144	0,7727	0,7887	0,7911	0,7904	0,768	0,7942	0,7878	0,7895	0,7918

TF+Keras+GPU	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7	Epoch8	Epoch9	Epoch10
loss	0,5811	0,4579	0,3764	0,294	0,1971	0,1072	0,0483	0,0389	0,0194	0,0217
val_loss	0,5227	0,4525	0,4623	0,4487	0,4754	0,6197	0,7802	0,7426	0,9542	0,9852
accuracy	0,6847	0,7855	0,8306	0,8737	0,9184	0,9612	0,9834	0,9872	0,9942	0,994
val_accuracy	0,7505	0,7822	0,7735	0,8038	0,8222	0,8212	0,8041	0,8153	0,8109	0,8195

TF+Keras+2XGPU	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7	Epoch8	Epoch9	Epoch10
loss	0,6529	0,5015	0,4308	0,3693	0,2958	0,2323	0,1644	0,0981	0,0764	0,037
val_loss	0,5549	0,4889	0,4651	0,4886	0,4424	0,4733	0,5977	0,6413	0,7054	0,9158
accuracy	0,627	0,7527	0,7982	0,8326	0,8708	0,903	0,9363	0,9647	0,9724	0,9895
val_accuracy	0,716	0,7601	0,7792	0,7792	0,8066	0,8156	0,7967	0,8106	0,809	0,8097

```yaml
version: '3.3'
services:
  jupyter:
    image: tensorflow/tensorflow:2.2.3-gpu-py3-jupyter
    ports:
      - "8889:8888"
    volumes:
      - "./:/tf/notebooks"
    environment:
      - JUPYTER_ENABLE_LAB=yes
      - GRANT_SUDO=yes
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

Docker docs: Enabling GPU access with Compose. https://docs.docker.com/compose/gpu-support/

MiniConda (TF+Keras)	HUOM! Tämä tehty tfds-datalla.
Mitattu 17.1.2023	Dataset: tensorflow dataset
Datan lähde: https://download.microsoft.com/download/3/E/1/3E1C3F21-ECDB-4869-8368-6DEBA77B919F/kagglecatsanddogs_5340.zip	

TF+Keras+CPU	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7	Epoch8	Epoch9	Epoch10
loss	0,6255	0,4952	0,4191	0,3406	0,2454	0,137	0,0651	0,0379	0,0331	0,0122
val_loss	0,536	0,4727	0,4461	0,4306	0,4987	0,5714	0,6825	0,8459	0,9861	1,0845
accuracy	0,6418	0,7599	0,8068	0,8487	0,8973	0,9472	0,978	0,9877	0,9892	0,9967
val_accuracy	0,728	0,777	0,7904	0,8084	0,8084	0,8111	0,7994	0,8043	0,7957	0,8051

TF+Keras+GPU	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7	Epoch8	Epoch9	Epoch10
loss	0,6034	0,4712	0,3832	0,3052	0,1968	0,0929	0,0498	0,0245	0,0198	0,0176
val_loss	0,5352	0,4457	0,4066	0,4115	0,4753	0,8113	0,7358	0,8559	0,8423	0,9501
accuracy	0,6666	0,7747	0,8271	0,8683	0,9208	0,9673	0,9825	0,9919	0,9937	0,994
val_accuracy	0,7325	0,7908	0,8116	0,816	0,815	0,7823	0,8206	0,8176	0,8133	0,8195

TF+Keras+2XGPU	Epoch1	Epoch2	Epoch3	Epoch4	Epoch5	Epoch6	Epoch7	Epoch8	Epoch9	Epoch10
loss	0,6159	0,5026	0,4403	0,3823	0,2989	0,2158	0,1268	0,0806	0,048	0,0322
val_loss	0,5206	0,5032	0,4591	0,4459	0,5076	0,5582	0,7843	0,7374	0,8106	0,9734
accuracy	0,6645	0,7542	0,794	0,8264	0,8713	0,9116	0,9515	0,9716	0,984	0,9894
val_accuracy	0,7402	0,7548	0,7786	0,7962	0,7866	0,7884	0,773	0,7931	0,7869	0,7762

PetImages Neuroverkon kehyksien Skew- & Kurtosis-arvoja

Skew-arvoja:

```
Accuracy_cpu          -1.746510
Val_accuracy_cpu      -1.455187
Loss_cpu               1.175297
Val_loss_cpu           1.134651
Accuracy_gpu          -1.658156
Val_accuracy_gpu      -1.819759
Loss_gpu               1.084279
Val_loss_gpu           1.073680
Accuracy_mgpu         -1.843614
Val_accuracy_mgpu     -0.871455
Loss_mgpu              1.241261
Val_loss_mgpu          0.625511
dtype: float64
```

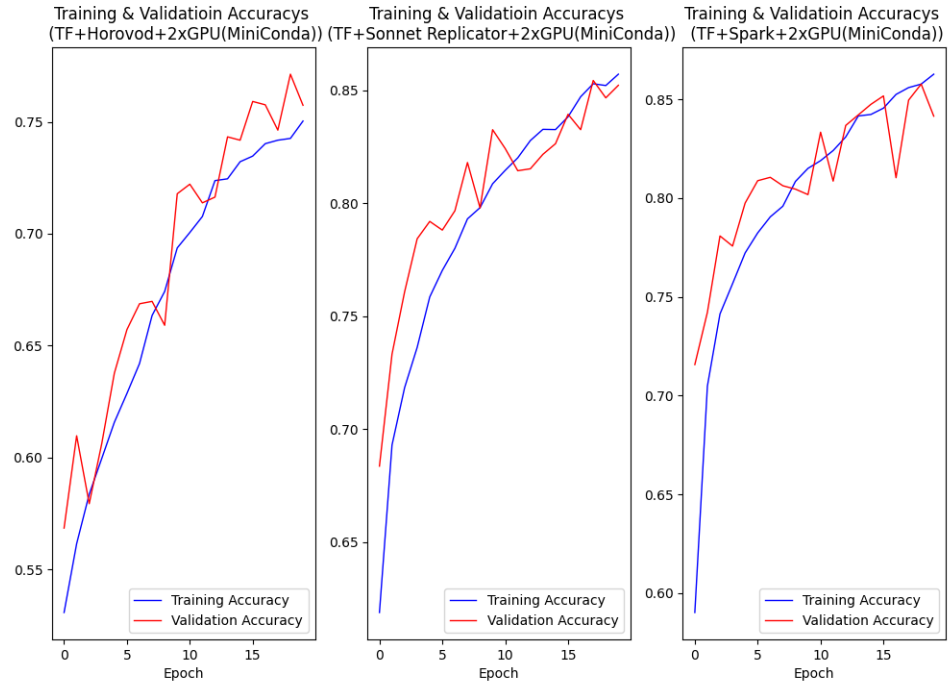
```
Accuracy_Horovod             -0.729059
Val_accuracy_Horovod         -0.547471
Loss_Horovod                  0.263613
Val_loss_Horovod              0.283256
Accuracy_Sonnet(Repl)        -1.398000
Val_accuracy_Sonnet(Repl)    -1.528332
Loss_Sonnet(Repl)             0.882965
Val_loss_Sonnet(Repl)         1.250079
Accuracy_Spark               -1.956460
Val_accuracy_Spark           -1.055455
Loss_Spark                    1.167744
Val_loss_Spark                0.915949
dtype: float64
```

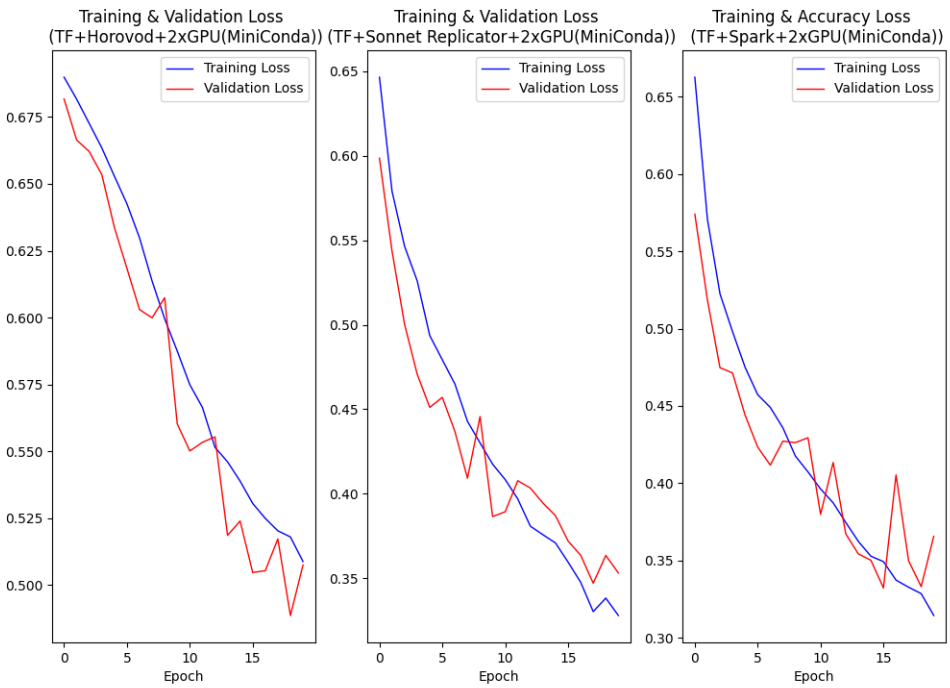
Kurtosis-arvoja:

```
Accuracy_cpu          3.517064
Val_accuracy_cpu      2.569233
Loss_cpu              1.092571
Val_loss_cpu          1.152247
Accuracy_gpu          3.166968
Val_accuracy_gpu      4.518619
Loss_gpu              1.022652
Val_loss_gpu          1.374272
Accuracy_mgpu         4.068957
Val_accuracy_mgpu     0.582719
Loss_mgpu             1.433650
Val_loss_mgpu         0.177257
dtype: float64
```

```
Accuracy_Horovod             -0.663786
Val_accuracy_Horovod         -0.965336
Loss_Horovod                 -1.423018
Val_loss_Horovod             -1.328064
Accuracy_Sonnet(Repl)         1.912559
Val_accuracy_Sonnet(Repl)     2.770628
Loss_Sonnet(Repl)             0.159264
Val_loss_Sonnet(Repl)         1.406755
Accuracy_Spark                4.797223
Val_accuracy_Spark            1.147671
Loss_Spark                    1.288171
Val_loss_Spark                0.855739
dtype: float64
```

Kehyksien Training- ja Validation Accuracys & Loss -plotteja





Opinnäytetyössä käytettyjä koodeja

Gitlab-linkki: https://gitlab.dclabra.fi/hekto/framework_forpdist_ml

Neuroverkon koulutuskoodin -template

```
#Select dataset directory
ds_pet_dir = "../../Cat_Dog/PetImages/"
#Generating a dataset
ds_pet = tf.keras.preprocessing.image_dataset_from_directory(ds_pet_dir)
#Defining parameters for the loader:
batch_size = 32
img_height = 180
img_width = 180
#Data augmentation
data_augmentation = keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1)])
#Checking data format
from keras import backend as K
if K.image_data_format() == "channels_first":
    input_shape = (3, img_height, img_width)
else:
    input_shape = (img_height, img_width, 3)
# build model
def build_and_compile_model():
    model = Sequential([
        data_augmentation,
        layers.experimental.preprocessing.Rescaling(1./255, input_shape=(input_shape)),
        layers.Conv2D(16, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Dropout(0.5),
        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dense(1, activation="sigmoid")
    ])
    model.compile(optimizer="Adam", loss="binary_crossentropy", metrics=["accuracy"])
    return model
# set strategy
strategy = tf.distribute.MirroredStrategy()
# strategy = tf.distribute.MirroredStrategy(devices=["/gpu:0", "/gpu:1"])
# set G_BATCH_SIZE
G_BATCH_SIZE = batch_size * strategy.num_replicas_in_sync
ds_pet_train = tf.keras.preprocessing.image_dataset_from_directory(
    ds_pet_dir,
    validation_split=0.2,
    subset="training",
    seed=1337,
    image_size=(img_height, img_width),
    batch_size=G_BATCH_SIZE)
ds_pet_test = tf.keras.preprocessing.image_dataset_from_directory(
    ds_pet_dir,
    validation_split=0.2,
    subset="validation",
    seed=1337,
    image_size=(img_height, img_width),
    batch_size=G_BATCH_SIZE)
with strategy.scope():
    model = build_and_compile_model()
history = model.fit(ds_pet_train, validation_data=ds_pet_test, epochs=20)
```

Random-kuva ja todennäköisyyskoodi

```
#Using a random cat picture found in the web
picture_url = "https://placekitten.com/g/200/300"
picture_path = tf.keras.utils.get_file("300", origin=picture_url)

img = keras.preprocessing.image.load_img(picture_path, target_size=(img_height, img_width))
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.sigmoid(predictions[0])

print("This image most likely belongs to {} with a {:.2f} percent confidence.".format(ds_pet.class_names[np.argmax(score)], 100 * np.max(score)))
```

Clean Data -koodi

```
#Filtering out corrupted images
import os
num_skipped = 0
for folder_name in ("Cat", "Dog"):
    folder_path = os.path.join(ds_pet_dir, folder_name)
    for fname in os.listdir(folder_path):
        fpath = os.path.join(folder_path, fname)
        try:
            fobj = open(fpath, "rb")
            is_jfif = tf.compat.as_bytes("JFIF") in fobj.peek(10)
        finally:
            fobj.close()
        if not is_jfif:
            num_skipped += 1
            # Delete corrupted image
            os.remove(fpath)
print("Deleted %d images" % num_skipped)
```