TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Liis Karbus 206017IACB

Tähtede ja numbrite tuvastamine konvolutsiooniliste närvivõrkude abil Pythonis ja MatLabis

Bakalaureusetöö

Juhendaja: Kristina Vassiljeva
PhD

Tallinn 2024

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Liis Karbus

02.01.2024

Annotatsioon

Bakalaureuse lõputöö raames uuriti, mis on süvaõpe ning selle erinevaid algoritme. Püstitati ülesanne, kus klassifitseeritakse käsitsi kirjutatud tähti ja numbreid, mille jaoks kasutati sügavat konvolutsioonilist närvivõrku. Seda ülesannet lahendati kahes programmeerimiskeeles: Python ja MatLab. Mõlema lahenduse tulemusi analüüsiti ja koostati täpsem ülevaade töökäigust dotsent Eduard Petlenkovi aine laborite jaoks, et tudengid saaksid iseseisvalt läbi teha ülesannet.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 6 peatükki, 13 joonist, 2 tabelit.

Abstract

Numbers and letters recognition using convolutional neural networks in Python and MatLab

The bachelor's thesis explored what deep learning is and its various algorithms. A task was set up to classify handwritten letters and numbers for which a deep convolutional neural network was used. It was solved in two languages: Python and MatLab. The results of both solutions were analysed, and a more detailed overview of the workflow was prepared for the laboratories of Professor Eduard Petlenkov, so that the students could also complete the task themselves.

Thesis is divided into five main chapters (seven total) that gives the reader better understanding of deep learning and convolutional neural networks. First, an overview of picture classification is given: what it is, how it has developed and the state-of-the-art, which is currently convolutional neural network. Secondly, deep learning is explained, its ties to machine learning and artificial intelligence, how a simple deep learning process would work, and then an overview of few algorithms like autoencoders, recurrent neural networks and convolutional neural networks. Convolutional neural networks were discussed in more detail, as this kind of algorithm is chosen for the task. In the next chapter an outline of different Python's deep learning open-sourced libraries is given and Tensorflow with Keras is chosen for the task. Then the finished code is explained in detail and how to set it up. In the next chapter a brief rundown of MatLab is given and then MatLab's task solution is explained in depth. In the last chapter an analysis of Python's and MatLab's results is given.

The thesis is in Estonian and contains 26 pages of text, 6 chapters, 13 figures, 2 tables.

Lühendite ja mõistete sõnastik

AI	Tehisintellekt
DCNN	Sügav konvolutsiooniline närvivõrk
CNN	Konvolutsiooniline närvivõrk
GPU	*Graphics processing unit*, graafikaprotsessor
FC	*Fully connected*, täielikult ühendatud
RNN	*Recurrent neural network*, rekurrentne närvivõrk
ReLU	*Rectified linear unit*, mittenegatiivne lineaarfunktsioon
NAS	Närvivõrgu arhitektuuriline otsing

Sisukord

1 Sissejuhatus 9
2 Pildi klassifikatsioon 10
3 Süvaõpe 12
3.1 Süvaõppe protsess 12
3.2 Süvaõppe algoritmid 13
3.2.1 Konvolutsiooniline sügav närvivõrk 14
4 Python 16
4.1 Lahendus 16
4.1.1 Kood 17
4.1.2 Algne ülesehitus 23
4.1.3 Tulemused 24
4.1.4 Probleemid 25
5 MatLab 27
5.1 Kood 27
5.2 Tulemused 33
6 Analüüs 35
7 Kokkuvõte 37
Kasutatud kirjandus 38
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 40

Jooniste loetelu

Joonis 1. Mudeli visualisatsioon 23
Joonis 2. Treenimise ja valideerimise täpsus ning kadu 24
Joonis 3. Treenimise ja valideerimise kadu 24
Joonis 4. Testimise ennustused 25
Joonis 5. Treenimise ja valideerimise täpsus üle sobitamisel 26
Joonis 6. Treenimise ja valideerimise kadu üle sobitamisel 26
Joonis 7. Üle treenimise täpsus 26
Joonis 8. Üle treenimise kadud 26
Joonis 9. Närvivõrgu mudeli visualisatsioon MatLabis 32
Joonis 10. Käskluste akna vaade 33
Joonis 11. Närvivõrgu täpsus 33
Joonis 12. Närvivõrgu kadud 33
Joonis 13. Ennustamise tulemused 34

Tabelite loetelu

Tabel 1. Meetrikad..... 36
Tabel 2. Pythoni ja Matlabi erinevuste ülevaade..... 36

1 Sissejuhatus

Käesoleva bakalaureusetöö eesmärk on uurida, mis on süvaõppe, selle erinevad algoritmid ning süvaõppe algoritmide seast valida parim numbrite ja inglise tähestiku käsitsi kirjutatud tähtede tuvastuseks. Algoritmide seast valiti sügav konvolutsiooniline närvivõrk mitmel põhjusel, millest kirjutatakse täpsemalt ülejärgmises peatükis, ning ülesannet lahendatakse MatLabi ja Pythoni programmeerimiskeelega. Mõlemal programmikoodil kasutatakse sama andmekogumit ja algoritmi, et oleks parem võrrelda nende tulemusi, et näha kumb lahendus oleks kiirem, täpsem ja mugavam. Seejärel tehakse tehtud tööst laborijuhend õppejõu Eduard Petlenkovi aine tudengitele, et nad saaksid iseseisvalt läbi teha ning võrrelda tulemusi õppejõu enda edasisuunalise närvivõrgu laboriga.

2 Pildi klassifikatsioon

Interneti Keskuse (IDC) andmetel oli kogu ülemaailmne andmemaht 2020. aastal jõudnud kuni 42ZB-ni [1]. Tohutu hulk sellest teabest edastatakse fotode või videotena. Nõudlus pilditöötlusele on aastatega kasvanud mitmetes rakendusvaldkondades näiteks meditsiinis, et tuvastada pildilt vähirakke, või sõjas, et tuvastada vaenlase sõidukeid. Inimesel on võime omandada ja tõlgendada seda rikkalikku teavet meie ümber, kuid see on tõsine väljakutse arvutile, sest masin näeb ainult bittide jadasid. Sellepärast on oluline mõista, kuidas pilditöötlus käib, et oleks võimalik ka arvutil pildist õigesti aru saada. Pildid koosnevad tulpadest ja ridadest täis piksleid, mis moodustavad mitmedimensioonilise maatriksi. Pildi klassifitseerimiseks on masinal vaja tuvastada mustreid. Igal objektil on oma muster ja tunnused, seega need, millel on sarnased karakteristikud, kuuluvad ühte mustriklassi. Näiteks iga täht ingliskeelses tähestikus koosneb tunnuste komplektist, kuhu kuuluvad horisontaalsed, vertikaalsed, kaldus sirged jooned ja ka kõverjoonelised joonelõigud. Kui tähte „A“ kirjeldab kaks kaldjoont ja üks horisontaaljoon, siis tähel „B“ on vertikaalne joon kahe kõverjoonelise segmendiga [2].

Traditsiooniliselt kasutati klassifikatsiooni probleemi lahendamiseks kaheetapilist lähenemist. Esmalt eraldati omadused pildist käsitsi, kasutades omaduse kirjeldusi ning need toimisid kui sisendina treenimise jaoks. Selle lähenemisviisi suureks probleemiks oli see, et klassifitseerimise ülesande täpsus sõltus liiga palju selle pealt, kui hästi oli omaduse eraldamise disain tehtud. Süvaõppe mudelid, mis kasutavad mitu mittelineaarset kihti andmetöötluseks ja omaduste eraldamiseks, on selle väljakutsega paremini toime tulnud. Nende hulgast on konvolutsiooniline närvivõrk saanud juhtivaks arhitektuuriks enamiku pildi tuvastamise ja klassifitseerimise ülesannete jaoks. Sügavad CNN-id (konvolutsiooniline närvivõrk) said populaarsemaks kui tulid GPU-d (*graphics processing unit*), suuremad andmestikud ja paremad algoritmid [3].

Kõige esimene CNN tutvustati aastal 1998 LeCun poolt, kus klassifitseeriti käsitsi kirjutatud tähti. CNN mudelil nimega LeNet on seitse kaaluga kihti, millest kolm on konvolutsiooniline, kaks on keskmise koonduvusega kihti, üks täielikult ühendatud kiht ja üks väljundkiht. LeNet saavutas testimise vea 0.95% MNIST andmestikuga.

Kõige rohkem tähelepanu sai DCNN (sügav konvolutsiooniline närvivõrk) aastal 2012, kui toimus ImageNet Large Scale Visual Recognition Challenge, kui võistluse võitja Krizhesky disainis suure DCNN, mille nimeks pandi AlexNet, et klassifitseerida umbes 1.2 miljon pilti 1000 klassist rekordiliste tulemustega. AlexNet oli arhitektuurilt sama, mis LeNet, aga tunduvalt suurem. Sellel on 8 treenimiskihti, sellest viis on konvolutsiooni kihid ja kolm on FC (täielikult ühendatud) kihid. Treenimise kiirendamiseks kasutatakse ReLU (mittenegatiivne lineaarfunktsioon) pärast igat treenimiskihti ja lisaks on kasutusel ka maksimaalse-koonduvuse kihid [4].

3 Süvaõpe

Süvaõpe on masinõppe alamvaldkond ning masinõpe on omakorda tehisintellekti alamvaldkond. AI (tehisintellekti) all mõeldakse masinat, mis imiteerib inimese intellekti ning kognitiivseid funktsioone nagu probleemide lahendamine ja õppimine. AI kasutab ennustamist ja automatiseerimist, et lahendada arvuti jaoks keerukaid probleeme, mida inimesed peavad igapäevaselt tegema, näiteks näotuvastus, otsuste tegemine ja tõlkimine.

Masinõpe, nagu mainitud, on tehisintellekti alamkategooria, mis õpib oma varasemast kogemusest, leides sisestatud andmetes mustreid. Masinõpe kasutab statistilisi algoritme, mis paraneb automaatselt ilma inimese abita. Enamik masinõppe meetoditel on ainult üks või kaks mittelineaarset kihti.

Süvaõppe puhul on samuti tegemist iseõppiva süsteemiga, aga andmestik on palju suurem ja kasutatakse rohkem kui kolme närvivõrgu kihti, mis on inspireeritud inimese ajust, mis saavad samuti õppida struktureerimata andmestikust. Lähenemine on samuti teistsugune kui masinõppel: selle asemel, et saada tunnused kätte käsitsi määratledes kindlad reeglid ja algoritmid, õpib süsteem ise automaatselt eraldama tunnuseid, nagu näiteks kuju, värv või tekstuur, õppimise protsessis. See tähendab, et süvaõpe on väga efektiivne keeruliste ülesannete lahendamisel, kus informatsiooni on tohutult ning andmed ei pruugi olla ära sildistatud, näiteks pildi- ja kõnetuvastuses [5], [6].

3.1 Süvaõppe protsess

Kasside ja koerte identifitseerimine on inimeste jaoks väga lihtne ülesanne – juba ühest pilgust piisab, et aru saada kumb loom on kass ning kumb on koer. Masinõppe puhul on vaja kõik reeglistikud täpselt kirja panna, et masin oskaks teha nende järgi otsuseid, kuid see osutub üpris keeruliseks, näiteks on hulk tõuge, mis võivad teineteisest erineda, näiteks chihuahua või labrador, kuid mõlemad on siiski koerad. Lisaks tekib probleem, kui on pildi perspektiiv halb, näiteks on pilt tehtud ülevalt vaatest, külje profiil vms, või loom on poolenisti kaetud/peidus. Kõige selle defineerimine on keeruline ning

sellepärast on süvaõpe väga kasulik - keerulisemad definitsioonid koosnevad lihtsamatest ja kihid toetuvad üksteisele hierarhiliselt. Esimene kiht süvaõppes on lähteandmed, mis on nähtavad meile, näiteks pilt kassist. Seejärel järgnevad peidetud kihid, mis eraldavad järjest abstraktsemaid tunnuseid pildist. Neid kutsutakse peidetuks, sest need väärtused ei ole andmestiku poolt antud, vaid mudeli enda kalkulatsioon, mis on oluline informatsioon pildi tuvastuseks. Esimene peidetud kiht võib näiteks tuvastada jooni pildis võrreldes pikslite heledust naabripikslitega. Järgmine kiht seejärel saab eraldada sisendist nurgad ja kontuurid, mis koosnevad esimese kihi joontest. Kolmas peidetud kiht võtab teise kihi nurgad ja kontuurid ning avastab kindlamad objektide osad, näiteks pea või kõrvad. Kolmanda kihi tulemusena saab süsteem teha pakkumise, et tegemist on näiteks kassiga [7].

3.2 Süvaõppe algoritmid

Üle aastate on tekkinud mitmeid kasulikke süvaõppe algoritme. Tehniliselt võib kõiki algoritme kasutada väga erinevate ülesannete jaoks, kuid mõned algoritmid on paremad teatud ülesannete jaoks kui teised ning laialdasemalt ka kasutusel, mis tähendab, et nende kohta eksisteerib rohkem informatsiooni. Lõputöös kirjutatakse mõningast närvivõrgu algoritmidest, kuid pikemalt jäädakse peatuma konvolutsioonilise närvivõrgule, sest see algoritm valitakse püstitatud ülesande lahendusel.

Autoenkooderid on teatud tüüpi edasisuunalised närvivõrgud, mille väljund on sisendi rekonstruktsioon. Neid kasutatakse peamiselt sisendandmete lihtsustamiseks, juhendamata õppimisel või anomaaliate tuvastamiseks. Närvivõrk koosneb kolmest komponendist: kodeerija, kood ja dekooder. Kodeerija pakib sisendi väiksemaks ja genereerib koodi, seejärel rekonstrueerib dekooder sisendi kasutades antud genereeritud koodi. Sel viisil on võrk sunnitud hoidma ainult olulist teavet ja hüljates müra [8].

RNN-id (rekurrentne närvivõrk) on loodud spetsiaalselt järjestikuste andmestruktuuride jaoks nagu aegridade (*time-series*) andmed, sündmuste jadad ja loomulikud keeled. RNN arhitektuuris liigub informatsioon tsüklis ja otsuste tegemisel võtab RNN arvesse praegust ja ka seda, mida ta on varem saadud sisenditest õppinud. Tänu sisemälule suudetakse saadud sisendi kohta olulisi asju meeles pidada, mis võimaldab närvivõrgul järgmisel ennustamisel olla täpsem. Lihtsal RNN arhitektuuril tekib probleeme kaduvate gradientidega, mille tõttu varasem mälu ei ole piisavalt mõjuv. Keerulisemad

arhitektuurid nagu pika lühimälu närvivõrgud lahendavad selle probleemi. RNN-i kasutatakse sellistes tarkvarades nagu Siri ja Google Tõlge [9], [10].

Süvaõppe närvivõrgud on väga võimsad arhitektuurid, kuid nende arendamine on keeruline ja aeganõudev protsess. Enamus närvivõrke on ehitatud inimeste poolt, mis tähendab, et läheb rohkem ressursse ja vead tekivad tihedamini. Sellepärast on NAS (närvivõrgu arhitektuuriline otsing) hakanud saama rohkem tähelepanu. NAS aitab leida parima närvivõrgu antud probleemi jaoks ja automatiseerib närvivõrgu disaini, tagades suurema jõudluse, kiiruse ja väiksemad kaod kui käsitsi kirjutatud arhitektuuril. NAS-il on kolm peamist komponenti: otsingu ruum, otsingu strateegia ja hindamisstrateegia (toimivuse prognoos). NAS otsingu ruum defineerib missuguseid arhitektuure saab avastada NAS algoritmi abil. See on määratletud operatsioonide kogumiga, mis määrab võrgu üldise struktuuri, kihte määratlevate üksuste või plokkide tüübi. Mida rohkem elemente otsinguruumis on, seda keerukamaks ja mitmekülgsemaks see muutub, kuid tõuseb ka kulukus. Otsingu strateegia paneb paika, kuidas navigeerida otsingu ruumis, et leida võimalikult kiiresti arhitektuuri, kuid samal ajal jälgida ka, et see arhitektuur oleks nende seast parim. Hindamisstrateegia komponendi eesmärk on leida kõrge sooritusega arhitektuure. Kõige parim moodus selleks on teha lihtne arhitektuuri treenimine ja valideerimine andmetega, kuid see läheb kalliks ning aega nõudvaks, kui seda peab kõikidele arhitektuurile tegema ükshaaval, sellepärast on uuemad uuringud keskendunud meetodite arendamiseks, mis vähendaksid selle protsessi aega ja kulukust [11].

3.2.1 Konvolutsiooniline sügav närvivõrk

CNNi kasutatakse peamiselt pildi- ja kõne tuvastuse ülesannete jaoks. Tüüpiline CNN koosneb kas ühest või mitmest konvolutsiooni ja diskreetimise kihtide plokist, pärast mida tuleb täielikult ühendatud kiht ja väljundkiht [4].

Konvolutsioon on matemaatiline operatsioon kahe funktsiooni vahel, mille tulemusel tekib kolmas funktsioon [12]. Närvivõrgu korral on konvolutsioon täpsemalt kahe maatriksi elementide korrutiste summa. Valemis on konvolutsioon tähistatud * operaatoriga, I on kahedimensiooniline sisendpilt ja K on kahedimensiooniline tuum.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n K(i-m, j-n)\, I(m,n)$$

Enamus masinõppe ja süvaõppe andmeteeke kasutavad tegelikult lihtsustatud ristkorrelatsiooni funktsiooni.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n K(i+m, j+n)\, I(m,n)$$

Konvolutsiooni kihtide eesmärk on eraldada tunnuseid pildist. Konvolutsiooni kihtides olevad neuronid esitatakse tunnuskaardina, kus igal tunnuskaardi neuronil on seos naaberneuronitega eelmises kihis kasutades treenimiskaale, kutsutud ka filtrina. Iga filter on kaalu maatriks, mis võib kahedimensioonilise pildi puhul olla näiteks suurused 3 x 3 või 5 x 5. Sisendid konvolutsioonitakse õpitud kaaludega, et arvutada uus tunnuskaart ja tulemused saadetakse ReLU abil edasi. Igal neuronil on kaal, mis on sunnitud olema võrdne, aga erinevatel tunnuskaarditel samas konvolutsiooni kihis on erinevad kaalud, et tunnuseid saaks eraldada igast kohast. Selle jaoks, et väljundpilt oleks samade dimensioonidega nagu sisendpilt, kasutatakse polsterdamist ehk täiendamist. Olemas on mitu erinevat polsterdamise meetodit nagu näiteks null polsterdus, mille puhul lisatakse äärtesse null bitte, või ümber mässitud polsterdus, mille puhul äärmised pikslid pannakse pildi teise ääre pikslite järgi [3], [5], [7].

Koondamine tähendab pildi diskreetimist. See võtab väiksed alad konvolutsiooni väljundist sisendiks ja diskreetib seda, et saada üksik väljund. Olemas on mitmeid erinevaid diskreetimise tehnikaid nagu maksimaalne diskreetimine, keskmine diskreetimine jne. Koondamise kiht lisatakse perioodiliselt konvolutsiooni kihtide vahele vähendamaks parameetrite arvu närvivõrgus ja töötlemise ressursse [3], [13].

Mitu konvolutsiooni ja koonduvat kihti on tavaliselt üksteise otsa paigaldatud, et eraldada järjest abstraktsemaid tunnuskaarte närvivõrgus edasi liikudes. Täielikult ühendatud kihid, mis järgnevad nendele kihtidele, tõlgendavad neid tunnuskaarte ja sooritavad kõrgetasemelise arutlemise [3].

4 Python

Pythoni programmeerimiskeeles käsitsi kirjutatud inglise tähestiku tähtede ja numbrite tuvastuse ülesande lahendamiseks on suureks abiks Pythoni süvaõppe raamistikud, mis aitavad programmeerijatel hoida aega kokku kodeerimisel, et iga kord ei peaks uuesti kirjutama suuri koguseid koode. TensorFlow, Keras ja PyTorch on kolm kõige populaarsemat süvaõppe raamistikku.

TensorFlow arendati välja Google Brain tiimi poolt ja tehti avalikkusele kättesaadavaks aastal 2015 Apache litsentsi all ehk seda saab kasutada tasuta. Üheks suureks plussiks TensorFlow juures on see, et sellel on hea kokkusobivus Kerase raamistikuga, mis tähendab, et kasutajad saavad lisada kõrge tasemelist funktsionaalsust oma koodile säilitades Kerase lihtsust. Peale selle saab TensorFlow-d kasutada ka teiste programmeerimiskeeltega peale Pythoni, näiteks Javascript ja C++. Kahjuks on TensorFlow-l ka mitu miinust näiteks ebajärjekindlad homonüümid ja pikk treenimisaeg [14].

Keras on samuti avalikkusele tasuta kättesaadav raamistik, mis loodi aastal 2015 François Chollet poolt. Keras on väga lihtne ja kasutaja sõbralik, mis aitab kaasa süvaõppe õppimisele ning sellel on suur kommuun täis programmeerijaid ja teadureid. Kerase miinusteks on näiteks ebakasulikud vigade sõnumid, mis ei aita kaasa vea peamise põhjuse leidmiseks, ning lisaks on Keras kõrge-tasemeline ehk kasutajal on vähem võimalust funktsioonide teisiti kasutamiseks [14].

PyTorch on kõige uudsem süvaõppe raamistik, mis on loodud Meta (Facebook) AI uuringu grupi poolt aastal 2016, mille eesmärgiks on olla kiirem alternatiiv NumPy-le. Läbi aastate on PyTorch kogunud omale mainet oma lihtsuse, paindlikkuse ja efektiivsuse tõttu. Samuti on PyTorchis juba väga tugev kommuun, kes näevad palju vaeva, et nende töö oleks hästi dokumenteeritud teiste programmeerijate jaoks. Üheks miinuseks PyTorchi juures on see, et visualiseerimiseks on vaja eraldi tööriista [15].

4.1 Lahendus

Treenimiseks ja testimiseks on vaja suurt hulka andmeid, mille kogumine võtab kaua aega, seega on hea kasutada juba ettevalmistatud andmekogumeid. Üks hea veebileht vajalike andmekogumite leidmiseks on Kaggle. Bakalaureusetöös kasutatakse

andmekogumit, kus on 3140 pilti inglise keele käsitsi kirjutatud tähtedest ja numbritest. Andmekogum sisaldab piltide kausta ja csv faili, kus on kaks veergu: ühes veerus piltide asukoht ja faili nimetus ning teises veerus iga pildi klassisilt. Kokku on 62 klassi, igal klassil 55 pilti, ja nendest on 10 klassi numbrid 0-9, 26 tähed A-Z trükitähtedena ja 26 klassi tähed a-z kirjatähtedena [16].

Raamistike kasutamiseks on vaja need oma arvutisse alla tõmmata, mille jaoks on paar võimalust: manuaalselt allalaadimine *pip* käsuga või kasutades rakendust Anaconda. Anacondaga on hea võimalus teha erinevaid keskkondi, kuhu saab tõmmata erinevaid pakettide versioone. See on oluline selleks, sest tihtipeale võib pakettide uuendamisel vana kood mitte töötada või uus pakett ei tööta teiste vanemate pakettidega. Sellise keskkonnaga on hea võimalus proovida või arendada erinevaid koode, mis vajavadki erinevaid paketiversioone.

4.1.1 Kood

Koodi alguses lisatakse vajaminevad välised teegid ja moodulid, mille järel saab nende funktsioone kasutada järgnevas koodis. *Os* mooduli importimine võimaldab meil suhelda operatsioonisüsteemiga, näiteks lugeda või kirjutada faile. *Pandas* teeki kasutatakse enamasti andmete lugemiseks ja analüüsimiseks. *Random* moodul on funktsioon, mis genereerib suvalisi numbreid. *Cv2* ehk *OpenCV* teeki kasutatakse piltide töötlemiseks. Lõputöös kasutatakse Tensorflow ja Kerase raamistikku ning nende mitmeid mooduleid. *ImageDataGenerator* moodulit kasutatakse andmete rikastamiseks. *Plot_model* moodulit kasutatakse närvivõrkude visualiseerimiseks ning selle mooduli kasutamiseks on vaja eraldi tõmmata alla veel *pydot*, *pydotplus* ja *graphviz* paketid. *Matplotlib.pyplot* moodulit kasutatakse treenimise tulemuste visualiseerimiseks Pythonis. *Sklearn.metrics* moodulit kasutatakse, et arvutada erinevaid meetrikaid nagu täpsus, tagastus ja F1.

```
import os
import pandas as pd
import random
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import plot_model
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix
import matplotlib.pyplot as plt
```

Järgmisena on kaks rida koodi, mis on olulised reprodutseeritavuse tagamiseks või kui ei soovi juhuslikkust arendamise või koodi silumise ajal. Valitud number ise ei ole oluline, ainult, et see oleks kõikidel kordadel sama.

```
random.seed(24)
tf.random.set_seed(24)
```

Esimene koodi funktsioon loeb funktsiooni sisestatud faili asukohas oleva csv formaadis faili ja tagastab selle. Lõputöös kasutatava andmestiku csv fail koosneb ainult kahest veerust: esimeses veerus on iga märgi pildi asukoht ja failinimi ning teises veerus on klass, kuhu see märk kuulub.

```
def read_csv(file_path):
    return pd.read_csv(file_path)
```

Teise funktsiooni eesmärk on andmestiku jagamine kolmeks osaks: treenimine, valideerimine ja testimine. Põhjus valideerimise ja testimise mõlema olemasolu jaoks on hinnata mudeli toimivust andmetel, mida mudel pole treenimisel juba näinud. *Rand_val* võtab 500 suvalist elementi sisse loetud andmestikust. *Validation_set* loob valideerimise kogumiku eraldades ridu suvaliselt valitud elementidest ja seejärel eraldatakse need read originaalsest andmestikust, et neid ei kasutataks treenimise jaoks. Testimise jaoks valitakse juhuslikult 5 elementi, nendest eraldatakse read test kogumiku jaoks ja valideerimise kogumikust eraldatakse need read. Seejärel funktsioon tagastab valideerimise ja treenimise kogumikud.

```
def create_validation_test_sets(df):
    rand_val = random.sample(range(len(df)), 500)
    validation_set = df.iloc[rand_val, :].reset_index(drop=True)
    df.drop(rand_val, inplace=True)

    rand_test = random.sample(range(len(validation_set)), 5)
    test_set = validation_set.iloc[rand_test,
:].reset_index(drop=True)
    validation_set.drop(rand_test, inplace=True)

    return validation_set, test_set
```

Kolmandaks funktsiooniks on andmete genereerimine, kus kasutatakse Kerase moodulit *ImageDataGenerator*. See Kerase funktsiooni tüüp täiendab n-ö käigupealt andmeid. Sellest on vale arusaam, et andmete rikastamine tähendab andmete lisamist, et kui

alguses on üks pilt, siis pärast funktsiooni kasutamist on näiteks 4 pilti. See ei ole Kerase funktsiooni puhul õige. See hoopis võtab sisestatud andmed ning muudab neid veidi, näiteks pöörab pilti horisontaalselt vms. See on oluline selle jaoks, et iga treenimise epohh ajal näeb mudel „uusi“ pilte ja sellega saavutatakse parem generaliseerimine. *Train_data_generatori* puhul kasutatakse erinevaid rikastamise tehnikaid nagu ümber skaleerimine, nihutamine ja sisse suumimine. *Data_generator* on lihtsam generaator, mis kasutab ainult ümber skaleerimist ja kasutatakse valideerimise ja test andmestiku jaoks. *Flow_from_dataframe* kasutatakse, et genereerida rikastatud andmete plokke. Seejärel rikastatud andmestikud tagastatakse ning kasutatakse treenimisel.

```
def create_data_generators(datadir, train_df, validation_df, test_df):
    train_data_generator = ImageDataGenerator(rescale=1/255,
shear_range=0.2, zoom_range=0.2)
    data_generator = ImageDataGenerator(rescale=1/255)

    training_data_frame =
train_data_generator.flow_from_dataframe(dataframe=train_df,
directory=datadir, x_col='image', y_col='label',

target_size=(64, 64), class_mode='categorical',
validate_filenames=False)
    validation_data_frame =
data_generator.flow_from_dataframe(dataframe=validation_df,
directory=datadir, x_col='image', y_col='label',

target_size=(64, 64), class_mode='categorical',
validate_filenames=False)
    test_data_frame =
data_generator.flow_from_dataframe(dataframe=test_df,
directory=datadir, x_col='image', y_col='label',

target_size=(64, 64), class_mode='categorical', shuffle=False,
validate_filenames=False)

    return training_data_frame, validation_data_frame, test_data_frame
```

Järgmine funktsioon on CNN mudeli loomine. Lõputöös kasutatakse arhitektuuri, mis on disainitud just pildi klassifitseerimiseks 62 klassi sisendiga. Esimene on sisendkiht – conv2D kiht koos 32 filtriga ja kerneli ehk konvolutsioonimaatriksi suurusega 3 x 3. Polstriks on pandud sama, mis tähendab seda, et sisendandmete servadele lisatakse täiendavaid pikslite ridu ja veerge, et tunnuskaartide suurus oleks sama, mis sisendandmetel. Lisaks on lisatud sisendpildi suurus ja ReLU aktiveerimine. Seejärel tuleb esimene konvolutsiooni ja koonduvuse kihtide plokk. Koonduvuse kihis kasutatakse maksimaalset koonduvust suurusega 2 x 2 ja väljalangemiskihis on

väljalangemise määraks pandud 0.25. Teises konvolutsiooni ja koonduvuse kihtide plokis on kaks konvolutsiooni kihti juba 64 filtriga, maksimaalse koonduvuse kiht ja väljalangemise kiht. Kolmas plokk on sama, mis teine plokk. Seejärel tuleb lamendamise kiht ning pärast seda tuleb täielikult ühendatud kiht 512 neuroni ja ReLU aktiveerimisega, mille järel tuleb väljalangemiskiht ja üks väljundkiht 62 neuroniga. Treenimise valikute all spetsifitseeritakse õppimise kiirust (*learning rate*), mis on suurusega 0.0001 ja ristentroopia kadu (*cross-entropy loss*).

```
def build_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), padding='same',
input_shape=(64, 64, 3), activation='relu'),
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.25),

        tf.keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.25),

        tf.keras.layers.Conv2D(64, (3, 3), padding='same',
activation='relu'),
        tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
        tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        tf.keras.layers.Dropout(0.25),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(62, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.RMSprop(lr=0.0001),
loss="categorical_crossentropy", metrics=["accuracy"])
    return model
```

Järgmisena tuleb treenimise funktsioon. Funktsiooni sisestatakse mudel ning treenimise ja valideerimise andmestik. Lisaks on pandud paika epohhide arv, milleks on 60. Treenimiseks kasutatakse Kerase funktsiooni *fit*(), kuhu saab sisestada ka valideerimise andmestiku. *Fit*() värskendab mudeli kaale pärast igat plokki. Funktsioon tagastab ajaloo (*history*) objekti, kus on olemas vajalik informatsioon treenimise protsessi koht. Seda on hea kasutada tulemuste visualiseerimiseks ja analüüsimiseks.

```
def train_model(model, train_data, validation_data, epochs=60):
    history = model.fit(train_data, validation_data=validation_data,
epochs=epochs)
    return history
```

Järgmine funktsioon on loodud tulemuste visualiseerimiseks. Selle jaoks kasutatakse *Matplotlib* moodulit luues kaks graafikut, kus ühes on treenimise ja valideerimise täpsus ning teises on treenimise ja valideerimise kadu.

```
def plot_training_history(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epochs_range = range(40)
    plt.figure(figsize=(15, 15))
    plt.subplot(2, 2, 1)
    plt.plot(epochs_range, acc, label='Training Accuracy')
    plt.plot(epochs_range, val_acc, label='Validation Accuracy')
    plt.legend(loc='lower right')

    plt.subplot(2, 2, 2)
    plt.plot(epochs_range, loss, label='Training Loss')
    plt.plot(epochs_range, val_loss, label='Validation Loss')
    plt.legend(loc='upper right')
    plt.show()
```

Viimane funktsioon teeb pakkumisi test andmestikul kasutades treenitud mudelit ja kuvab tulemused. Lisaks arvutatakse siin funktsioonis erinevad meetrikaid, nagu täpsus, tagastus ja F1, mis arvutatakse ennustuste täpsust siltide suhtes.

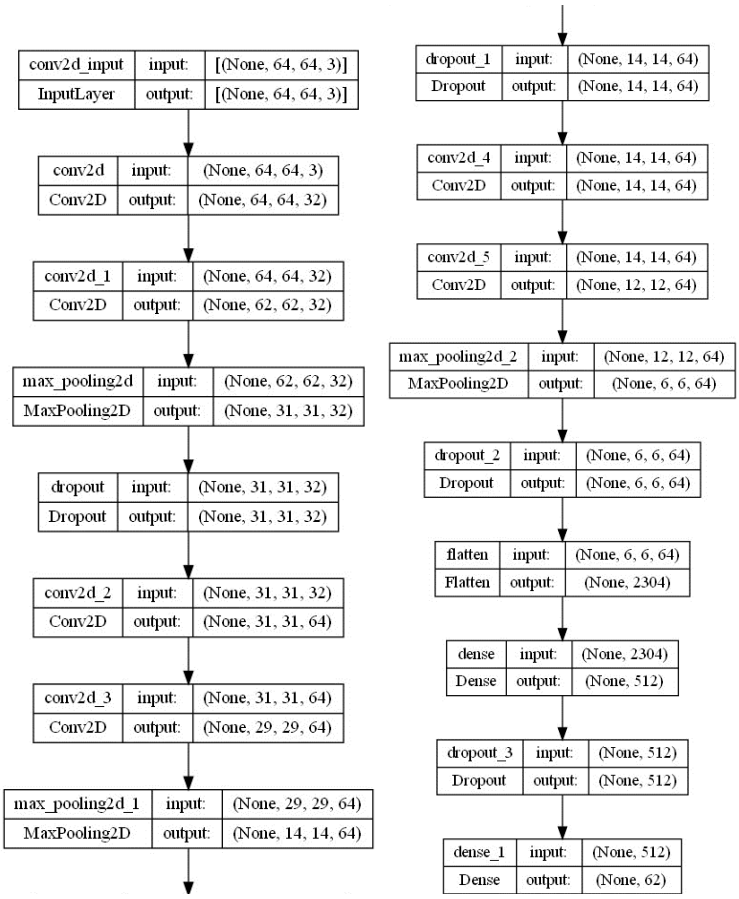
```
def predict_and_display_results(model, test_data_frame, test_set,
datadir, class_dict):
    pred = model.predict(test_data_frame)
    pred_classes = pred.argmax(axis=1)
    reversed_class_dict = {str(v): k for k, v in class_dict.items()}
    true_class_names = test_set['label'].values
    true_labels = [reversed_class_dict[label] for label in
true_class_names]
    accuracy = accuracy_score(true_labels, pred_classes)
    print('Accuracy: %f' % accuracy)
    precision = precision_score(true_labels, pred_classes,
average='micro')
    print('Precision: %f' % precision)
    recall = recall_score(true_labels, pred_classes, average='micro')
    print('Recall: %f' % recall)
    f1 = f1_score(true_labels, pred_classes, average='micro')
    print('F1 score: %f' % f1)

    output_df = pd.DataFrame(pred)
    max_index = list(output_df.idxmax(axis=1))

    for i in range(len(test_set)):
        image = cv2.imread(os.path.join(datadir, test_set.at[i,
'image']))
        plt.title(class_dict.get(max_index[i], "error"))
```

```
        plt.imshow(image)
        plt.show()
```

Kõige viimasena on jäänud veel peamine *main*() funktsioon, kus kutsutakse kõik funktsioonid välja. Alguses pannakse paika vajamineva faili nimi ja selle faili asukoht arvutis. Seejärel loetakse csv fail ning luuakse treenimise, valideerimise ja treenimise andmestikud. Kutsutakse välja andmerikastamise funktsioon, CNN mudeli ehitamise ja treenimise funktsioon. Seejärel visualiseeritakse treenimise tulemused ning samuti mudel ise visualiseeritakse ja salvestatakse arvutisse PNG failina. Viimasena on pakkumise funktsioon test andmestikul ja selle visualiseerimine. Paremaks loetavuseks lühendatakse *class_dict* väärtusi.

```
def main():
    script_dir = os.getcwd()
    file = 'english.csv'
    file_path = os.path.normcase(os.path.join(script_dir, 'data',
file))
    datadir = os.path.normcase(os.path.join(script_dir, 'data'))

    df = read_csv(file_path)
    validation_set, test_set = create_validation_test_sets(df)

    train_data, validation_data, test_data =
create_data_generators(datadir, df, validation_set, test_set)

    model = build_model()

    plot_model(model, to_file='model_plot2.png', show_shapes=True,
show_layer_names=True)

    history = train_model(model, train_data, validation_data)

    plot_training_history(history)
    model.save('model.h5')

    class_dict = {
        0: '0', …, 9: '9', 10: 'A',… , 35: 'Z', 36: 'a', .., 61: 'z'
    }
    predict_and_display_results(model, test_data, test_set, datadir,
class_dict)

if __name__ == "__main__":
    main()
```

Koodi kirjutamiseks kasutati järgmiseid allikaid: [17]–[24].



Joonis 1. Mudeli visualisatsioon

4.1.2 Algne ülesehitus

Anacondas tuleb alguses teha omale keskkonna nimega *baka* kasutades käsku „*conda create –name baka*“. Kui ei spetsifitseerita Pythoni versiooni, siis Anaconda teeb keskkonna hetkeseisu Pythoni versiooniga, mis on arvutil alla tõmmatud. Versiooni spetsifitseerimiseks on vaja käsu lõppu panna *python=versioon* ehk terve käsk oleks näiteks: „*conda create –name baka python=3.10*“. Lõputöös kasutati Python versiooni 3.10. Seejärel tuleb see keskkond aktiveerida käsuga: „*conda activate baka*“ ja hiljem desaktiveerimiseks lihtsalt: „*conda deactivate baka*“.

Programmeerimise lihtsustamiseks kasutatakse erinevaid raamistikke ja selle jaoks on vaja need arvutisse alla tõmmata. Bakalaureusetöös kasutatakse järgmiseid paketi alla tõmbamise käskluseid:

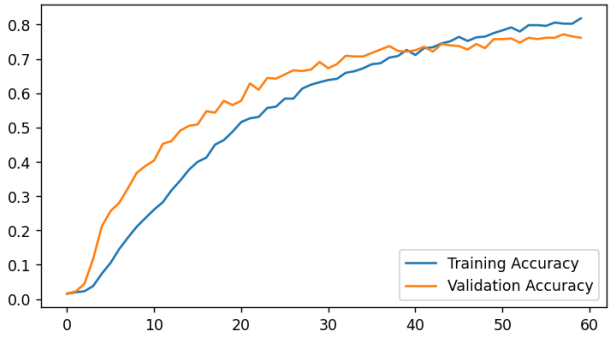
- *conda install tensorflow* (selle käsklusega tõmmatakse alla ka Keras, kuid vajadusel saab Kerase paketti eraldi alla tõmmata käsklusega *conda install keras*)

- *conda install opencv matplotlib panda pydot scikit-learn*

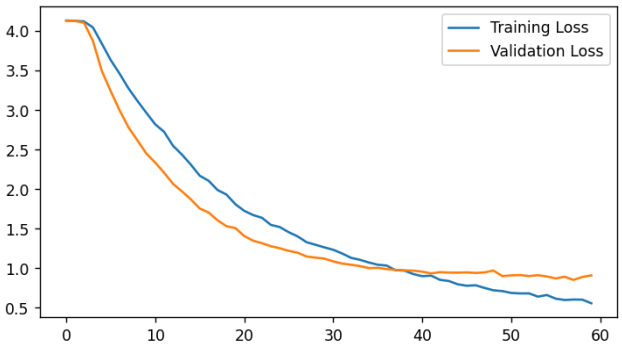
Koodi tööle panemiseks on vaja alguses kasutada käsklust: „*cd <faili asukoht/kaust arvutis>*“, et Python oskaks leida koodi õigest kohast. Seejärel saab koodi kompileerida käsuga: „*python <failinimi>*“, näiteks lõputöös Pythoni koodi pealkiri on TensorKerasCode.py, siis käsklus oleks: „*python TensorKerasCode.py*“. Selle jaoks, et kood leiaks üles Kaggle keskkonnast alla tõmmatud failid, tuleb andmekaust lisada samasse kausta, kus asub tööle pandav kood.

4.1.3 Tulemused

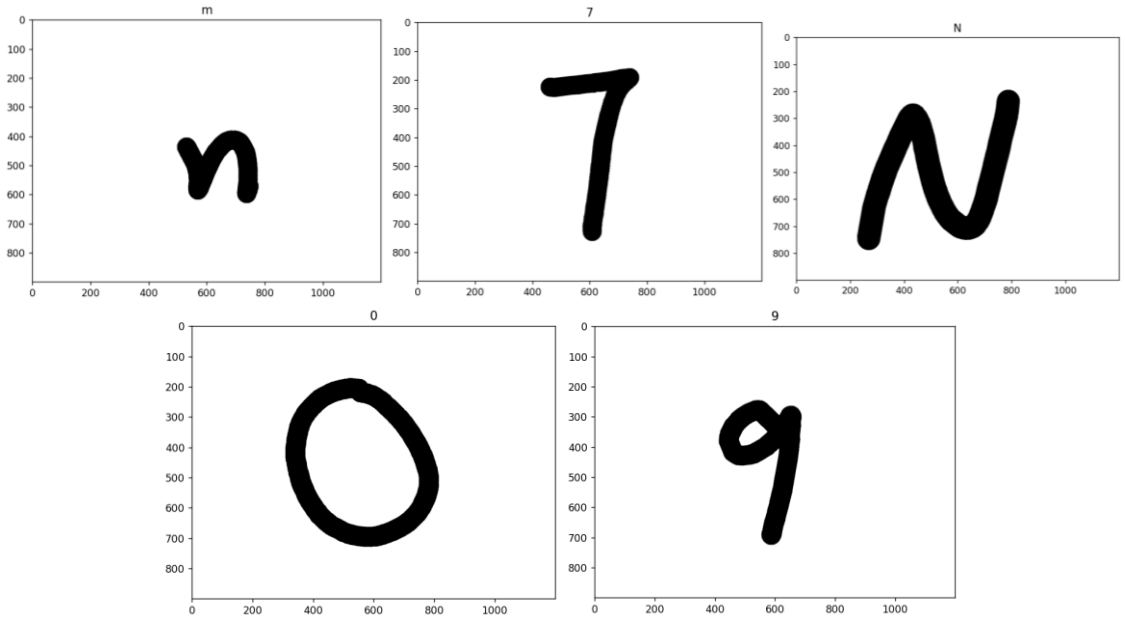
Pythoni konvolutsioonilise närvivõrgu treenimise tulemusena saavutati valideerimise täpsus 76% ja treenimise täpsus 80% ning kadude lõpp tulemused tulid vastavalt 1,1 ja 0,6. Test elementide peal ennustamisel pandi ainult üks pakkumine mööda. Joonis 4 on näha käsitsi kirjutatud tähtede ja numbrite pilte, kus jooniste pealkirjaks on arvuti pakkumine.



Joonis 2. Treenimise ja valideerimise täpsus ning kadu



Joonis 3. Treenimise ja valideerimise kadu



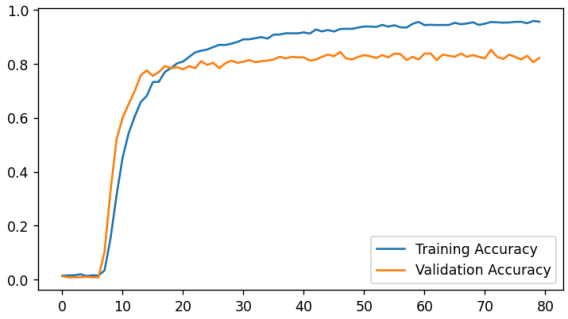
Joonis 4. Testimise ennustused

4.1.4 Probleemid

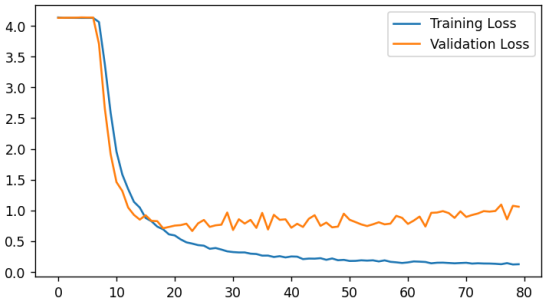
CNN mudeli treenimisel võib tekkida mitmeid probleeme nagu ala sobitamine, üle sobitamine või üle treenimine.

Ala sobitamine juhtub kui mudel ei suuda jõuda piisavalt madalale treenimiskadule ehk mudel ei oska piisavalt õppida mustreid treenimisandmestikus. Ala sobitamisel on lihtne lahendus – tuleb lihtsalt lisada rohkem kihte või neuroneid närvivõrku.

Üle sobitamine juhtub kui närvivõrgu mudel treenib andmestikku „liiga hästi“ ja ei oska üldistada valideerimisandmeid. Sellest saab aru kui vaadata treenimis- ja valideerimiskadude graafikut: nende vahe on liiga suur. Selle probleemi lahendamine on veidi keerulisem, näiteks võib eemaldada kihte või neuroneid närvivõrgust, aga seda soovitatakse ainult väiksema andmestiku puhul, või lisada tugevamad reguleerimise tehnikaid [5].

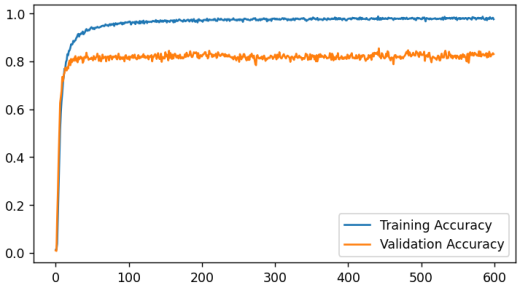


Joonis 5. Treenimise ja valideerimise täpsus üle sobitamisel

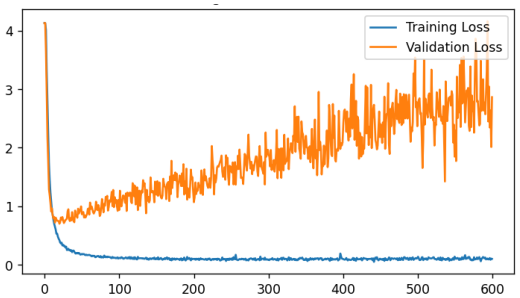


Joonis 6. Treenimise ja valideerimise kadu üle sobitamisel

Üle treenimisel on näha, kuidas treenimise ja valideerimise täpsus stagneerub, kuid valideerimise kadu tõuseb ajaga tohutult ja ei ole üldse stabiilne.



Joonis 7. Üle treenimise täpsus



Joonis 8. Üle treenimise kadud

5 MatLab

MatLab on aastal 1984 firma MathWorks poolt tehtud andmetöötluskeskkond ja kõrgetasemeline programmeerimiskeel. MatLab keskkonnas on olemas erinevad tööriistad, mis on MatLabi poolt professionaalselt välja töötatud, rangelt testitud ja täielikult dokumenteeritud [25]. Bakalaureuse MatLabi lahenduses kasutatakse järgmiseid süvaõppe tööriistu:

- *Deep Learning HDL Toolbox*
- *Deep Learning Toolbox*
- *Image Processing Toolbox*
- *Simulink*
- *Statistics and Machine Learning Toolbox*

Koodi tööle panemiseks tuleb fail avada MatLabi keskkonnas ning seejärel on vaja vajutada ainult käivitamise (*run*) nuppu. Selle jaoks, et kood leiaks üles Kaggle keskkonnast alla tõmmatud failid, tuleb andmekaust lisada samasse kausta, kus asub tööle pandav kood.

5.1 Kood

MatLab kood tehti võimalikult sarnaselt Pythoni koodiga, et saaks kõige paremini võrrelda tulemusi, kuid täiesti üksühele teha oli keerulisem. Probleemseks kohaks tuli närvivõrgu mudeli koostamine. Pythoni närvivõrgu mudeli ülesehitus on veidi teistsugune kui MatLabi oma ning osad kihid on teise nimetusega, mille tõttu oli ekvivalentsete leidmine raskendatud.

MatLabi kood hakkab pihta seemne paika panemisega, kuid kuna Python ja MatLab initsialiseerivad erinevalt ei ole seemne numbrid ühtsed ehk piltide sildid ei ühti mõlemal platvormil. MatLabis ei ole vaja eraldi koodiridu, et importida erinevaid teeke nagu Pythonis, sest tööriistade lisamine MatLabi rakenduses on lihtsustatud.

MatLabi koodi struktuur on veidi erinev Pythonist – *main*() funktsioon on esimesena ja tavalised funktsioonid tulevad pärast seda. *Main*() funktsiooni põhimõte on sama, mis Pythoni koodilgi. Alguses loetakse csv fail sisse, andmestik jaotatakse kolmeks, ehk treenimise, valideerimise ja testimise andmestik, ning seejärel augmenteeritakse ehk rikastatakse treenimise andmestik. Järgmisena ehitatakse närvivõrgu mudel ja hakatakse treenima seda 60 epohhiga. Seejärel visualiseeritakse närvivõrgu mudel ja salvestatakse see ka arvutisse. Viimasena tehakse ennustamist test andmestikul, arvutatakse erinevad meetrikad ja kuvatakse tulemused. Paremaks loetavuseks lühendatakse *key_user* ja *values_user* väärtusi aruandes.

```
rng(24);

main();

function main()
    script_dir = pwd;
    file = 'english.csv';
    file_path = fullfile(script_dir, 'data', file);
    datadir = fullfile(script_dir, 'data');

    df = read_csv(file_path);
    [validation_set, test_set] = create_validation_test_sets(df);

    [training_data_frame, validation_data_frame, test_data_frame] =
create_data_generators(datadir, df, validation_set, test_set);

    model = build_model(training_data_frame, validation_data_frame);

    analyzeNetwork(model)
    save('model.mat', 'model');

    keys_user = [0, …, 61];
    values_user = ["0", …, "9", "A", …, "Z", "a", …, "z"];
    class_dict = dictionary(keys_user,values_user);
    reversed_class_dict = dictionary(values_user, keys_user);
    predict_and_display_results(model, test_data_frame, test_set,
datadir, class_dict, reversed_class_dict);
end
```

Esimene funktsioon pärast peafunktsiooni on csv faili sisse lugemine. Teine funktsioon on esialgse andmestiku jagamine kolmeks sama moodi nagu Pythoni koodis: kogu andmestikust eraldatakse 500 suvaliselt valitud elementi valideerimise jaoks ja 5 testimise jaoks, sellest alles jäänud andmestik läheb treenimisele.

```
function df = read_csv(file_path)
    opts = detectImportOptions(file_path);
    opts.VariableNames;
    opts = setvartype(opts,"label","string");
    df = readtable(file_path, opts);
end
```

```
function [validation_set, test_set] = create_validation_test_sets(df)
    rand_val = randperm(height(df), 500);
    validation_set = df(rand_val, :);
    df(rand_val, :) = [];
    rand_test = randperm(height(validation_set), 5);
    test_set = validation_set(rand_test, :);
    validation_set(rand_test, :) = [];
end
```

Järgmisena on väike lisa funktsioon nimega *cutomReadDatastoreImage*, mida Pythonis ei olnud. Seda funktsiooni kasutatakse sisseehitatud funktsioonis *ImageDatastore*, et eeltöödelda andmeid. *ImageDatastore* funktsiooni kasutatakse andmete konteinerina, et treeningu ajal tõhusamalt hallata ja laadida suuri andmestike komplekte. Treenimise paremaks tulemuseks kasutatakse andmete rikastamist, täpsemalt tehakse piltide nihutamist ja sisse suumimist nagu Pythoni koodis.

```
function data = read_datastore_image (filename)
    onState = warning('off', 'backtrace');
    c = onCleanup(@() warning(onState));
    image_size = [64 64];
    data = imread(filename);
    data = imresize(data, image_size);
end
function [training_data_frame, validation_data_frame, test_data_frame]
= create_data_generators(datadir, train_df, validation_df, test_df)
    shearRange = 0.2;
    zoomRange = [0.8, 1.2];

    train_data_generator = imageDataAugmenter(...
        'RandXShear', [-shearRange, shearRange], ...
        'RandYShear', [-shearRange, shearRange], ...
        'RandScale', [zoomRange(1), zoomRange(2)]);
    image_size = [64 64];

    training_data = imageDatastore(fullfile(datadir, train_df.image),
'Labels', categorical(train_df.label), 'ReadFcn',
@read_datastore_image);
    validation_data_frame = imageDatastore(fullfile(datadir,
validation_df.image), 'Labels', categorical(validation_df.label),
'ReadFcn', @read_datastore_image);
    test_data_frame = imageDatastore(fullfile(datadir, test_df.image),
'Labels', categorical(test_df.label), 'ReadFcn',
@read_datastore_image);
    training_data_frame = augmentedImageDatastore(image_size,
training_data, 'DataAugmentation', train_data_generator);
end
```

Järgmisena tuleb närvivõrgu mudeli loomine. Pythoni närvivõrgul on konvolutsiooni kihil endal samal real näiteks sisend ja ReLU aktiveerimine, kuid MatLabis peavad kõik olema eraldi kihtidena. Samuti ei ole MatLabis *dense* nimetusega kihti, vaid selle ekvivalent on hoopis *FullyConnectedLayer*. Pythonis oli treenimise valikute all

ristentroopia kadu (*cross-entropy loss*), kuid MatLabis oli vaja eraldi närvivõrgu kihina lisada *ClassificationLayer*, et arvutada ristentroopia kadu. MatLabi treenimise valikutel on lisatud paar terminit, mida Pythoni treenimise valikutel ei olnud. Üks nendest on sõnaohtrus (*Verbose)*, mis kuvab treenimise protsessi tekstiliselt, mida on näha Joonis 10. Teine termin on treenimise edenemine (*training-progress*), mis kuvab treenimist graafiliselt, tänu millele ei ole vaja eraldi visualiseerimise funktsiooni nagu Pythonis.

```
function model = build_model(training_data_frame,
validation_data_frame)
    layers = [
        imageInputLayer([64 64 3])
        convolution2dLayer(3, 32, 'Padding', 'same')
        batchNormalizationLayer
        reluLayer
        convolution2dLayer(3, 32)
        reluLayer
        maxPooling2dLayer(2, 'Stride', 2)
        dropoutLayer(0.25)

        convolution2dLayer(3, 64, 'Padding', 'same')
        reluLayer
        convolution2dLayer(3, 64)
        reluLayer
        maxPooling2dLayer(2, 'Stride', 2)
        dropoutLayer(0.25)

        convolution2dLayer(3, 64, 'Padding', 'same')
        reluLayer
        convolution2dLayer(3, 64)
        reluLayer
        maxPooling2dLayer(2, 'Stride', 2)
        dropoutLayer(0.25)

        flattenLayer
        fullyConnectedLayer(512)
        reluLayer
        dropoutLayer(0.5)
        fullyConnectedLayer(62)
        softmaxLayer
        classificationLayer
    ];
    options = trainingOptions('rmsprop', 'InitialLearnRate', 0.0001,
'MaxEpochs', 60, 'Shuffle','every-epoch', 'ValidationData',
validation_data_frame, 'ValidationFrequency', 30, 'Verbose',
true,'Plots', 'training-progress');
    model = trainNetwork(training_data_frame, layers, options);
end
```

Seejärel tuleb erinevate meetrikate arvutamise funktsioon. Pythonis kasutati selle jaoks *sklearn.metrics* teeki, kuid MatLabis oli selle jaoks vaja luua eraldi kohandatud funktsiooni, mille põhja allikas on [26], mida muudeti vastavalt vajadusele. See arvutab meetrikaid nagu täpsus, f1 ja spetsiifilisus.

```
function [stats] = stats_of_measure(confusion, verbatim)
    tp = [];
    fp = [];
    fn = [];
    tn = [];
    len = size(confusion, 1);
    for k = 1:len
        tp_value = confusion(k,k);
        tp = [tp, tp_value];
        fp_value = sum(confusion(:,k)) - tp_value;
        fp = [fp, fp_value];
        fn_value = sum(confusion(k,:)) - tp_value;
        fn = [fn, fn_value];
        tn_value = sum(sum(confusion)) - (tp_value + fp_value +
fn_value);
        tn = [tn, tn_value];
    end

    prec = tp ./ (tp + fp); % precision
    prec(isinf(prec)|isnan(prec)) = 0; % Replace NaNs and infinite
values with zeros
    recall = tp ./ (tp + fn); % sensitivity, recall
    recall (isinf(recall)|isnan(recall)) = 0;
    acc = sum(tp) ./ sum(sum(confusion));
    acc(isinf(acc)|isnan(acc)) = 0;
    f1 = (2 .* prec .* sens) ./ (prec + sens);
    f1(isinf(f1)|isnan(f1)) = 0;

    name = ["true_positive"; "false_positive"; "false_negative";
"true_negative"; ...
        "precision"; " recall "; "accuracy"; "F-measure"];
    varNames = ["name"; "classes"; "macroAVG"];
    values = [tp; fp; fn; tn; prec; recall; repmat(acc, 1, len); f1];
    macroAVG = mean(values, 2);
    stats = table(name, values, macroAVG, 'VariableNames',varNames);
    if verbatim
        stats
    end
end
```

Viimases funktsioonis toimub masina poolt pakkumine viiele test elemendile ning seejärel kuvatakse tulemused. Samuti kutsutase siin välja meetrikate arvutamise funktsioon, mille sisendiks on segadusmaatriks.

```
function predict_and_display_results(model, test_data_frame, test_set,
datadir, class_dict,reversed_class_dict)
    pred = classify(model, test_data_frame);
    pred_classes = grp2idx(pred) - 1;
    true_class_names = test_set.label;
    for i=1:length(true_class_names)
        true_labels(i)= lookup(reversed_class_dict,
true_class_names{i});
    end
    true_labels = reshape(true_labels, [], 1);
    confusion = confusionmat(true_labels, pred_classes);
    stats_of_measure(confusion, 1);
    for i=1:length(pred_classes)
        pred_label(i)= lookup(class_dict, pred_classes(i));
    end
```

```
    for i = 1:numel(test_set.image)
        image = imread(fullfile(datadir, test_set.image{i}));
        figure;
        imshow(image);
        title(pred_label(i));
    end
end
```

Koodi kirjutamisel kasutati firma MathWorks professionaalselt tehtud abi lehekülgi [27] ning lisaks ühte tavakasutaja poolt programmeeritud meetrikate arvutamise funktsiooni [26].

ANALYSIS RESULT

	Name	Type	Activations	Learnable Proper...
1	imageinput 64×64×3 images with 'zerocenter' norma...	Image Input	64(S) × 64(S) × 3(C) × 1(B)	-
2	conv_1 32 3×3×3 convolutions with stride [1 1] a...	2-D Convolution	64(S) × 64(S) × 32(C) × 1(B)	Weig... 3 × 3 × 3 … Bias 1 × 1 × 32
3	batchnorm Batch normalization with 32 channels	Batch Normalization	64(S) × 64(S) × 32(C) × 1(B)	Offset 1 × 1 × 32 Scale 1 × 1 × 32
4	relu_1 ReLU	ReLU	64(S) × 64(S) × 32(C) × 1(B)	-
5	conv_2 32 3×3×32 convolutions with stride [1 1] ...	2-D Convolution	62(S) × 62(S) × 32(C) × 1(B)	Weig... 3 × 3 × 32... Bias 1 × 1 × 32
6	relu_2 ReLU	ReLU	62(S) × 62(S) × 32(C) × 1(B)	-
7	maxpool_1 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling	31(S) × 31(S) × 32(C) × 1(B)	-
8	dropout_1 25% dropout	Dropout	31(S) × 31(S) × 32(C) × 1(B)	-
9	conv_3 64 3×3×32 convolutions with stride [1 1] ...	2-D Convolution	31(S) × 31(S) × 64(C) × 1(B)	Weig... 3 × 3 × 32... Bias 1 × 1 × 64
10	relu_3 ReLU	ReLU	31(S) × 31(S) × 64(C) × 1(B)	-
11	conv_4 64 3×3×64 convolutions with stride [1 1] ...	2-D Convolution	29(S) × 29(S) × 64(C) × 1(B)	Weig... 3 × 3 × 64... Bias 1 × 1 × 64
12	relu_4 ReLU	ReLU	29(S) × 29(S) × 64(C) × 1(B)	-
13	maxpool_2 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling	14(S) × 14(S) × 64(C) × 1(B)	-
14	dropout_2 25% dropout	Dropout	14(S) × 14(S) × 64(C) × 1(B)	-
15	conv_5 64 3×3×64 convolutions with stride [1 1] ...	2-D Convolution	14(S) × 14(S) × 64(C) × 1(B)	Weig... 3 × 3 × 64... Bias 1 × 1 × 64
16	relu_5 ReLU	ReLU	14(S) × 14(S) × 64(C) × 1(B)	-
17	conv_6 64 3×3×64 convolutions with stride [1 1] ...	2-D Convolution	12(S) × 12(S) × 64(C) × 1(B)	Weig... 3 × 3 × 64... Bias 1 × 1 × 64
18	relu_6 ReLU	ReLU	12(S) × 12(S) × 64(C) × 1(B)	-
19	maxpool_3 2×2 max pooling with stride [2 2] and pa...	2-D Max Pooling	6(S) × 6(S) × 64(C) × 1(B)	-
20	dropout_3 25% dropout	Dropout	6(S) × 6(S) × 64(C) × 1(B)	-
21	flatten Flatten	Flatten	2304(C) × 1(B)	-
22	fc_1 512 fully connected layer	Fully Connected	512(C) × 1(B)	Weights 512 × 2304 Bias 512 × 1
23	relu_7 ReLU	ReLU	512(C) × 1(B)	-
24	dropout_4 50% dropout	Dropout	512(C) × 1(B)	-
25	fc_2 62 fully connected layer	Fully Connected	62(C) × 1(B)	Weights 62 × 512 Bias 62 × 1
26	softmax softmax	Softmax	62(C) × 1(B)	-
27	classoutput crossentropyex with '0' and 61 other clas...	Classification Output	62(C) × 1(B)	-

Joonis 9. Närvivõrgu mudeli visualisatsioon MatLabis

Command Window

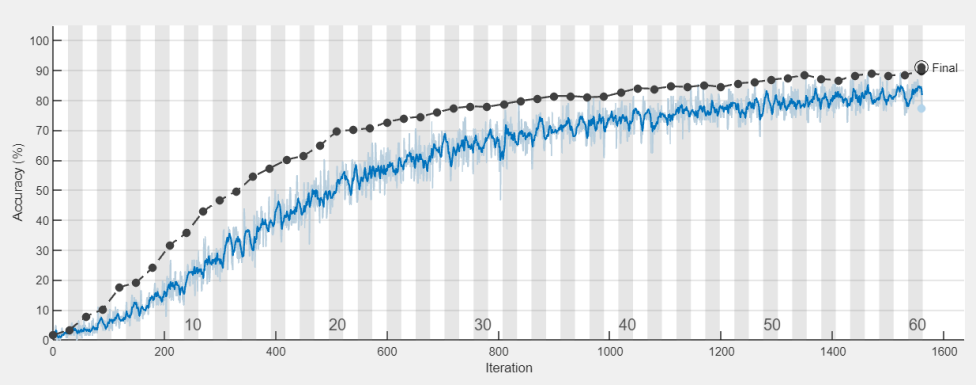
New to MATLAB? See resources for Getting Started.

```
>> matlab
Training on single CPU.
Initializing input data normalization.
|=========================================================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Validation  |  Mini-batch  |  Validation  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |   Accuracy   |     Loss     |     Loss     |      Rate       |
|=========================================================================================================================|
|       1 |           1 |       00:00:41 |        0.00% |        2.42% |       4.2496 |       4.1582 |      1.0000e-04 |
|       1 |          26 |       00:04:46 |        1.56% |        3.23% |       4.1195 |       4.0945 |      1.0000e-04 |
|       2 |          50 |       00:08:15 |        1.56% |              |       4.1149 |              |      1.0000e-04 |
|       2 |          52 |       00:08:55 |        1.56% |        5.45% |       4.0954 |       4.0573 |      1.0000e-04 |
|       3 |          78 |       00:13:00 |        3.12% |        8.89% |       4.0732 |       3.9638 |      1.0000e-04 |
```

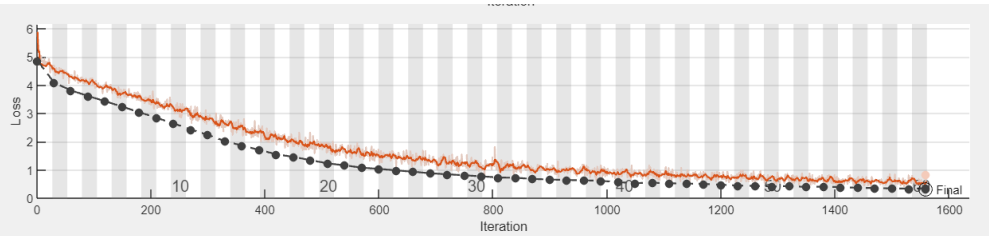
Joonis 10. Käskluste akna vaade

5.2 Tulemused

MatLabi konvolutsiooni närvivõrgu treenimise tulemuseks 60 epohhiga tuli valideerimise täpsus 91%, treenimise täpsus 80% ja testimisel tulid kõik viis pakkumist õigesti, mille tõttu olid ka kõik meetrikate tulemused 1 ehk parim võimalik tulemus. Joonis 11 on graafiliselt näidatud täpsused, kus sinisega kujutakse treenimise täpsust ja mustaga valideerimise täpsust. Joonis 12 on punasega näidatud treenimise kadusid ja mustaga valideerimise kadusid.

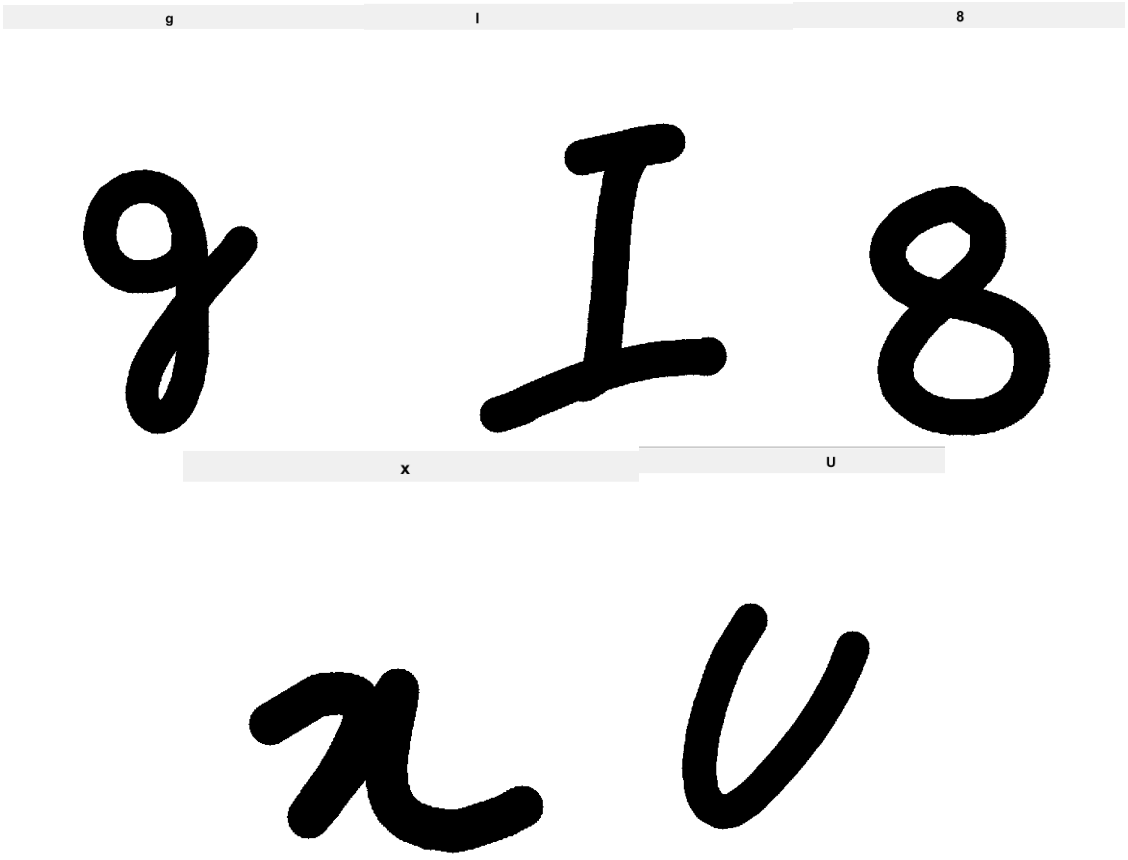


Joonis 11. Närvivõrgu täpsus



Joonis 12. Närvivõrgu kadud

Pildi klassifitseerimise ennustamise funktsioonis pakkus masin kõik numbrid ja tähed õigesti, mida on näha järgmisel joonisel, kus pildil on käsitsi kirjutatud täht või number ja selle kohal on pealkirjana arvuti pakkumine.



Joonis 13. Ennustamise tulemused

6 Analüüs

Pythoni lahenduse puhul oli hea see, et Kerase ja Tensorflow raamistikkude konvolutsiooni närvivõrgu lahendusi leidus internetis rohkem ja seega oli probleemidele lahendust leida lihtsam ja kiirem. MatLabil on vähem kasutajaid, sest see on tasuline tarkvara, ning seega leidus ka vähem informatsiooni spetsiifiliste probleemide korral, aga see-eest on MatLabi firma poolt tehtud dokumentatsioone rohkem ning küsimuste korral vastab MatLabi tugi paari argipäeva jooksul. Kodeerimist alustati Pythoni programmeerimiskeelega, mille tõttu oli hiljem selle võrra lihtsam kodeerida MatLabis.

MatLabi üks hea omadus on see, et MatLabi rakendus on integreeritud programmeerimiskeskkond, kus on olemas erinevad tööriistad nagu teksti redaktor, silur ning süvaõppe tööriistad on lihtsasti lisatavad. See muudab alustamise ja kodeerimise protsessi kiiremaks võrreldes Pythoniga, sest Pythoni jaoks on vaja eraldi teksti redaktorit ning vaja minevaid süvaõppe pakette on vaja alla tõmmata käsurealt, mis võtab tunduvalt rohkem aega alustamisel.

Lõputöös kasutati vanemat sülearvutit, mille tõttu oli treenimisajad suuremad kui modernsematel arvutitel. Pythonis läks 60 epohhi treenimine keskmiselt aega 75 minutit ja MatLabis 240 minutit ehk Pythonis treenimine läks tunduvalt kiiremini. Selle põhjus võib tuleneda asjaolust, et MatLab on eraldi rakendus ning seal on rohkem lisandeid juba kaasas, mida tegelikult lõputöö koodi tegemiseks vaja ei läinud.

Närvivõrgu treenimisel tuli Pythoni valideerimise täpsus 76% ja treenimise täpsus 80% ning MatLabil tuli valideerimine 91% ja treenimine 80% ehk MatLabi lahenduse valideerimise täpsus oli 15% parem ja treenimise täpsus oli sama mis Pythonil.

MatLabi erinevate meetrikate, nagu täpsus, kordustäpsus ja F1, tulemused tulid head, sest masina ennustamisel pandi kõik pakkumised täppi. Pythoni meetrikate tulemused tulid halvemad kui MatLabi omad, sest ennustamisel läks masinal üks pakkumine valesti.

Tabel 1. Meetrikad

	Python	MatLab
Täpsus (*accuracy*)	0.8	1
Kordustäpsus (*precision*)	0.8	1
F1	0.8	1
Tagasivõtt (*recall*)	0.8	1

Tabel 2. Pythoni ja Matlabi erinevuste ülevaade

	MatLab	Python
60 epohhi treenimise kiirus	240 minutit	75 minutit
Lihtsus	Rakendusel on olemas juba vaja minevad vahendid ning tööriistu on lihtne lisada	Vajab eraldi tekstiredaktorit, silurit, paketid vaja eraldi alla tõmmata käsurealt
Treenimise ja valideerimise täpsus	Vastavalt 80% ja 91%	Vastavalt 80% ja 76%
Tasulisus	Tasuline omanduslik vara (*proprietary, closed-source software)*	Vabavara (*open-sourced*)
Kommuun	Väiksem	Suurem
Visualiseerimine	Lihtsam, tööriistad selle jaoks on rakenduses juba olemas	Vajab eraldi tööriistade pakettide alla tõmbamist

7 Kokkuvõte

Bakalaureuse lõputöö raames uuriti erinevaid süvaõppe algoritme nagu rekurrentne närvivõrk ja autoenkooderid. Erinevate algoritmide seast osutus parimaks konvolutsiooniline närvivõrk püstitatud ülesande jaoks – lahendada käsitsi kirjutatud tähtede ja numbrite klassifikatsiooni ülesanne Pythoni ning MatLabiga. Seejärel võrreldi mõlema programmeerimiskeele lahenduse tulemusi. Mõlemal Pythonil ja MatLabil on omad head ja halvad küljed, näiteks oli Pythoniga CNN mudeli treenimine kiirem kui MatLabiga, kuid see-eest oli MatLabi rakendust lihtsam kasutada. Lõputöö autor peab Pythonit paremaks valikuks, peamiselt sellepärast, et see on vabavara ning omab suuremat kommuuni.

Kasutatud kirjandus

[1] D. Reinsel, J. Gantz, ja J. Rydning, „The Digitization of the World From Edge to Core“, 2018.

[2] T. Acharya ja A. K. Ray, „Image Processing Principles and Applications“, 2005.

[3] W. Rawat ja Z. Wang, „Deep convolutional neural networks for image classification: A comprehensive review“, *Neural Computation*, kd 29, nr 9. MIT Press Journals, lk 2352–2449, 1. september 2017. doi: 10.1162/NECO_a_00990.

[4] F. Sultana, A. Sufian, ja P. Dutta, „Advancements in Image Classification using Convolutional Neural Network“, mai 2019, doi: 10.1109/ICRCICN.2018.8718718.

[5] A. Rosebrock, „Deep Learning for Computer Vision with Python Starter Bundle 1st Edition (1.1.0)“, 2017.

[6] N. Sharma, R. Sharma, ja N. Jindal, „Machine Learning and Deep Learning Applications-A Vision“, *Global Transitions Proceedings*, kd 2, nr 1, lk 24–28, juuni 2021, doi: 10.1016/j.gltp.2021.01.004.

[7] I. Goodfellow, Y. Bengio, ja A. Courville, *Deep Learning*. MIT Press, 2016. Vaadatud: 26. oktoober 2023. [Online]. Available at: https://www.deeplearningbook.org/

[8] A. Shrestha ja A. Mahmood, „Review of Deep Learning Algorithms and Architectures“, *IEEE Access*, kd 7, lk 53040–53065, 2019, doi: 10.1109/ACCESS.2019.2912200.

[9] A. Sherstinsky, „Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network“, *Physica D*, kd 404, märts 2020, doi: 10.1016/j.physd.2019.132306.

[10] C. Janiesch, P. Zschech, ja K. Heinrich, „Machine learning and deep learning“, doi: 10.1007/s12525-021-00475-2/Published.

[11] T. Elsken, J. H. Metzen, ja F. Hutter, „Neural Architecture Search: A Survey“, aug 2018, [Online]. Available at: http://arxiv.org/abs/1808.05377

[12] S. Smith, *Digital Signal Processing: A Practical Guide for Engineers and Scientists*, 1. tr. Newnes, 2002.

[13] W. Wang ja Y. Yang, „Development of convolutional neural network and its application in image classification: a survey“, *Optical Engineering*, kd 58, nr 04, lk 1, apr 2019, doi: 10.1117/1.oe.58.4.040901.

[14] I. Vasilev, D. Slater, G. Spacagna, P. Roelants, ja V. Zocca, *Python deep learning : exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow*, 2. tr. Packt Publishing Ltd., 2019.

[15] N. Ketkar ja J. Moolayil, *Deep learning with python: Learn Best Practices of Deep Learning Models with PyTorch*. Apress Media LLC, 2021. doi: 10.1007/978-1-4842-5364-9.

[16] T. ~E. de Campos, B. ~R. Babu, ja M. Varma, „Character recognition in natural images“. Vaadatud: 26. oktoober 2023. [Online]. Available at: https://www.kaggle.com/datasets/dhruvildave/english-handwritten-characters-dataset?rvi=1

[17] „Image Classification Using CNN | Step-wise Tutorial“. Vaadatud: 21. november 2023. [Online]. Available at: https://www.analyticsvidhya.com/blog/2021/01/image-classification-using-convolutional-neural-networks-a-step-by-step-guide/
[18] „OCR with Keras, TensorFlow, and Deep Learning“. Vaadatud: 21. november 2023. [Online]. Available at: https://pyimagesearch.com/2020/08/17/ocr-with-keras-tensorflow-and-deep-learning/
[19] „character classification - neural network“. Vaadatud: 21. november 2023. [Online]. Available at: https://www.kaggle.com/code/icrybaby/character-classification-neural-network
[20] „English handwritten Recognition“. Vaadatud: 21. november 2023. [Online]. Available at: https://www.kaggle.com/code/hamzahjodeh/english-handwritten-recognition
[21] „Keras ImageDataGenerator and Data Augmentation“. Vaadatud: 21. november 2023. [Online]. Available at: https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/
[22] „Flow_from_dataframe: A memory-friendly approach“. Vaadatud: 21. november 2023. [Online]. Available at: https://www.kaggle.com/code/hrmello/flow-from-dataframe-a-memory-friendly-approach
[23] „Precision, Recall, Specificity, Prevalence, Kappa, F1-score check with R“. Vaadatud: 23. november 2023. [Online]. Available at: https://www.datatechnotes.com/2019/02/accuracy-metrics-in-classification.html
[24] „How to Calculate Precision, Recall, F1, and More for Deep Learning Models“. Vaadatud: 23. november 2023. [Online]. Available at: https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/
[25] compsoft-sysmatlab Newsgroup, „The Language of Technical Computing Computation Visualization Programming Getting Started with MATLAB Anonymous FTP server Product enhancement suggestions Subscribing user registration“, 1984. [Online]. Available at: http://www.mathworks.comWeb
[26] „Precision, Specificity, Sensitivity, Accuracy & F1-score“, Vaadatud: 5. detsember 2023. [Online]. Available at: https://se.mathworks.com/matlabcentral/fileexchange/86158-precision-specificity-sensitivity-accuracy-f1-score
[27] „MathWorks Help Center“. Vaadatud: 5. detsember 2023. [Online]. Available at: https://se.mathworks.com/help/index.html

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Liis Karbus

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Tähtede ja numbrite tuvastamine konvolutsiooniliste närvivõrkude abil MatLabi ja Pythoniga“, mille juhendaja on Kristina Vassiljeva
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

02.01.2024

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.