TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Sander Kornet 179291IAIB

PIR-sensori signaalide analüüs ja klassifitseerimine kasutades masinõppe meetodeid

Bakalaureusetöö

Juhendaja: Priit Järv

Phd

Tallinn 2022

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: Sander Kornet

30.05.2022

Annotatsioon

Passiiv-infrapunasensorid on turvaseadmetes laialdast kasutust omavad riistvarakomponendid. Suurendamaks antud komponentidest tulenevat väärtust, oli käesoleva töö eesmärgiks kasutada reaalsetes turvaseadmetes kasutatava konfiguratsiooniga, installatsiooniga ning seadistusparameetritega infrapunasensorite poolt regulaarse töö käigus kogutud andmete peal masinõppe pakutud võimalusi. Masinõppelised meetodid lubavad omandada kogutud andmetest lisateadmisi seadme poolt tajutud olukorrast, et pakkuda seadmeid kasutavale turvatöötajale lisaväärtust juba toimivatest seadmetest. Töö käigus koostati antud tingimustele vastavate seadmete mõõtetulemustest andmestik, rakendati andmestikule masinõppe lahendusi ning analüüsiti loodud lahenduste tulemuslikkust. Kogutud andmetest oldi võimeline andmeid kogunud seadmete konfiguratsiooni muutmata luua klassifitseerimismudel, mis suudab infrapunasignaali analüüsides tuvastada infrapunaenergiat kiiranud keha klassi järgnevatest valikutest: inimene, loom ning liiklusvahend. Käesoleva töö käigus omandatud sügavam mõistmine passiiv-infrapunasensoritest lubab järgnevaid katseid sarnaste seadmete poolt loodud mõõtetulemuste põhjal kergema vaevaga läbi viia.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 5 peatükki, 13 joonist ja 4 tabelit.

Abstract

PIR-Sensor Signal Analysis and Classification using Machine Learning

Passive-infrared sensors are used in a wide range of usecases, from the automatic doors at large shopping centres to intruder detection devices and other security appliances. The benefits of an infrared sensor are, among others, its low electricity usage and effective utilisation as a binary decision maker regarding the presence of a heat radiating body. Once such a body has been found, the sensor often triggers a secondary functionality, such as the opening of a door or an attached camera module in order to photograph an intruder.

The aim of this thesis was to uncover additional ways to utilise the recorded infrared light levels in order to obtain additional information about the situation that triggered the activation of the sensor, utilising the recorded information from devices that are already actively in use without requiring a change in configuration, to assist the security worker assessing the situation and provide them with additional insight about the nature of a possible intrusion.

In order to fulfil the aim of this thesis, a novel dataset containing samples from actively in-use devices in security installations across Estonia and England was curated and analysed. A selection of neural network architectures, selected from assessing relevant literature regarding time-series data and passive-infrared sensors, were used to train models that were tasked with assigning the class labels of human, animal or vehicle to a given signal. The models were evaluated based on a chosen loss function and aimed to minimise the categorical crossentropy of a given model on the created dataset and its train and validation counterparts.

Out of the assessed architectures, a model based on ResNet performed the best out of all tried architectures, and learning was achieved on most tried architectures. This also fulfilled the aim of the thesis, as passive-infrared sensor data was successfully shown to contain additional value in understanding a security situation, specifically containing hidden features that allow a neural network to accurately label the signal as coming from a human, vehicle or animal. This in turn provides invaluable information to the security worker to aid in their decision-making, as a trespassing animal often requires a different response to a trespassing human, likewise with a trespassing vehicle.

The thesis is in Estonian and contains 34 pages of text, 5 chapters, 13 figures, 4 tables.

Lühendite ja mõistete sõnastik

PIR	*Passive Infrared*, passiiv-infrapuna
int	*Integer*, täisarv
CNN	*Convolutional Neural Network,* konvolutsiooniline masinnärvivõrk
LSTM	*Long Short-Term Memory*
FCN	*Fully Convolutional Network,* täielikult konvolutsiooniline võrk.
ReLU	*Rectified Linear Unit*
UCR	*University of California, Riverside*
ResNet	*Residual Neural Network*
One-hot encoding	Andmetöötlusmeetod, mis tõhtustab kategooriliste andmete peal toimiva masinõppe algoritmi tööd.

Sisukord

1 Sissejuhatus **11**
1.1 Eesmärk 12
1.2 Ülesehitus 13
2 Taust ja teema ülevaade **14**
2.1 PIR-Sensor 14
2.2 Detektori konfiguratsioon 15
2.3 Relevantne kirjandus ja võrdlus käesoleva tööga 16
2.3.1 Ajaseeria andmestikele masinõppeliste meetodite rakendamine 16
2.3.2 Passiiv-infrapunaseadmetele rakendatud masinõpe 17
3 Eksperiment **19**
3.1 Andmete kogumine 19
3.1.1 Sündmuste valiku kriteeriumid 20
3.1.2 Andmestiku kirjeldus 23
3.2 Andmete analüüs 24
3.2.1 Visuaalne vaatlus 24
3.2.2 Klasside andmete analüüs 24
3.2.3 Mõõtmistulemuste formaadi analüüs 25
3.3 Andmetele masinõppe meetodite rakendamine 26
3.3.1 Konvolutsiooniline masinnärvivõrgustik (CNN) 27
3.3.2 Rekurrentne masinnärvivõrgustik (LSTM) 29
3.3.3 ResNet (Residual neural network) 31
4 Tulemuste analüüs ning valideerimine **33**
4.1 Kaofunktsioon 33
4.2 Tulemuste valideerimine 34
4.3 Tulemuste analüüs 35
4.3.1 Konvolutsiooniline masinnärvivõrgustik (CNN) 35
4.3.2 Rekurrentne masinnärvivõrgustik (LSTM) 36
4.3.3 ResNet 38

4.4 Tulemuste järeldused 39
4.5 Võimalikud arengud 41
4.5.1 Hüperparameetrite optimisatsioon 41
4.5.2 Uudne mudel 41
4.5.3 Masinõppe meetodite välised lahendused 41
4.5.3 Alternatiivsete lisaandmete kogumine 42
5 Kokkuvõte 44
Kasutatud kirjandus 46
Lisa 1 - Lihtlitsens lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 49
Lisa 2 - Klassifitseerimisülesande hõlbustamiseks loodud tööriista kood 50
Lisa 3 - Skaleerimisfunktsioon 58
Lisa 4 - LSTM mudeli loomise ja treenimise kood 59
Lisa 5 - CNN mudeli loomise ja treenimise kood 62
Lisa 6 - ResNet mudeli loomise ja treenimise kood 65
Lisa 7 - ResNet mudeli struktuur 69

Jooniste loetelu

Joonis 1. Antud töös kasutusel oleva PIR-sensori läätse kirjeldus [6].

Joonis 2. Pilt antud töö käigus kasutatavatest detektorseadmetest.

Joonis 3. Kuvatõmmis töö käigus loodud klassifitseerimistööriistast.

Joonis 4. Histogramm kõikide sündmuste individuaalsetest mõõtmistest, mis vastasid sündmuste valiku esimesele kriteeriumile.

Joonis 5. Joonisel 4 kujutatud histogrammi keskosa.

Joonis 6. Antud töös kasutatud CNN-i struktuur.

Joonis 7. Antud töös kasutatud LSTM-närvivõrgu struktuur.

Joonis 8. CNN mudeli treenimise kulg.

Joonis 9. LSTM mudeli treenimise kulg.

Joonis 10. Ühekihilise LSTM-mudeli skeem.

Joonis 11. Ühekihilise LSTM mudeli treenimise kulg.

Joonis 12. ResNet-arhitektuuril põhineva mudeli treenimise kulg.

Joonis 13. Kahest samas suunas liikuvast kehast tehtud pildid ning nende signaalid.

Tabelite loetelu

Tabel 1. Antud töös kasutatud CNN mudeli hüperparameetrid.

Tabel 2. Antud töös kasutatud LSTM mudeli hüperparameetrid.

Tabel 3. Antud töös kasutatud ResNet mudeli hüperparameetrid.

Tabel 4. Antud töös kasutatud mudelite tulemused.

1 Sissejuhatus

Passiiv-infrapuna sensorid, edaspidi PIR-sensorid, on laialdase kasutusega mitmes valdkonnas, sealhulgas ka turvalisuse valdkonnas. Selliste sensorite eelis on väga madal energiakulu, mida saab efektiivselt rakendada binaarsete otsuste tegemiseks – kas soojusttekitav keha eksisteerib sensori vaateväljas, või sellist keha ei eksisteeri. Peale keha tuvastamist on võimalik käivitada sensorit kasutava seadme mõni teine funktsioon, näiteks ukse avamine või filmimise alustamine [1]. Sellised süsteemid on kasutusel nii kaubanduskeskuste automatiseeritud ustes kui ka automaatsete turvakaamerate detektorites.

Käesoleva töö eesmärgiks on omandada PIR-sensorite poolt mõõdetud signaalidelt lisainformatsiooni soojusttekitava keha kohta, rakendades mõõdetud signaali väärtustest tekkinud informatsioonil masinõppe meetodeid. Eesmärgi täpsem kirjeldus on antud peatükis 1.1.

Masinõpe on andmeanalüüsi meetod, mis automatiseerib analüütiliste mudelite loomist. Tegemist on tehisintellekti allharuga, mis suudab minimaalse inimesepoolse sekkumisega õppida andmetest, identifitseerida andmetes mustreid ning nende põhjal teha otsuseid [2]. Masinõppe tulemused sõltuvad esiteks kogutud andmestikust ning teiseks, andmestikule rakendatud metoodikast. Masinõppe protsess hõlmab loodud mudeli kihtides olevate peidetud väärtuste järkjärgulist muutmist epohhide kaupa, üritades saavutada etteantud parameetri parim võimalik väärtus.

Kuna PIR-sensorid on väga laialdase kasutusega ning väga madala energiakuluga, on oluline antud lahendusest tulenevaid võimalusi maksimaalselt ära kasutada ning uusi võimalusi avastada. Antud töö raames käsitletud PIR-sensorite konfiguratsioon on väga levinud turvasüsteemides, liikumisandurites ning automaatsetes valguslahendustes [3], kuid relevantset kirjandust uurides, millest annab käesolev töö ülevaate teises peatükis, on signaalianalüüsi rakendatud laboratoorsetes tingimustes kogutud andmetele või kasutatud mitmete sensorite või sensoriribade kombineeritud väljundeid, mis ei vasta üldjuhul päriselulistele PIR–sensorite rakenduskonfiguratsioonidele.

Masinõppe tulemused sõltuvad lisaks meetoditele ka andmestikust, mille peal masinõppe lahendusi rakendatakse, mistõttu on oluline, et masinõppe jaoks kasutatav andmestik oleks kogutud samadel tingimustel kui andmed, mille peal treenitud mudel hakkab igapäevaselt töötama. Seda probleemi suurendavad PIR-sensorite negatiivsed küljed, milleks on keskkonnatundlikkus seoses temperatuuriga või objekti aeglane liikumine [4], mis ei kajastu laboratoorsetes tingimustes kogutud andmetega ning võivad kaasa tuua laboratoorsete andmetega treenitud mudeli jõudluse ja täpsuse halvenemise, kui kasutada välitingimustes installeeritud sensorite väljundit, kus tegurid nagu taustatemperatuur, kaugus infrapunakiirgust kiirgavast kehast või peegeldustest tulenevad temperatuurikõikumised muudavad analüüsitavaid andmeid piisavalt, et mudeli tööd segada ja mudeli hinnanguid nagu täpsus negatiivselt mõjutada.

Käesolevas töös käsitletud PIR-sensorid asuvad kasutuses olevate valveseadmestike koosseisu kuuluvates kaamerates, mis on installeeritud valveobjektidele ning on aktiivses kasutuses valvetegevuse hõlbustamiseks. Andmestiku loomiseks kasutatud kaamerad asuvad peamiselt Eestis ning Inglismaal. Andmestiku kogumiseks on tehtud koostööd valvelahendust pakkuva firmaga, kellele on kliendid andnud loa valveobjektil salvestatud andmeid töödelda. Töö käigus on tagatud kogutud andmete maksimaalne võimalik anonüümsus, töö autor on volitatud andmetöötleja rolliga ning firmaga seotud olnud ligikaudu üks aasta.

1.1 Eesmärk

Käesoleva töö eesmärk on edendada passiiv-infrapuna sensorite mõistmist läbi masinõppe meetodite rakendamise sensorite väljundile, mis on kogutud päriselt kasutusel olevatest passiiv-infrapuna seadmetest, mistõttu on tõenäolisem loodud lahenduse rakendatavus päriselt kasutusel olevatele sensoritele ilma suurema lisatööta. Töö täpsem eesmärk oleks PIR-sensori salvestatud mõõtmistulemustest olla võimeline ära identifitseerida objekti tüüp, mis selle infrapunasignaali tekitas, kas loom, inimene või liiklusvahend. See on ülimalt oluline olukordades, kus kaamerapildist ei ole võimalik tuvastada infrapunasignaali allikat, et tagada valveobjekti puutumatus.

Selle eesmärgi saavutamise jaoks on tarvis läbida järgnevad sammud: 1) Koostada masinõpet toetav andmestik PIR-sensorite väljunditest mis töö autorile kättesaadavad on, 2) rakendada andmestiku peal valik masinõppe lahendusi, 3) analüüsida ja valideerida saadud tulemusi ning 4) teha järeldusi PIR-sensorite hetkelisest potentsiaalist ja võimalustest seoses masinõppega töö raames piiritletud tingimustel kui ka andes suunitlusi arusaamise jätkuvaks edendamiseks, tuues välja töö raames käsitlemata jäänud võimalused ning töö raames piiritletud tingimuste poolt põhjustatud olukorra eripärad ning potentsiaalsed arengusuunad.

1.2 Ülesehitus

Antud töö koosneb viiest osast. Sissejuhatus annab ülevaate töö eesmärkidest, töö loomise motivatsioonist ja töö struktuurist.

Teises peatükis antakse ülevaade töö taustast, kasutusel olevatest seadetest ja süsteemidest ning lühitutvustus relevantsest kirjandusest ja siiani loodud lahendustest.

Kolmandas peatükis kirjeldatakse üksikasjalikult töö andmestiku loomisprotsessi ja andmestikele rakendatud masinõppe meetodeid.

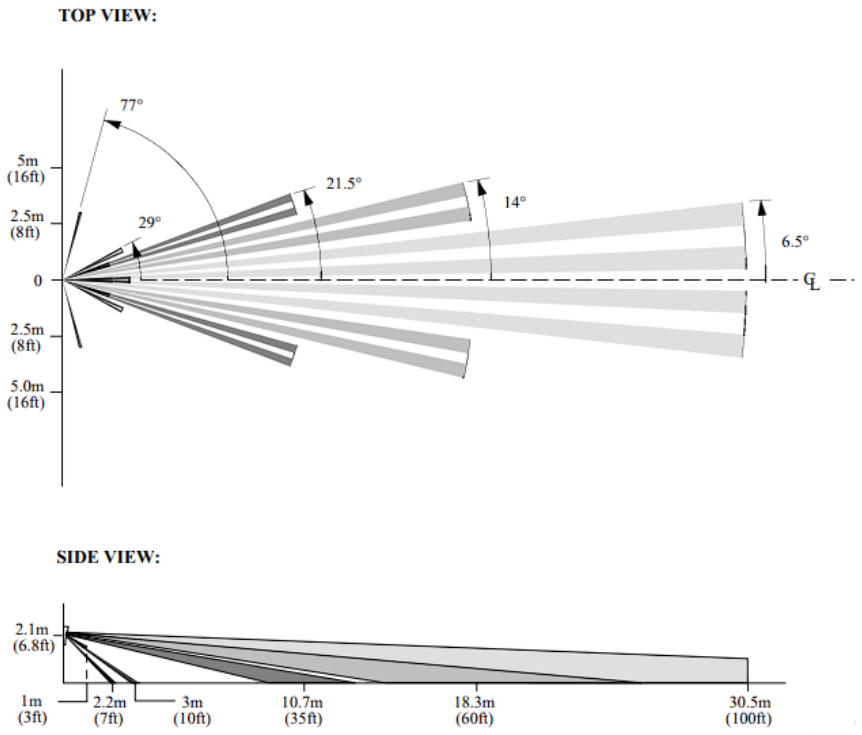
Neljandas peatükis toimub analüüsi ning valideerimise meetodite kirjeldamine, kolmandas peatükis omandatud tulemustele kirjeldatud meetodite rakendamine ning järelduste tegemine.

Viiendas peatükis antakse tehtud töö ning leitud lahenduste kokkuvõte ning tehakse soovitusi edasiseks uuringuks.

2 Taust ja teema ülevaade

2.1 PIR-Sensor

Passiiv-infrapuna tunnetus toimib läbi muutuse tunnetamise kiiratud infrapunakiirguse hugas [4]. Püroelektrilised elemendid sensori sees tunnetavad infrapunaenergiat sensori poolt kaetud alas, ning suure infrapuna-energia hulga muutuse korral element soojeneb või jahtub, mille tagajärjel tekivad elemendil elektrilaengud [5]. Infrapunakiirguse suunab väikesele kristalli osale *Fresnel*-lääts, mis jaotab valve all oleva ala eristatavateks tsoonideks (joonis 1). Infrapunaenergiat kiirgava keha sattumisel ühte tsooni tekitab püroelektrilises elemendis mõõdetava elektrilaengu, mille väärtus (antud töös kasutatud seadmete puhul erinevus signaali keskväärtusest) sõltub püroelektriku poolt tuntava kiirguse hulgast.



Joonis 1. Antud töös kasutusel oleva PIR-Sensori läätse kirjeldus [6].

Joonisel 1 on näha töös kasutatud PIR-sensorite teoreetilised detektsioonitsoonid. Nendes tsoonides asuvate kiirgusallikate infrapunaenergia koondatakse läätse poolt püroelektrilisele elemendile, millest tekkiv elektrilaeng on antud töös kasutatavate PIR-andurite (edaspidi detektorite) tööpõhimõte, detektor on kujutatud joonisel 2.

Joonis 2. Pilt antud töö käigus kasutatavatest detektorseadmetest.

Detektsioonid toimuvad peamiselt 2,2m kuni 3 meetri vahemikus ning 9 kuni 30 meetri vahemikus. Detektori antud seadistuse puhul ei ole võimalik määrata, kui kaugel ning millises tsoonis infrapunakiirgust kiirgav keha täpselt liikus. Detektori vasakpoolsed ning parempoolsed tsoonid tekitavad mõõdetud signaalis kõikumisi nivooväärtusest kõrgemale ja madalamale, mistõttu on osadel juhtudel signaalist võimalik visuaalselt näha objekti liikumissuunda paremalt vasakule või vastupidi sõltuvalt signaalile jäänud lainest.

2.2 Detektori konfiguratsioon

Antud detektorid mõõdavad infrapunakiirgust oma detektsioonitsoonidest kümme korda sekundis ning transleerivad mõõtmise momendil püroelektrikul leiduva laengu üheks väärtuseks 0 ja 255 vahel. Seade hoiab lühimälus viimased 30 sekundit mõõtmistulemusi (300 mõõtmist), kasutades mälu efektiivse rakendamise eesmärgil

kuueteistkümnendsüsteemi. Kirjeldatud konfiguratsioon tagab minimaalse energiakulu ning riistvara keerukuse, säilitades võimekuse edukalt töötada detektorina. Mõõtmistulemused edastatakse edasi serverile mis haldab kogu edasist valvetegevust kui seadme suletud lähtekoodiga sisemine algoritm tuvastab piisavalt suurt infrapunaenergia kõikumist. Mõõtmistulemustele lisaks edastatakse serverile veel automaatselt salvestatud pilt olukorrast, mis energia kõikumise tekitas ning teisi antud töö raames mitteolulisi andmeid. Edaspidi nimetatakse siin töös ühte mõõtmistulemust ja sellega seonduvat informatsiooni üheks sündmuseks (inglise keeles *Event*). Kuna sündmuse mõõtmistulemus on sündmuse tekitanud keha infrapunasignaali digitaalne representatsioon, kasutatakse käesolevas töös terminit mõõtmistulemus ja signaal edaspidi vaheldumisi. Käesoleva töö käigus analüüsitakse ainult seadmete regulaarse töö kaudu serverisse jõudnud sündmuste andmeid.

2.3 Relevantne kirjandus ja võrdlus käesoleva tööga

Järgneb kiire ülevaade töö loomisele eelnevalt kirja pandud allikatest, mis kirjeldavad PIR-sensorite väljunditele rakendatud masinõppe meetodeid. Antud kirjandus on kogutud autori poolt töö tegemise jooksul, omandatud tagasisidena töö ülesandepüstituse arutelu käigus või pakutud koostööd teinud firma poolt uurimiseks.

2.3.1 Ajaseeria andmestikele masinõppeliste meetodite rakendamine

Zhiguang, W. et al. [7] kirjeldab oma töös kolme erineva närvivõrgu-arhitektuuril loodud mudeli rakendamist tehtud valiku UCR ajaseeria andmestike [8] peal, kasutamata suurel hulgal andmete eeltöötlust, ning võrdleb seda teiste väljapakutud mudelitega. Parimad tulemused saavutas *Fully Convolutional Network* ehk FCN, kuigi autor toob välja ka ResNet-i hea tulemuslikkuse.

Allikas [8] leiduvad ajaseeria andmestikud on ajaseeria andmete põhiste uuringute jaoks loodud keskne andmestike kogum, et antud vallas töötavad uurijad saaksid oma uuringuid kergemini võrrelda, opereerides sarnastel andmetel.

Smirnov, D. et al. [9] edendab ja võtab kokku [7] tehtud uuringuid, rakendades valiku masin-närvivõrke kõigi UCR ajaseeria andmestike [8] peal, ning toob välja rekurrentsete närvivõrkude paremaid tulemusi kasutades LSTM närvivõrgukihte.

2.3.2 Passiiv-infrapunaseadmetele rakendatud masinõpe

Kim, D. H., et al. [10] kasutas passiiv-infrapunasensorit, et koguda sensori ligidalt liikuva inimese olemasolu kohta informatsiooni, saavutades CNN-põhise närvivõrguga täpsuse 97.62% treeningandmete, validatsioonitäpsuse 80.5% ja testimistäpsuse 72.8%. Võrreldes käesoleva tööga on andmete kogumise tihedus magnituudides suurem ning uuritava subjekti kaugus mõõteseadmest kordades väiksem, mistõttu on andmetes leiduvate tunnuste, mis viitaksid inimese olemasolule, leidmine väga tõenäoline. Antud infrapunadetektori seadistus on aga väga ebatüüpiline turvasüsteemides, kuna pidev tihe mõõtmine vajab suurel hulgal elektrit ning ei ole praktiline näiteks pikkade ja kõrvaliste detektorseadmete jaoks, mille eesmärk on valvata maa-alade piire ning lihtsa liikumisanduri jaoks ei ole nii tihe mõõtetulemuste kogumine vajalik.

K. A. Muthukumar, et al. [11] on kasutanud passiiv-infrapunasensorite kogumikku, et klassifitseerida sensori vaateväljas oleva isiku tegevusi, saavutades mitme sensori informatsiooni kasutades ligikaudu 90%-se täpsuse. Katsete käigus oli valitud keskkonnaks kaks siseruumi ning tagatud laboratoorsed tingimused läbi kliimaseadme. Sellised tingimused ei vasta taaskord paljude valveseadmete töötingimustele, kus välised tegurid nagu temperatuurikõikumised miinuskraadidest kuni üle 30°C, temperatuurikõikumised tänu ilmastikutingimustele nagu pilvkate, udu ning tuul ja muud segajad on väga tüüpilised. Töös on ka välja toodud, et teiste objektide toomine sensorite vaatevälja tekitab väljapakutud lahenduses probleeme. Turvaseadmed on tihti installeeritud kohtadesse kus võib esineda mitmeid ebaolulisi infrapunakiirguse tekitajaid, näiteks taimestik või kliimaseadme väljundiava.

Juba kasutuses olevate turvaseadmete poolt mõõdetud andmestike peal tehtud katsetusi või uuringuid, et suurendada seadmelt saavutatud väärtust kaasates seadmelt pärinevat informatsiooni turvalisuse tagamisse läbi masinõppelise andmetöötluse ning

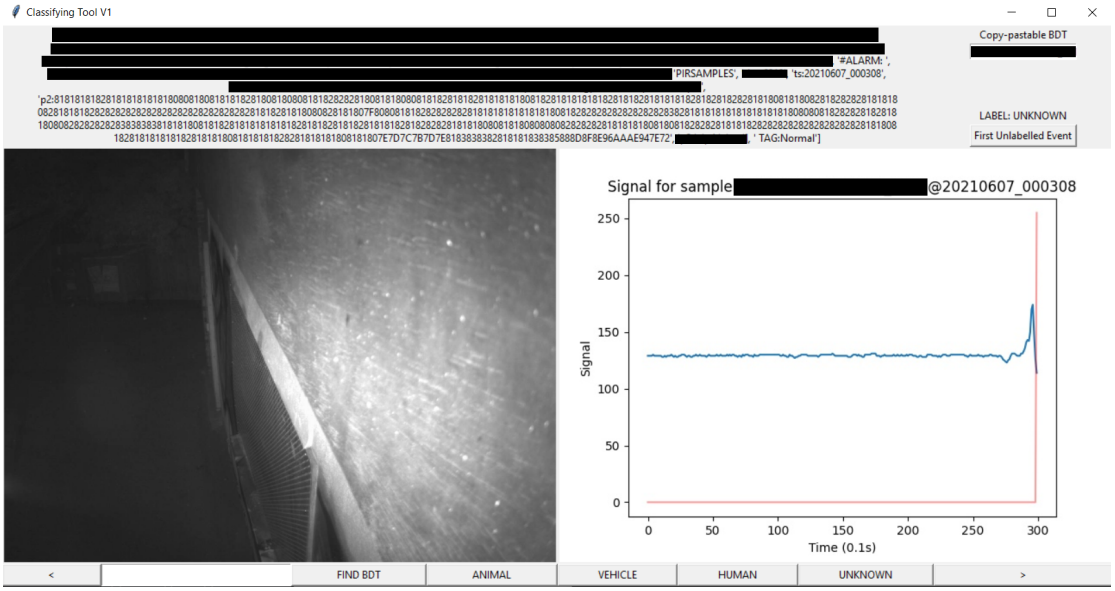
informatsiooni kogumise, pakkudes turvatöötajale resultaadina kõrgendatud arusaamist tekkinud olukorrast mida on võimalik arvesse võtta, et valveobjekti turvalisus ja puutumatus tagada, käesoleva töö autor töö kirjutamise ajal ei leidnud.

3 Eksperiment

3.1 Andmete kogumine

Käesoleva töö andmestik on valim ühe päeva jooksul valvelahendust pakkuva firma serveritesse jõudnud sündmustest. Üks päev annab hea ülevaate nii öisetest kui päevastest tingimustest ning vähendab andmestiku mõjutavate välistegurite esinemistõenäosust, mis suudaksid tekitada kogutud andmete põhjal treenitud masinõppelises mudelis anomaaliaid, mis võivad viia tagajärgedeni nagu madal võime generaliseerida teiste päevade andmestike tõttu või temperatuuri anomaaliate tõttu kallutatud andmestik. Töö jaoks valitud päev oli keskmisest sündmusterohkem kuid välistegureid, nagu ilm, arvestades igati ootuspärane. Antud töö käigus kasutatakse ühte seitsmekümnendikku kõikidest sündmustest, mida süsteem antud päevas analüüsis. Osa päevas mõõdetud sündmusi eemaldati peatükis 3.1.1 kirjeldatud tingimustele põhinedes. Kuna antud töö raames loodud andmestik on käsitsi klassifitseeritud, oleks olnud rohkemate sündmuste klassifitseerimiseks vaja rohkem klassifitseerijaid kui töö autor.

Sündmustest võetakse PIR-sensori mõõtmistulemused, ning kasutades teisi sündmuse andmeid, näiteks sündmusest salvestatud pilti, klassifitseeritakse mõõtmistulemus ühte klassi kokku võimalikust kolmest klassist, milleks on: 1) Inimene, 2) liiklusvahend ja 3) loom. Juhul kui sündmuse teistest andmetest ei ole võimalik sündmuse tekitanud allikat määrata, jäeti antud sündmus andmestikust välja märgistusega *Unknown*. Klassifitseerimisülesande hõlbustamise jaoks on töö autor loonud lihtsa tööriista kasutades *tkinter*-it [12], mis on *Python*-i *package* graafiliste kasutajaliideste kiireks loomiseks. Kuvatõmmis tööriistast on näha joonisel 3.



Joonis 3. Kuvatõmmis töö käigus loodud klassifitseerimise tööriistast. Musta ribaga on kaetud sensitiivne informatsioon seoses valvetegevuses kasutatava informatsiooniga ning informatsioon, mis ei ole käesoleva töö raames oluline. Antud sündmust ei kasutatud andmestiku loomisel.

3.1.1 Sündmuste valiku kriteeriumid

Sündmuste valik, mis klassifitseerimiseks valiti, oli tehtud kolmel kriteeriumil.

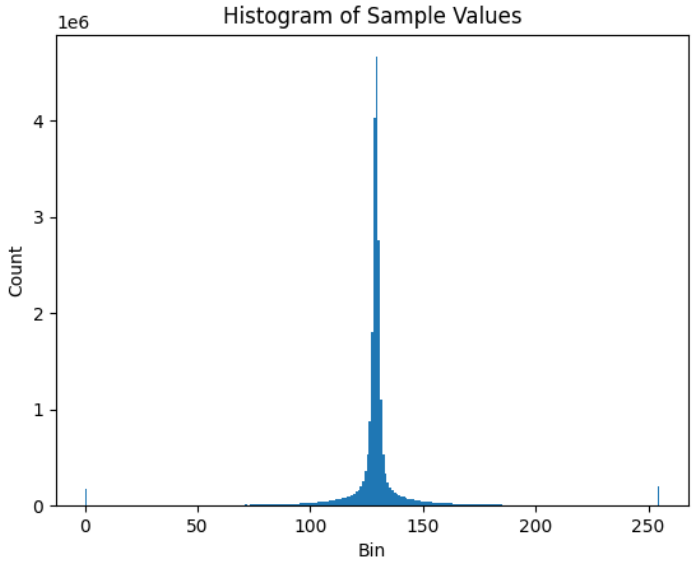
Esimeseks kriteeriumiks oli sündmuse terviklikkus. Selle all mõeldakse seda, et serverisse jõudnud informatsioonis ei ole puudu ühtegi andmepunkti mis tagab suurima võimaluse, et seadme töös ei ole tekkinud vigu. Vigase seadme poolt mõõdetud sündmuse analüüsi kaasamisel on oht segada masinõppe poolt loodud mudeli tööd.

Teiseks kriteeriumiks oli signaali täieliku kuju kajastumine saadetud andmetes. Tänu võimalusele, et püroelektrikult mõõdetud elektrilaeng võib ületada vahemikku mida detektor on võimeline eristama, on detektor seadistatud kõiki lubatust kõrgemaid väärtusi ja lubatust madalamaid väärtusi mõõtma vastavalt kui maksimaalset või minimaalselt väärtust, mis on selgelt eristatav teistest signaalidest kui mõõtmistulemuse lõpus on jada miinimumväärtusi (0) või maksimumväärtusi (255), mis lõikab ära vastavalt lainenõo või laineharja mis omakorda võib sisaldada kriitilise tähtsusega

väärtusi, muutes antud sündmusest saadud mõõtmistulemuse masinõppe rakendamise eesmärgi jaoks kõlbmatuks.

Kolmandaks kriteeriumiks oli signaali puhtus. Olukordades, kus signaalis on silmaga nähtav taustamüra, ehk suurem osa 30 sekundi pikkusest mõõtmisest on täis pidevaid kõikumisi, mis on väljaspool signaali oodatud väärtusi kui detektor on puhkeolekus ehk sellel ajal kui ei ole tuvastatud sündmust. Sellise kõikumise puhul võib olla tegemist vigase seadmega, halva paigaldusega või kõrvaliste teguritega, mis võivad mõjutada antud mõõtmistulemust õppimiseks kasutavat masinõppe algoritmi negatiivselt.

Eesmärgiga paremini aru saada signaalide oodatavatest väärtustest, loodi histogramm, milles on kujutatud kõikide sündmuste individuaalsete mõõtmiste väärtused. Sündmus pidi esmaselt vastama esimesele põhimõttele, et tema mõõtmistulemuses sisalduvaid mõõtmisi antud histogrammi loomisel arvestatakse.



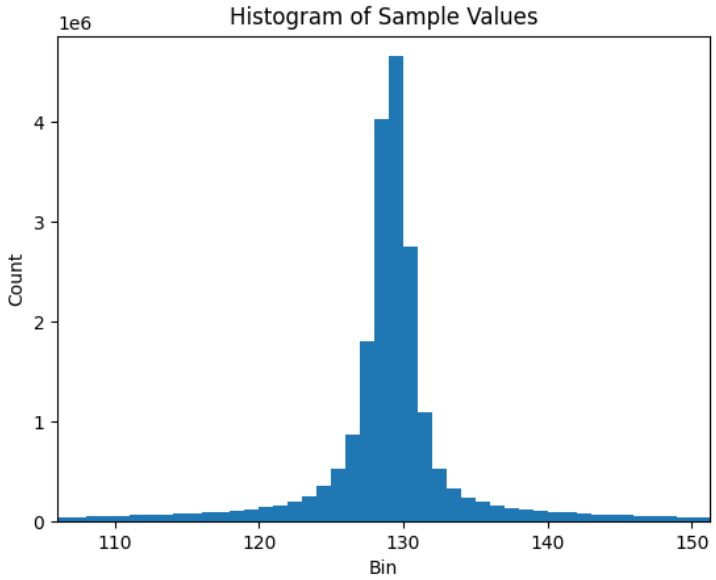
Joonis 4. Histogramm kõikide sündmuste individuaalsetest mõõtmistest, mis vastasid sündmuste valiku esimesele kriteeriumile.

Histogrammilt on näha, et mõõtmistulemused meenutavad visuaalsel vaatlusel normaaljaotust. Antud normaaljaotuse standardhälve (kalkuleeritud *Numpy*-i *package*-i

std käsuga) on 23.78. Histogrammi otstes olevate 0 ja 255 väärtuste jooned viitavad signaalidele, mis rikuvad sündmuste valiku teist kriteeriumit.

Antud analüüsist tulenes vajadus filtreerida kogutud andmetest välja potentsiaalsed vigased seadmed (kriteerium üks), signaalid mis ei ole terviklikud (kriteerium kaks) ning signaalid, millel esineb piisavalt hulgal segatud müra (kriteerium kolm).

Vigaste seadmete kontroll hõlmas kontrollimist, kas kõik ühe sündmusega seotud informatsioon, sealhulgas näiteks pilt, on edukalt serverisse jõudnud. Terviklike signaalide filtreerimine realiseeriti läbi kontrolli, et ükski mõõtmistulemuses sisalduv individuaalne mõõtmine ei oleks väärtustatud 0 või 255-ga. Signaalide müra filtreerimise jaoks otsustas töö autor arbitraarselt lugeda need signaalid müravabaks, mille väärtused on 90% mõõtmistulemusest kuni 2 mõõtühiku võrra kaugemal kõige levinumad mõõtmistulemusest, mis antud andmestiku puhul on väärtus 129. Kahe mõõtühiku parameeter osutus kõikidest proovitud variantidest parimaks, kuna ühe mõõtühiku kasutamine jättis alles liiga vähese hulga signaale ning kolme ja rohkema mõõtühiku kasutamine parameetrina oleks andmestiku suuruse muutnud viiekohaliseks numbriks, mille õigeaegne klassifitseerimine oleks vajanud kõrvalist abi. Kõige levinum mõõtmistulemuse väärtus on selgelt nähtaval joonisel 5.



Joonis 5. Joonisel 4 kujutatud histogrammi keskosa.

Nendel kriteeriumitele tuginedes koostati filter, mis võttis antud kriteeriumid arvesse ning tagastas nimekirja tingimustele vastavatest sündmustest. Antud sündmused kompileeriti töös kasutatavaks andmestikuks.

3.1.2 Andmestiku kirjeldus

Eelnevas peatükis kirjeldatud põhimõtetele toetudes loodi andmestik, mis koosneb 1088-st klassifitseeritud mõõtmistulemusest. Iga mõõtmistulemuse ette lisati klassi identifikaator – 0 kui oli tegemist loomaga, 1 kui oli tegemist inimesega ning 2 kui oli tegemist liiklusvahendiga. Ühe andmerea pikkus on 301 – klassifikaator vahemikus 0 kuni 2 (kaasa arvatud) ning 300 individuaalset mõõtmist mis moodustavad ühe mõõtmistulemuse. Mõõtmistulemused viidi inimloetavuse suurendamise eesmärgil kuueteistkümnendiksüsteemilt üle kümnendiksüsteemile.

3.2 Andmete analüüs

3.2.1 Visuaalne vaatlus

Andmete analüüsimist alustati osade mõõtmistulemuste graafilist esitamist koos kontrolliga, kas erinevaid klassifikaatoreid omavatel mõõtmistulemustel on võimalik visuaalsel vaatlusel vahet teha. Triviaalse visuaalse inspektsiooniga eristatavatele signaalidele ei oleks vajalik rakendada masinõppelisi lahendusi. Erinevatelt infrapunaenergiat kiirgavatelt allikatelt pärinenud infrapuna-signaali mõõtmistulemuste graafilistel esitlustel ei olnud võimalik visuaalset vaatlust kasutades vahet teha.

3.2.2 Klasside andmete analüüs

Järgnevalt vaadati üle andmestiku jagunemine kolmeks klassiks ning nende klasside arvulised näitajad. Kokku jagunesid 1088 mõõtmistulemust kolme klassi järgnevalt: klassi loom sattus 85 mõõtmistulemust, klassi inimene 347 mõõtmistulemust ning klassi liiklusvahend 656 mõõtmistulemust. Selline jaotus on küll ootuspärane arvestades autori senist kogemust uurimise all olevast probleemiruumist (*problem domain*), kuid jaotust uurides on selge, et tegemist on ebaühtlase jaotusega.

Ebaühtlane jaotus võib segada masinõppe mudeli täpset hindamist [13]. Allikas [13] on kirjeldatud antud probleemile viis lahendust. Selle töö puhul osutus valitud lahenduseks *minority-class upsampling* ehk harvaesinevate klasside mitmekordne andmestikku lisamine. *Majority-class downsampling* oleks antud olukorras halb lahendus, kuna vähendaksime andmestiku suurust ligikaudu 75% ning teised allika poolt pakutud lahendused ei lahenda otseselt klasside ebaühtlase jaotuse probleemi.

Antud andmetest koostati treening- ning validatsiooniandmestik. Kuna klassi loom oli esindatud väga vähe, pidi andmete jaotamisel valima standardse validatsiooniandmestiku, mis oleks kaasa toonud klassi loom väga madalat arvu treeningandmestikus või väikese validatsiooniandmestiku, mis oleks lubanud rohkem loom-sildistatud mõõtetulemusi treeningandmestikus, vahel. Käesoleva töö jooksul valiti antud variantidest teine. Validatsiooniandmestiku jaoks valiti 15 suvalist

eksemplari igast klassist, mis eemaldati treeningandmestikust, seejärel rakendati *Minority-class upsamplingut* töö käigus klassile loom, sampeldades igat allesjäänud looma klassi näidet kolm korda. Treeningandmestiku lõplikud numbrid oli 332 klass inimene eksemplari, 210 klass loom eksemplari ja 641 klass liiklusvahend eksemplari.

3.2.3 Mõõtmistulemuste formaadi analüüs

Mõõtmistulemuste puhul on tegemist ajaseeria andmestikuga, mis hõlmab enda all ühe infrapunasensori 30 sekundit väljastatud väärtusi, mille käigus on infrapunaenergiat kiiranud keha sensori vaateväljast läbi käinud. Joonisel 1 peatükis 2.1 näidatud detektsioonialad määravad ära sensori vaatevälja.

Kuna lõppeesmärgiks on klassifitseerimata mõõtmistulemuste klassifitseerimine läbi loodud mudeli, on meie klassifitseerija lõpptulemuseks üks *int* väärtusvahemikus [0, 2]. Kuna klassifitseerijal ei ole tarvilik anda mõõtmistulemusele mitu silti, oleks mõistlik kasutada *sparse loss*-funktsiooni, mis ei nõua *one-hot encoding*-ut ja seetõttu lisatööd, et muuta andmestikku kategooriliseks.

Kuna andmete näol on tegemist mõõtmistulemustega kuni detektor on tuvastanud piisavalt suurt infrapunaenergia kõikumist, on andmetes kajastatud ainult objekti poolt sensori alasse sisenedes tekitatud infrapunaenergia mõõdud, kuid mitte sensori alast lahkuvad mõõdud. Praegune detektori seadistus lubab väga hästi tuvastada infrapunaenergia kõikumist ning läbi selle objekti sattumist detektori valvetsooni, kuid ei ole hea seadistus, et omandada täispilt sellest infrapunaenergiast, mida objekt kiirgab läbides detektori poolt valvatud ala. Olukordades, kus objekti lahkumise poolt tekitatud infrapunaenergia väärtused võivad sisaldada väärtuslikku infot mida detektori valvetsooni sisenemine ei sisalda, läheb see info praeguse detektori konfiguratsiooniga kaduma ning loodavad mudelid nendest potentsiaalsetest andmetest õppida ei saa.

Analüüsi hetkel olid mõõtmistulemuse väärtused skaalal 0 kuni 255. Masinõpe töötab paremini standardiseeritud andmestikul, mistõttu skaleeriti enne masinõppe meetodite rakendamist andmed vahemikku -1 kuni 1 eesmärgiga vähendada andmete väärtuste kaugust üksteisest. Skaleerimise jaoks kasutatud funktsioon on välja toodud Lisas 3.

3.3 Andmetele masinõppe meetodite rakendamine

Antud töös kasutatud valik masinõppe meetodeid, mida andmetele rakendati, valiti töö teises osas tehtud relevantse kirjanduse uurimise põhjal. Konvolutsioonilisi närvivõrke mainiti [7], [9] ja [11] nende hea tulemuslikkuse eest ajaseeria andmestikke analüüsides. Wang, Z. et al. [7] oli esile tõstnud ResNet-i arhitektuuri. Smirnov, D. et. al. [9] ning vestlused juhendajaga olid esile tõstnud rekurrentseid närvivõrke kui ajaseeria andmestikel varasemalt hästi opereerinud lahendusi, ning [9] oli spetsiifiliselt eelistanud LSTM-i arhitektuuri üle teiste rekurrentsete närvivõrkude.

Antud töö käigus olid kõik närvivõrgu arhitektuurid realiseeritud kasutades *Python*-i masinõppe teeki *Keras, TensorFlow backend*-iga. Arhitektuuride skeemid on loodud *Keras*-e poolt pakutud funktsiooniga *plot_model*. Kõiki töös loodud arhitektuure kasutades loodi neile vastavas peatükis leiduvas tabelis antud parameetritega kolm mudelit iga arhitektuuri kohta, et vähendada masinõppimise stohhastilisuse impakti töö tulemustele, ning käesolevas töös on raporteeritud vaid treenimise järgselt validatsiooniandmestiku peal kõrgeimat täpsust näidanud mudel. Graafikute koostamiseks on kasutatud teeki *Matplotlib*.

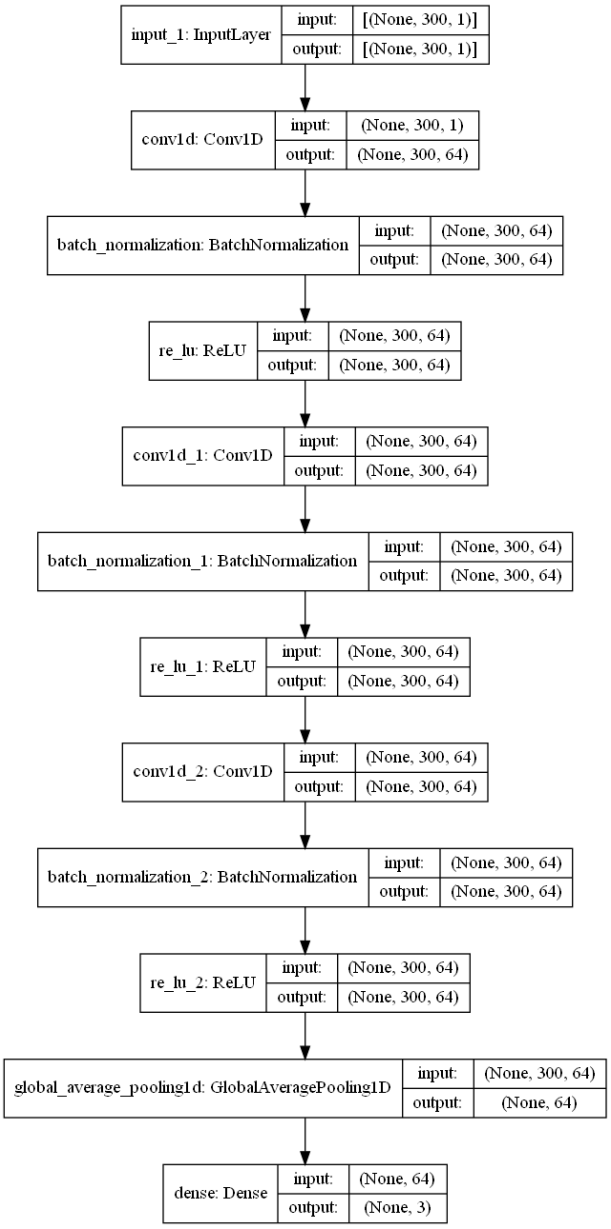
Käesolevas töös kasutatud masinnärvivõrkude erinevatest kihtidest annab hea ülevaate [11] ning *Keras* teegi dokumentatsioon. Järgnevalt on esitatud lühike ülevaade erinevatest kihtidest, mis leidub ka [11]-s: **Konvolutsiooniline kiht** võtab närvivõrku sisestatud andmed või mõne teise kihi poolt juba töödeldud andmed ning rakendab neile valiku filtreid et võimaldada andmetest selgemini näha relevantseid tunnuseid. ***Max pooling* kihti** kasutatakse tunnuste dimensionaalsuse vähendamiseks valides mingi osa tunnuste jaoks suurima väärtusega tunnus. ***Dense* kiht** koondab kõik tunnused eelmistest kihtidest kokku ning kaardistab nad, lubades teha lõplike otsuseid leitud andmete klassi kohta.

Lisaks kirjeldatud kihtidele on töös kasutatud ka ***LSTM*-kihti**, millest annab ülevaate [14] ning ***Batch Normalization* kihti**, millest annab ülevaate [15].

Käesolevas töös kasutatud programmikood kirjeldatud mudelite loomiseks ja treenimiseks on antud Lisas 4, Lisas 5 ja Lisas 6.

3.3.1 Konvolutsiooniline masinnärvivõrgustik (CNN)

Käesolevas töös kasutatud CNN-mudel koosneb kolmest sügavalt ühendatud närvivõrgukihist. Wang, Z. et al. [7] kasutas oma töös sarnast CNN-i variatsiooni, kus lõpliku *dense* närvivõrgukihi asemel on kasutatud 1x1 konvolutsiooniline kiht. Kuna *dense* närvivõrgukiht suudab paremini generaliseerida kui 1x1 konvolutsiooniline närvivõrgukiht [16], otsustati antud töös jääda *dense* närvivõrgukihi juurde. Mudeli loomisega aitas *Keras*-e dokumentatsioonis leiduva koodinäidis [17]. Mudeli treenimisel kasutati *Keras.callbacks* võimalusi platoodel õppimiskiiruse vähendamise jaoks ning varajase õppimise lõpetamise jaoks kui õppimiskadu vähenemist ei ole toimunud 50 iteratsiooni jooksul. Antud töö jooksul kasutatud CNN-i struktuur on antud joonisel 6. Antud närvivõrguarhitektuuril põhineva mudeli loomise ning treenimise jaoks kasutatud hüperparameetrid on esitatud tabelis 1. Parim antud parameetritel loodud mudel näitas validatsiooniandmestiku peal täpsust 0.8444 ning kadu 0.4649. Mudeli treenimine lõpetati 172-sel *epoch*-il. Treenimine võttis ligikaudu 30 minutit.



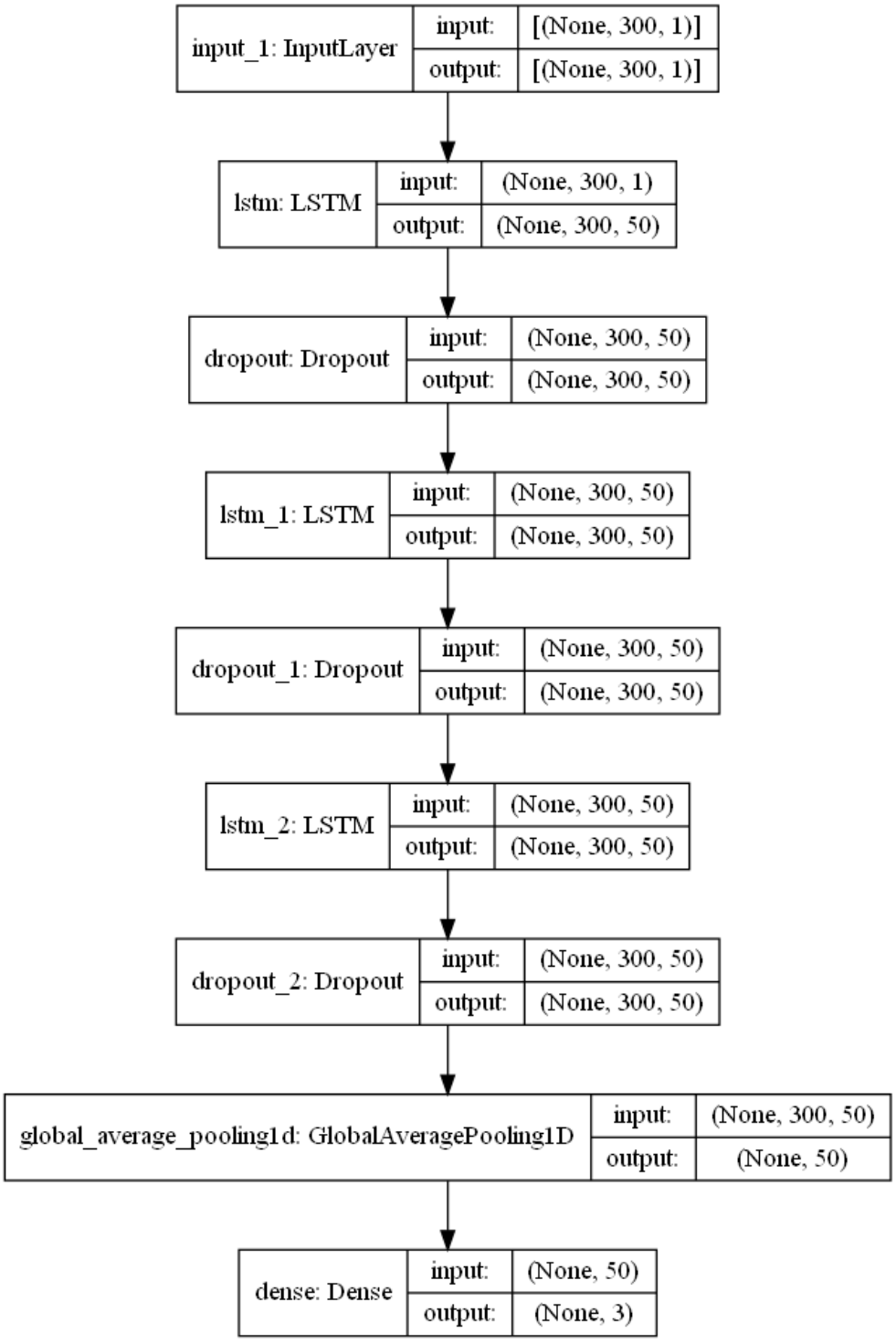
Joonis 6. Antud töös kasutatud CNN-i struktuur.

Tabel 1. Antud töös kasutatud CNN mudeli hüperparameetrid.

Hüperparameetri nimetus	**Hüperparameetri väärtus**
Batch size	16
Validation split	0.2
Max Epoch count	1000
Optimizer	*Adam*
Early stopping patience	50
Output layer activation	*Softmax*

3.3.2 Rekurrentne masinnärvivõrgustik (LSTM)

Käesolevas töös kasutatud LSTM-arhitektuur koosneb kolmest LSTM-kihist ning neile järgnevatest *Dropout* kihtidest. Mudeli treenimisel kasutati *Keras.callbacks* võimalusi platoodel õppimiskiiruse vähendamise jaoks ning varajase õppimise lõpetamise jaoks kui õppimiskadu vähenemist ei ole toimunud 50 iteratsiooni jooksul. Antud töö jooksul kasutatud LSTM-i struktuur on antud joonisel 7. Antud närvivõrguarhitektuuril põhineva mudeli loomise ning treenimise jaoks kasutatud hüperparameetrid on esitatud tabelis 2. Parim antud parameetritel loodud mudel näitas validatsiooniandmestiku peal täpsust 0.3333 ning kadu väärtuse üle ühe. Mudeli treenimine lõpetati 87ndal *epoch*-il. Treenimine võttis ligikaudu 20 minutit.



Joonis 7. Antud töös kasutatud LSTM-närvivõrgu struktuur.

Tabel 2. Antud töös kasutatud LSTM mudeli hüperparameetrid.

Hüperparameetri nimetus	**Hüperparameetri väärtus**
Batch size	16
Validation split	0.2
Max Epoch count	1000
Optimizer	*Adam*
Early stopping patience	50
Output layer activation	*Softmax*
Dropout Layer dropout rate	0.25

3.3.3 ResNet (Residual neural network)

Käesolevas töös kasutatud ResNet mudel koosneb kolmest *residual* närvivõrgukihist, mis on üksteisega ühenduses läbi nii tüüpiliste konvolutsiooniliste närvivõrgukihtide kui ka otseühenduste, mis lubab mudeli alamkihtidel teha tööd informatsiooniga mis tuleneb nii konsolutsiooniliste kihtide tulemustest kui vähem töödeldud tulemustest [7]. Mudeli implementeerimisel kasutati *Keras*-e teegi pakutud näidet [18], kuid muudeti näites loodud ResNet-i struktuuri olemaks vastuvõtlik ajaseeria informatsioonile, vähendades kovolutsiooniliste kihtide dimensioonaalsust. Mudeli treenimisel kasutati *Keras.callbacks* võimalusi platoodel õppimiskiiruse vähendamise jaoks ning varajase õppimise lõpetamise jaoks kui õppimiskadu vähenemist ei ole toimunud 50 iteratsiooni jooksul. Antud töö jooksul kasutatud ResNet-i struktuur on liiga pikk, et joonisel kuvada, ning on välja toodud Lisas 7. Antud närvivõrguarhitektuuril põhineva mudeli loomise ning treenimise jaoks kasutatud hüperparameetrid on esitatud tabelis 3. Parim antud parameetritel loodud mudel näitas validatsiooniandmestiku peal täpsust 0.9555 ning kadu väärtust 0.0736. Mudeli treenimine lõpetati 87ndal *epoch*-il. Treenimine võttis ligikaudu kaks tundi.

Tabel 3. Antud töös kasutatud ResNet mudeli hüperparameetrid.

Hüperparameetri nimetus	**Hüperparameetri väärtus**
Batch size	16
Validation split	0.2
Max Epoch count	1000
Optimizer	*Adam*
Early stopping patience	50
Output layer activation	*Softmax*

4 Tulemuste analüüs ning valideerimine

Loodud mudelite valideerimine toimub läbi *Keras*-e poolt pakutud võimaluste, täpsemalt läbi *evaluate* funktsiooni, mis kasutab töö jooksul loodud valideerimisandmestikku, et määrata loodud mudeli täpsust mudeli poolt varem nägemata andmetel. Lisaks täpsusele annab *evaluate* funktsioon mudeli loomisel valitud *loss*-funktsiooni väärtuse valideerimisandmestiku peal, et määrata mudeli kadu, mille täpsem kirjeldus on antud peatükis 4.1. Lisaks eelnevale joonistati mudeli treenimise ajal treenimise graafik, kus on näha mudeli treenimise ajal mudeli poolt näidatud täpsus ning mudeli hetkeline täpsus treeningandmestikust lahutatud andmete peal, mida sellel hetkel treenimiseks ei kasutatud ning mille mahu määras hüperparameeter *validation split*.

4.1 Kaofunktsioon

Mudeli *loss* ehk kadu on mõõdetav parameeter, mis näitab kui kaugel on algoritmi praegune väljund tema oodatavast väljundist [19]. Mudeli kadu väärtus sõltub mudeli koostamisel valitud kaofunktsioonist. Kuna parameeter mida mis töö käigus optimiseeriti on kategooriline täpsus, on valitud kaofunktsiooniks *categorical crossentropy*, mille valemiks on: $Loss = -\sum_{i=1}^{output\ size} yi \cdot log\ \hat{y}i$.

Antud kaofunktsioonis tähistab y*i* õiget ennustust ja ŷ*i* mudeli antud ennustust. Parim võimalik kaofunktsiooni väärtus on 0. Kuna käesolevas töös on ennustuse väärtused üksteist välistavad, on mälu kokkuhoiuks kasulik kasutata antud funktsiooni *sparse* ehk hõredat versiooni, mis on realiseeritud *Keras*-e teegis funktsiooninimega *Sparse Categorical Crossentropy* [20].

4.2 Tulemuste valideerimine

Tulemuste valideerimiseks kasutati peatükis 3.2.2 loodud validatsiooniandmestikku, mis sisaldab võrdse arvu kõigist kolmest võimalikult klassist, mis on treening- ja validatsiooniandmestike loomisel treeningandmetest kõrvaldatud. Kokku valideeritakse loodud mudelid vaid 45 erineva mõõtmistulemusega, mille väike arv oli tingitud klassi loom üldisest madalast esinemisest andmestikus. Võrdne arv igast klassist mõõtetulemusi oli vajalik, et valideerida mudeli võimet eristada klasside peidetuid tunnuseid teiste klasside tunnustest, mille jaoks ei tohiks mudel toetuda statistiliselt ühe või teise klassi eelistamisele üle vähemesinenud klassi.

Väikesest mõõtmistulemuste hulgast tulenevalt on oht, et validatsiooniandmestikus esinevad andmed ei esinda kõiki võimalikke peidetud tunnuseid, mistõttu antud validatsiooniandmestikul väga kõrgete tulemuste saavutamine võib viidata ülesobitumisele ning ei garanteeri, et mudel suudab saavutada sama häid tulemusi uuel informatsioonil mudeli tegeliku töö käigus. Sellele vaatamata on validatsiooniandmestikul oluline roll mudeli hindamisel koostöös teiste hindamismeetoditega.

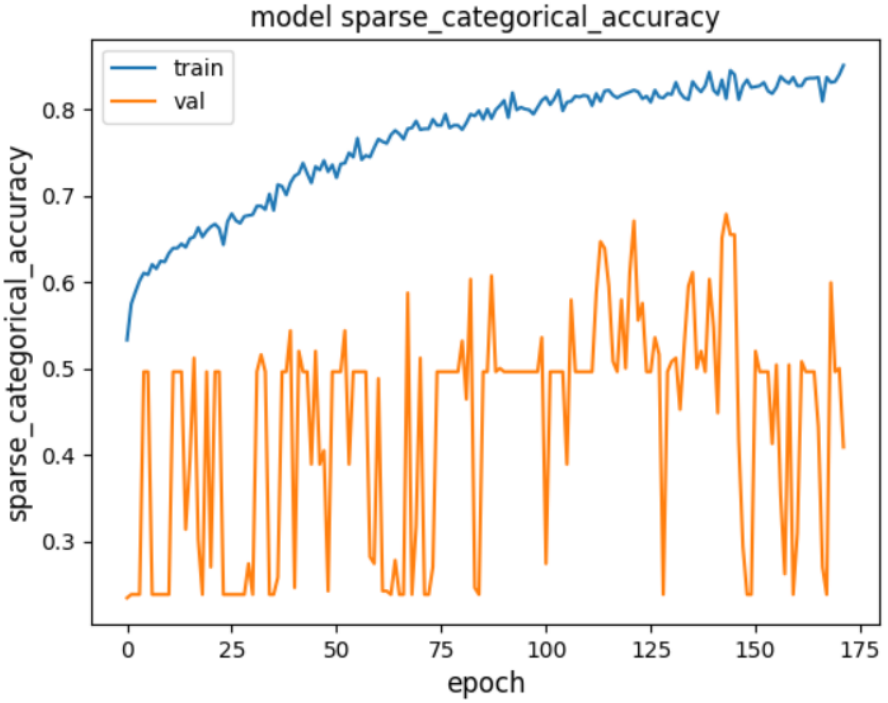
Loodud mudeli hindamisel peab lisaks treenitud mudeli lõppsooritusele validatsiooniandmestikul jälgima treeningprotsessi, et ära tunda võimalikud probleemid nagu ülesobitamine ja sellest tulenev halb üldistusvõime või algoritmi täielik suutmatus õppida, mille kõige selgem tunnus on pidevalt sama klassi ennustamine. Seetõttu oli oluline jälgida, et treeningu jooksul ei jääks täpsus ligikaudu 0.53, mis viitab kõige rohkem-esineva klassi ehk liiklusvahendi pidevale arvamisele. Antud probleemi teine ilmselge tunnus oleks validatsiooniandmestikul täpsuse 0.333 saavutamine, mis kolme klassi võrdsel esinemisel oleks vaid ühe klassi ennustamise täpsus. Antud täpsustest madalama tulemuse saavutamine viitaks hoopis valede tunnuste õppimisele, mis seaks kahtluse alla kas antud andmestikust on üldse võimalik otsituid peidetud tunnuseid leida või kas sellised peidetud tunnused üldse eksisteerivad.

4.3 Tulemuste analüüs

Treenimise käigus salvestati parim saavutatud versioon mudelist vastavalt minimaalsele saavutatud kaofunktsiooni väärtusele. Treeningu käigus salvestatakse mudeli täpsuse väärtustest graafik, milles sinine *train* joon tähistab treeningandmete peal näidatud täpsust ning oranž joon *val* tähistab iga mudeli treeningu alguses parameetri *validation split* määratud ning treeningandmetest kõrvale pandud osa peal demonstreeritud täpsust. Oluline märkus on see, et treeningandmetest kõrvale pandud andmed vahetusid iga uue mudeli loomisega, kuid spetsiifiliselt töö käigus loodud validatsiooniandmestik oli ainult lõpliku valideerimise jaoks kasutatud ning sisaldas käsitsi valitud mõõtetulemusi.

4.3.1 Konvolutsiooniline masinnärvivõrgustik (CNN)

Konvolutsiooniline närvivõrk saavutas parima tulemusena validatsiooniandmestiku peal täpsuse 0.8444, kaofunktsiooni väärtusega 0.4649.

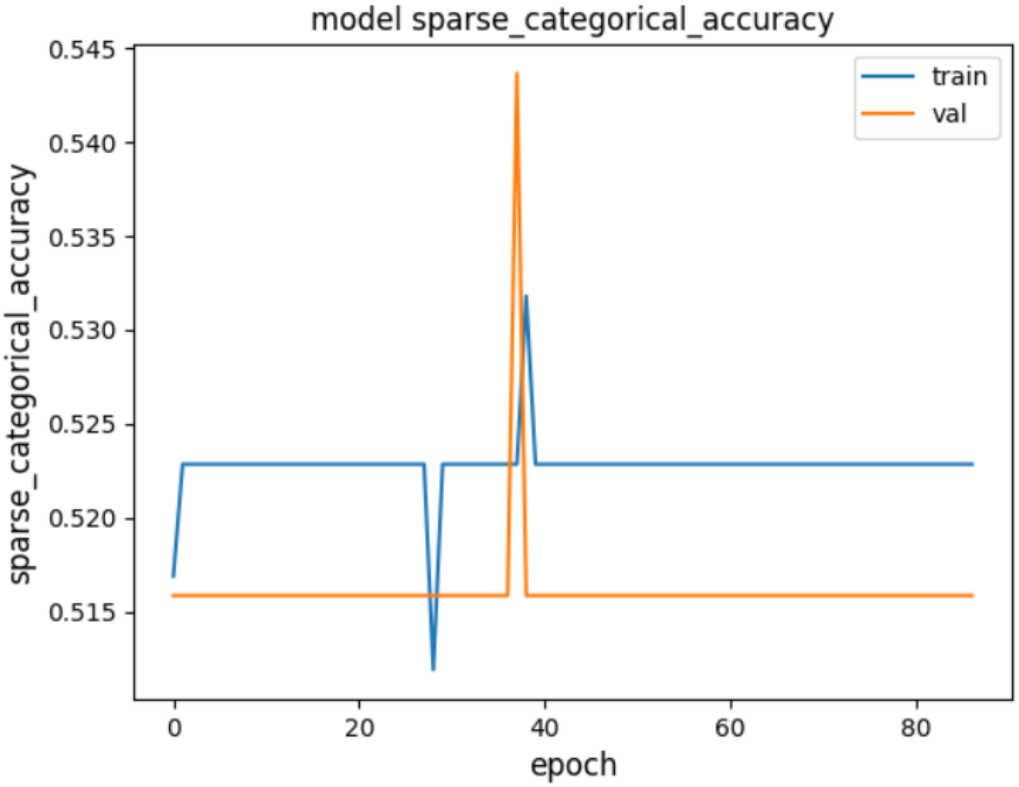


Joonis 8. CNN mudeli treenimise kulg.

Joonisel 9 on selgelt näha, kuidas treeningandmestiku täpsus kasvab, kuid treeningandmetest kõrvaldatud valiku täpsus ei suurene kordagi üle 0.7-e. Antud graafik viitab mingil määral ülesobitumisele, kuid validatsiooniandmestikul saavutatud kõrged tulemused viitavad sellele, et mudel on suuteline tuvastama olulisi tunnuseid antud andmestikest.

4.3.2 Rekurrentne masinnärvivõrgustik (LSTM)

Rekurrentne masinnärvivõrgustik saavutas parima tulemusena validatsiooniandmestiku peal täpsuse 0.3333, kaofunktsiooni väärtusega 1.1612

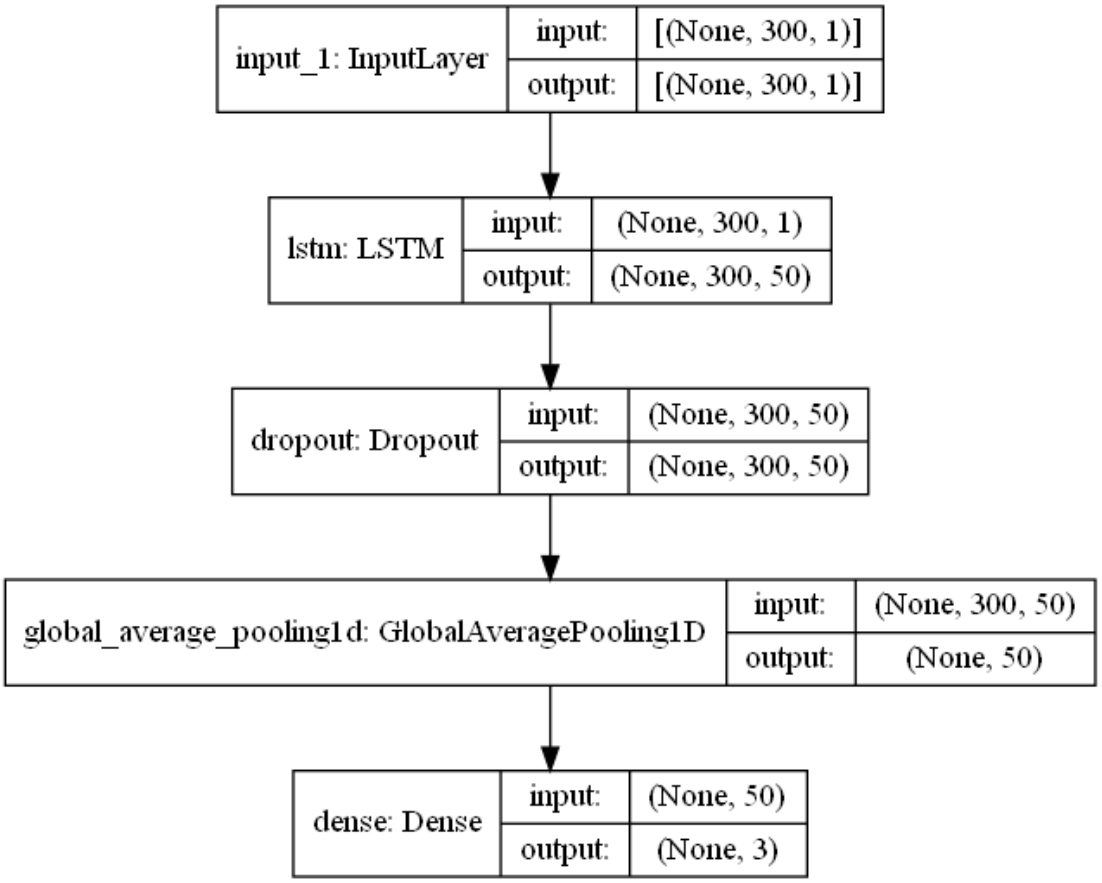


Joonis 9. LSTM mudeli treenimise kulg.

Joonisel 10 on näha ilmselge näide täielikust õppimisvõimetusest, peatükis 4.2 kirjeldatud täpsushinnangud mis viitavad õppimisvõime puudumisele on selgelt

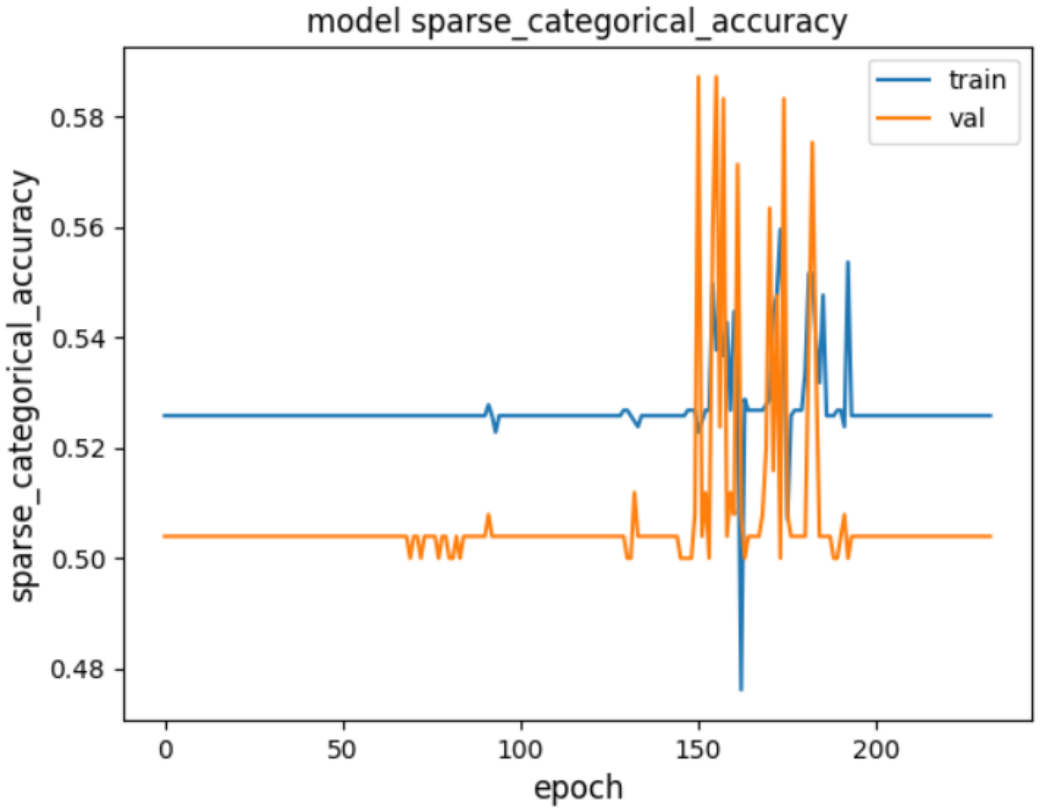
esindatud nii treeningugraafikus kui ka validatsiooniandmestiku tulemusel. Antud mudel ennustab vaid kõige levinumat klassi.

Kuna LSTM-i tulemused olid varasemast kirjandusest tulenenud ootuspärastest tulemustest täiesti erinevad, jooksutati katset lisaks veel ühekihilise LSTM-närvivõrguga.



Joonis 10. Ühekihilise LSTM-mudeli skeem.

Hüperparameetrid jäeti kolmekihilise LSTM-mudeliga võrreldes samaks, mis on kirjas tabelis 2.

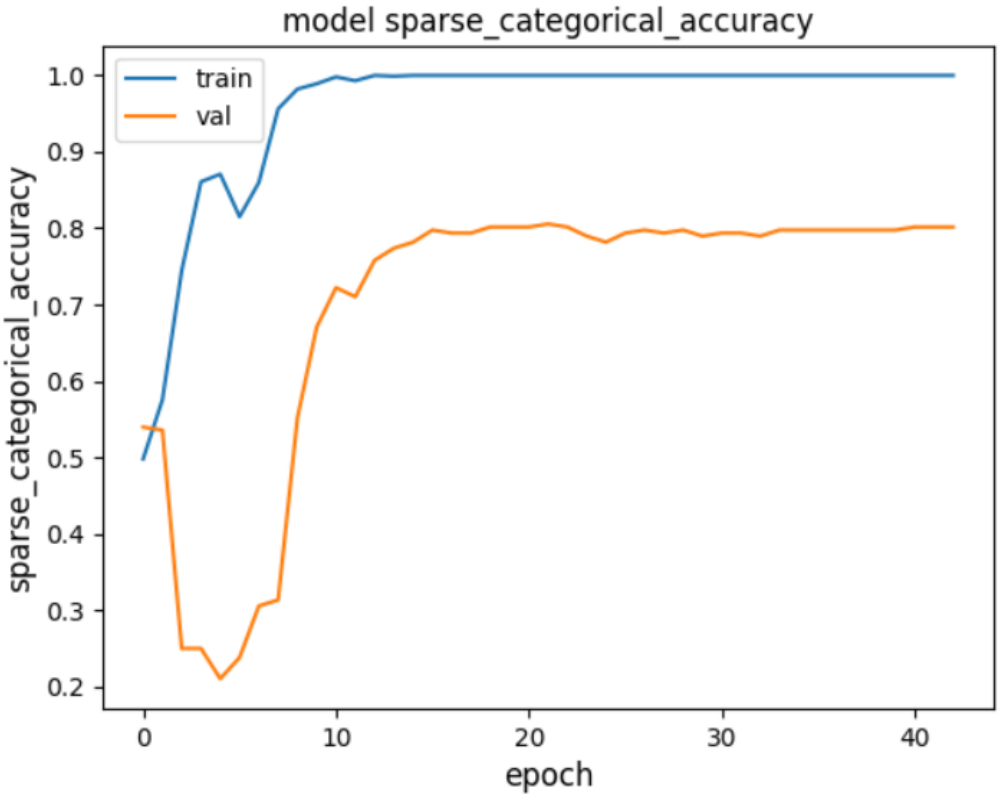


Joonis 11. Ühekihilise LSTM mudeli treenimise kulg.

Ühekihiline LSTM-närvivõrk saavutas validatsiooniandmestikul täpsuse 0.5333 ning kaofunktsiooni väärtuseks 1.0332. Ühekihilisel LSTM-närvivõrgul põhinev mudel suutis alles sajaviiekümnendaks epohhiks alustada treeningtulemusi mõjutavate muudatuste tegemist, mis suurendasid võrreldes mitmekihilise LSTM-iga validatsiooniandmestiku peal saavutatud täpsust 20 protsendipunkti võrra, ehk LSTM oli lõpuks suuteline andmestikust vähemalt pooltel juhtudel leidma olulised tunnused, kuigi tema tulemused on ikkagi sügavatest närvivõrkudest kehvemad.

4.3.3 ResNet

ResNet arhitektuuril põhinev närvivõrk saavutas validatsiooniandmestikul täpsuse 0.9555, kaofunktsiooni väärtusega 0.0736.



Joonis 12. ResNet-arhitektuuril põhineva mudeli treenimise kulg.

ResNet saavutas juba kümnendaks epohhiks treeningandmestiku ülesobitamise, saavutades ning treeningu käigus hoides 100% täpsust treeningandmete peal, ning treeningandmetest automaatselt loodud valimi peal saavutatud parim tulemus oli vaid 80%-line täpsus.Validatsiooniandmestiku peal saavutatud 95%-ne täpsus aga viitab sellele, et ResNet oli väga edukas andmetesse peidetud oluliste tunnuste õppimisega, kuigi õppis esmaselt valedest tunnustest mida on selgelt näha treeningu jooksul epohhide 3-7 tulemustest.

4.4 Tulemuste järeldused

Tabelis 4 on antud kokkuvõtvalt töö käigus loodud mudelite tulemused.

Tabel 4. Antud töös kasutatud mudelite tulemused.

Mudeli Nimetus	**Treening-andmestiku peal saavutatud parima täpsuse väärtus**	**Treening-andmestikust võetud suvalise valimi peal saavutatud parima täpsuse väärtus**	**Validatsiooni-andmestiku peal saavutatud kaofunktsiooni väärtus**	**Validatsiooni-andmestiku peal saavutatud täpsuse väärtus**
CNN	0.83	0.65	0.4649	0.8444
3-kihiline LSTM	0.53	0.54	1.1612	0.3333
1-kihiline LSTM	0.56	0.58	1.0332	0.5333
ResNet	1.0	0.80	0.0736	0.9555

Tabelist 4 on selgelt näha, et sügavad närvivõrgud nagu CNN ja ResNet suutsid loodud andmestikus peidetuid klasside tunnuseid kätte saada kõvasti edukamalt kui rekurrentsed närvivõrgud seda suutsid. See on osaliselt ootuspärane tulemus, kuna sügavate närvivõrkude võime leida sügavamale peidetuid seoseid on ajaseeria andmestike puhul teadaolev eelis rekkurentsete närvivõrkude üle, mis on paremad ajaseeria andmete ennustamises kui nende klassifitseerimises [21]. Loodud mudelite parim indikatsioon pärisandmetel toimimise tulemustest on suvaline, käsitsi mittekoostatud valim treeningandmetest, ning mudelite tulemused sellise valimi puhul on näidatud tabeli 4 kolmandas tulbas. Mudeli validatsiooniandmestike peal saavutatud tulemus on hea indikaator mudeli usaldusväärsusest iga individuaalse ennustuse jaoks, kuna validatsiooniandmestiku koostamisel ei arvestatud päriseluliste klasside jaotusega ning sooviti, et mudel oleks suuteline tegema iga mõõtetulemuse jaoks individuaalselt õige otsuse kasutades õpituid tunnuseid, toetamata statistilisele andmestikuanalüüsile. Vastavalt nendele parameetritele on kõige edukam mudel passiiv-infrapuna sensori väljundi analüüsimiseks ResNet-il põhinev mudel, ning antud andmeid on võimalik edukalt töödelda masinõppelistel meetoditel, et omandada töö käigus kasutatud detektoritele paigaldatud PIR-sensoritest lisaväärtust detektorite tööülesannete täitmisel.

4.5 Võimalikud arengud

Käesoleva töö skoopi ei langenud mõned potentsiaalselt saavutatud tulemusi parandavad meetodid, millest töö autor on teadlik ning millest on järgnevalt tehtud kiire ülevaade.

4.5.1 Hüperparameetrite optimisatsioon

Töö käigus kasutatud hüperparameetrid olid peamiselt otsustatud relevantses kirjanduses välja toodud näidete alusel või töö autori suva järgi. *Tensorflow* pakub hüperparameetrite optimiseerimise jaoks teeki *Keras_tuner* [22], mis aitab *Tensorflow*-ga implementeeritud süsteemidel valida parimad hüperparameetrid, mis toimivad ka *Keras*-ega. Optimaalsed, antud andmestiku jaoks spetsiifiliselt leitud hüperparameetrid on suutelised mudelite edukust andmete analüüsimisel tõstma, kui mudel on juba mingil määral võimeline andmeid analüüsima.

4.5.2 Uudne mudel

Töö autor ei väida, et läbiproovitud närvivõrgustruktuuridest ei leiduks paremat, töö tegemise käigus avastamata jäänud närvivõrguarhitektuuri, mis suudab läbiproovitud närvivõrkudest antud andmestikul veel parema tulemuse saavutada. Kuna iga loodud andmestik on erinev, kasvõi PIR-sensori mõõtmiskiirusest või mõõtmiskeskkonnast tulenevatel põhjustel, oli antud töö eesmärk uurida PIR-sensori realistlikust rakendamisest tulenevaid võimalusi üldlevinud meetodeid kasutades, mitte läbi proovida mudeli parameetrite optimiseerimisvabrikuid kuni on leitud selles ajahetkes kõige optimaalseim võimalik konfiguratsioon erinevatest närvivõrgukihtidest, mis spetsiifiliselt antud töö käigus kasutatud andmestikule rakendatult kõige tulemuslikum on.

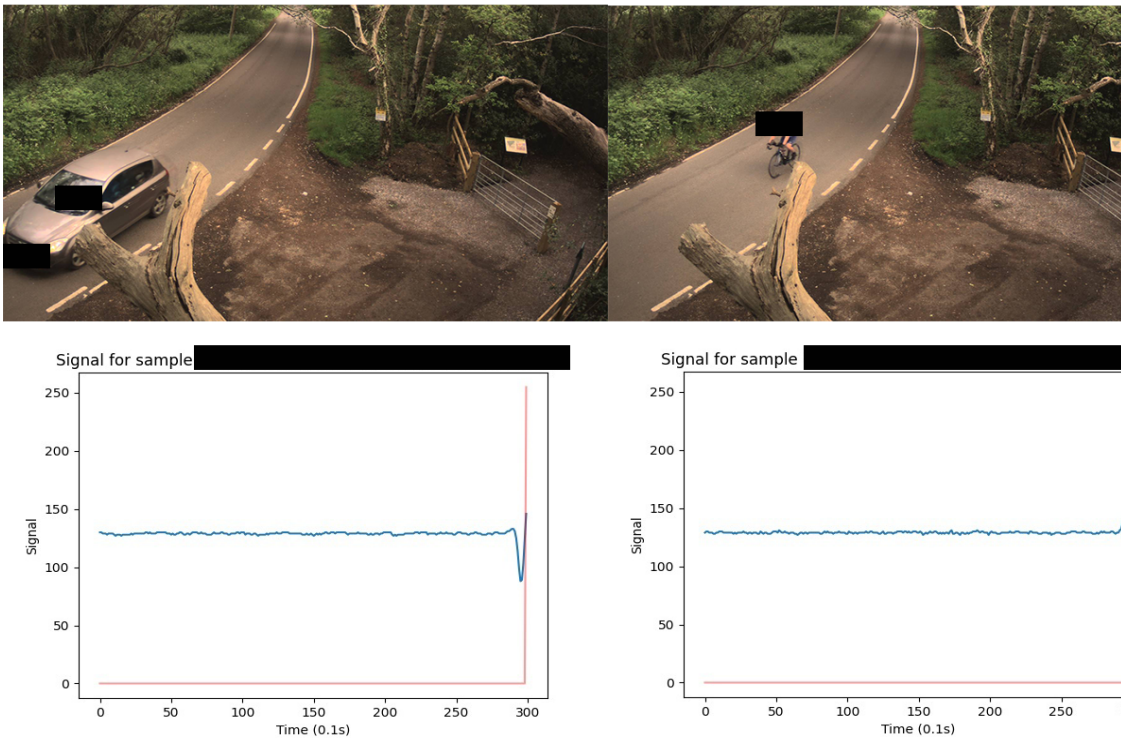
4.5.3 Masinõppe meetodite välised lahendused

Töö autor ei välista, et PIR-sensori konfiguratsiooni muutmisel masinõppesõbralikumaks, teistsuguse riistvaralise lahenduse kasutamine PIR-sensori

mõõdetud infrapunasignaali väärtuse muundamisel digitaalseks väärtuseks, PIR-sensori väljundile lisaks ka rohkemate olukorda kirjeldavate väärtuste mõõtmine ning nende masinõppelisele lahendusele ligipääsetavaks muutmine, rohkemate sensorite kasutamine või seni nimetamata võimalus oleks suuteline ületama praeguse tööga saavutatud tulemusi. Antud töö tulemused on saavutatud päriselt rakenduses olevate detektorseadmete regulaarse töö pealt, mis on peamiselt optimiseeritud autonoomsuse, kulutõhususe ja turvalisuse eesmärgil, mitte võimalikult detailsete andmete kogumise eesmärgil, mis antud töö eesmärkide täitmisele oleks kaasa aidanud.

4.5.4 Alternatiivsete lisaandmete kogumine

Infrapunasignaal võib endas tõenäoliselt sisaldada rohkem lisainformatsiooni kui signaali tekitaja klass, kuid antud töö selle informatsiooni tuvastamist ei käsitle, peale veendumise, et visuaalsest vaatlusest ei piisa selle informatsiooni tuvastamiseks. Joonisel 13 on näha samas suunas liikuvaid objekte ning nende poolt tekitatud signaale.



Joonis 13. Kahest samas suunas liikuvast kehast tehtud pildid ning nende signaalid.

Antud joonist vaadeldes on näha samas suunas liikunud objektide selgelt erinevat signaali, mis vajaks arvatavasti käesolevast tööst erinevat andmetöötlemise metoodikat, klassifitseerimise parameetritest kuni kasutatavate mudeliteni välja. Soovides PIR-sensori andmetest omandada teisi, alternatiivseid andmeid signaali tekitanud objektist või olukorrast, peaks käesolevas töös kirjeldatud metoodikat rakendama iga võimaliku uue lisaandme jaoks.

5 Kokkuvõte

Käesoleva töö käigus võeti aktiivses kasutuses olevate valveseadmestike koosseisu kuuluvatest PIR-sensoritest mõõtmistulemusi sensori vaatevälja sattunud infrapunaenergiat kiirgavatelt kehadelt. Antud mõõtmistulemusi kasutati unikaalse sildistatud andmestiku loomiseks, et seda seejärel kasutada masinõppeliste mudelite treenimise jaoks, mis suudaksid klassifitseerida infrapunasignaali kiirgaja tüüpi ainuüksi PIR-sensori väljundist. Antud töö raames käsitleti kolme klassi kiirgajaid: inimesed, loomad ning liiklusvahendid. Loodud andmestik sisaldas 1088 unikaalset mõõtmistulemust. Iga mõõtmistulemus koosnes kolmesajast individuaalsest infrapunakiirguse hulga mõõdust antud anduri vaateväljas, mis omandati kolmekümne sekundi jooksul.

Loodud andmestikul treenitud masinõppe mudelitest osutus parimaks klassifitseerijaks ResNet-i arhitektuuril põhinev mudel, mis suutis valideerimisandmestikul saavutada 96%-se täpsuse. Teised mudelid suutsid andmestikul saavutada samuti õppimisvõime, kuid nende täpsus jäi nii valideerimisandmestikul kui treenimisandmestikul madalamaks. Antud tulemused sellegipoolest kinnitasid tõsiasja, et PIR-sensorite andmetest on võimalik omandada lisaväärtust, muutmata igapäevases kasutuses olevate detektorite konfiguratsiooni.

Passiiv-infrapuna sensorite paremat mõistmist on edukalt edendatud: sensorite poolt kogutud informatsioon sisaldab endas peidetuid tunnuseid, mis lubavad eristada signaali tekitanud keha klassi infrapunasensorit kasutava seadme regulaarse töö käigus ning sensori väljundist on edukalt omandatud turvatöötajaid abistav lisainformatsioon.

Töö käigus loodud parima mudeli peal jätkatakse arendust, proovides katsetamisega suurendada mudeli tulemuslikkust töö käigus käsitlemata ja peatükis 4.5 mainitud andmetöötlus- ning masinõppemeetodeid rakendades. Töö käigus loodud andmestik võetakse põhjaks, et luua suurem andmestik, mis hõlmab endas mitme erineva päeva mõõtetulemusi ning lubab käesolevas töös loodud mudelite ning tulevaste mudelite treenimist tõhustada. Antud töös omandatud sügavam mõistmine

passiiv-infrapunasensoritest lubab järgnevaid katseid sarnaste seadmete poolt loodud mõõtetulemuste põhjal kergema vaevaga läbi viia. Peale loetletud etappide läbimist liidetakse signaali klassifikaator koostööd teinud firma poolt pakutud turvaliidesega, et tõhustada antud firma poolt pakutud turvalahenduste tööd, vähendada turvatöötajate tööga kaasnevaid riske ning pakkuda firma klientidele lisaväärtust nende soetatud seadmetest.

Kasutatud kirjandus

[1] Narayana, S., Prasad, R.V., Rao, V. S, et al. PIR sensors: characterization and novel localization technique, 2015, doi: 10.1145/2737095.2742561

[2] SAS Insights, *Machine Learning*, [Online]. Loetud aadressil: https://www.sas.com/en_us/insights/analytics/machine-learning.html Kasutatud: 10.04.2022

[3] J. S. Cook. Arrow Electronics, *The Right Tool for the Job: Active and Passive Infrared Sensors*, 2018. [Online]. Loetud aadressil: https://www.arrow.com/en/research-and-events/articles/understanding-active-and-passive-infrared-sensors Kasutatud: 10.04.2022

[4] A. J. Lester and C. L. Smith, "Analyses of performance of volumetric intrusion detection technologies," *Proceedings IEEE 33rd Annual 1999 International Carnahan Conference on Security Technology* (Cat. No.99CH36303), 1999, pp. 101-111, doi: 10.1109/CCST.1999.797902.

[5] Thakre, A., Kumar, A., et al. Pyroelectric Energy Conversion and Its Applications-Flexible Energy Harvesters and Sensors. *Sensors (Basel),* vol. 19,9 2170. May. 2019, doi:10.3390/s19092170

[6] Fresnel Technologies, *XX 1.2 Gl 12 VX,* 2009. [Online]. Loetud aadressil: https://www.fresneltech.com/hubfs/Spec%20Sheets/PIR%20Arrays/XX1.2GI12VX.pdf Kasutatud: 01.05.2022

[7] Z. Wang, W. Yan and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," *2017 International Joint Conference on Neural Networks (IJCNN),* 2017, pp. 1578-1585, doi: 10.1109/IJCNN.2017.7966039.

[8] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi , Chotirat Ann Ratanamahatana, Yanping Chen, Bing Hu, Nurjahan Begum, Anthony Bagnall , Abdullah Mueen, Gustavo Batista, & Hexagon-ML. *The UCR Time Series Classification Archive,* 2019 .[Online]. Loetud aadressil: https://www.cs.ucr.edu/~eamonn/time_series_data_2018/ Kasutatud: 03.05.2022

[9] Smirnov, D & Mephu N, Engelbert. “Time Series Classification with Recurrent Neural Networks”. *3nd ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data,* Dublin, Iirimaa. 2018. Loetud aadressil: https://www.researchgate.net/publication/348659317_Time_Series_Classification_with_Recurrent_Neural_Networks Kasutatud 08.05.2022

[10] Kim, D. H., et al. “Machine Learning-Based Object Detection System Using PIR Sensor”, *ACCSE 2018 : The Third International Conference on Advances in Computation, Communications and Services*. Department of Computer Science and Engineering, Pusan National University, Pusan, South Korea, pp 11-14. Loetud aadressil: https://www.thinkmind.org/download_full.php?instance=ACCSE+2018 Kasutatud: 08.05.2022

[11] K. A. Muthukumar, M. Bouazizi and T. Ohtsuki, "A Novel Hybrid Deep Learning Model for Activity Detection Using Wide-Angle Low-Resolution Infrared Array Sensor," *IEEE Access,* vol. 9, pp. 82563-82576, 2021, doi: 10.1109/ACCESS.2021.3084926.

[12] Python 3.10.4 documentation. “tkinter — Python interface to Tcl/Tk” [Online] Loetud aadressil: https://docs.python.org/3/library/tkinter.html Kasutatud: 08.05.2022

[13] Elite Data Science, *How to Handle Imbalanced Classes in Machine Learning,* 2019. [Online] Loetud aadressil: https://elitedatascience.com/imbalanced-classes Kasutatud: 01.05.2022

[14] Schmidhuber, J., “Long Short-Term Memory”, 2017. [Online] Loetud aadressil: https://people.idsia.ch/~juergen/rnn.html Kasutatud: 08.05.2022

[15] Ioffe, S. & Szegedy, C, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, 2015. [Online] Loetud aadressil: https://arxiv.org/abs/1502.03167 Kasutatud: 08.05.2022

[16] Towards Data Science, *Understanding and implementing a fully convolutional network (FCN)*, 2020. [Online] Loetud aadressil: https://towardsdatascience.com/implementing-a-fully-convolutional-network-fcn-in-tensorflow-2-3c46fb61de3b Kasutatud: 08.05.2022

[17] Fawaz, H. I., "Timeseries classification from scratch", *Keras.io*, 2020. [Online]. Loetud aadressil: https://keras.io/examples/timeseries/timeseries_classification_from_scratch/ Kasutatud: 08.05.2022

[18] Lazar, D., "Building a ResNet in Keras", *Towards Data Science*, 2020. [Online]. Loetud aadressil: https://towardsdatascience.com/building-a-resnet-in-keras-e8f1322a49ba Kasutatud: 08.05.2022

[19] François Chollet, Deep learning with Python (2017), Manning, peatükk 1 p.6

[20] knowledge Transfer, "How to choose cross-entropy loss function in Keras?", 2021. [Online]. Loetud aadressil: https://androidkt.com/choose-cross-entropy-loss-function-in-keras/ Kasutatud: 09.05.2022

[21] Dobilas, S., "RNN: Recurrent Neural Networks — How to Successfully Model Sequential Data in Python", *Towards Data Science*, 2020. [Online]. Loetud aadressil: https://towardsdatascience.com/rnn-recurrent-neural-networks-how-to-successfully-model-sequential-data-in-python-5a0b9e494f92 Kasutatud: 09.05.2022

[22] Tensorflow Core, "Introduction to the Keras Tuner", 2022. [Online]. Loetud aadressil: https://www.tensorflow.org/tutorials/keras/keras_tuner Kasutatud: 09.05.2022

Lisa 1 - Lihtlitsens lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Sander Kornet

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "[Lõputöö pealkiri]" , mille juhendaja on Priit Järv
   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.

Lisa 2 - Klassifitseerimisülesande hõlbustamiseks loodud tööriista kood

```
import sys
import time
import threading
import queue
import numpy as np
from PIL import Image, ImageTk
import tkinter as tk

interesting_samples_full_file = "interesting_samples_full_lines.out"
classified_samples_file = "classified.out"

show_sig = False
show_img = False

IMAGEDIR = "Sander3"

#MAKE SURE SAMPLES ARE GENERATED BEFORE THIS SCRIPT IS RAN, OTHERWISE WE
WILL CRASH.
SAMPLEDIR = "Sander3_signals"

maindict = {}

#Show some extra debug text
DEBUG = True

#Should the next event come up instantly after classifying the previous
one.
NEXT_POST_CLASSIFY = True

VERSION = "V1"

#####
#Tkinter base-base execution
#####
window = tk.Tk()
window.title(f"Classifying Tool {VERSION}")

####
#Event Node
####

class Node:
```

```
    def __init__(self, BDT, line):
        self.BDT = BDT
        self.nref = None
        self.pref = None
        self.text = line
        self.labelled = False

    def get_nref(self):
        return self.nref

    def get_pref(self):
        return self.pref

    def set_nref(self, nref):
        self.nref = nref

    def set_pref(self, pref):
        self.pref = pref

    def get_BDT(self):
        return self.BDT

    def get_text(self):
        return self.text

    def __repr__(self):
        return f"{self.text}"


###
#MAINDICT NODE POPULATING
###
CURRNODE = None
if DEBUG:
    print(CURRNODE)

firstnode = None

with open(interesting_samples_full_file, 'r') as file:
    lastnode = None
    for line in file:
        if line == "\n":
            break

        identificator = line[0:37]

        if CURRNODE == None:
```

```
            #Do not need to manage nref and pref for the first node
            #just create it and remember it as CURRNODE
            #so all subsequent node generation steps work

            FirstNode = Node(identificator, line)
            CURRNODE = FirstNode
            maindict[identificator] = FirstNode

            #Add global firstnode reference for future use
            firstnode = FirstNode

        else:
            #Make Node from ID
            newNode = Node(identificator, line)

            #Set last node next-reference to look at the current node
            lastnode = CURRNODE
            lastnode.set_nref(identificator)

            #Set new node prev to look at last node
            newNode.set_pref(lastnode.get_BDT())

            #Make sure new node is current.
            CURRNODE = newNode
            print(CURRNODE.get_BDT())

            #Make dictionary tag
            maindict[newNode.get_BDT()] = newNode

CURRNODE = firstnode

try:
    with open(classified_samples_file, 'r') as savefile:
        for line in savefile:
            line = line.strip()
            identificator = line[0:37]
            maindict[identificator].labelled = line.split(':')[1]
            print(maindict[identificator].labelled)
except FileNotFoundError:
    print("Savestate file not found, enjoy the start of a new labeling
project!")


def imagegrab():
    global CURRNODE
    try:
        current_Image =
Image.open(f"{IMAGEDIR}\{CURRNODE.get_BDT()}_.jpg")
```

```
        print(f"Set current images to {CURRNODE.get_BDT()}")
    except FileNotFoundError:
        print(f"\n\nImage for {CURRNODE.get_BDT()} isn't avaliable!")
        current_Image = Image.open(f"placeholder.png")
    finally:

        current_Sample =
Image.open(f"{SAMPLEDIR}\{CURRNODE.get_BDT()}.jpg")

        realimg = ImageTk.PhotoImage(current_Image.resize((640, 480),
Image.NEAREST))
        realsample = ImageTk.PhotoImage(current_Sample.resize((640,
480), Image.NEAREST))

        nodetext.set(f"{CURRNODE.get_text().rstrip()}")
        labeltext.set(f"LABEL: {CURRNODE.labelled}")

        BDT.delete(0, tk.END)
        BDT.insert(tk.END, f'{CURRNODE.get_BDT()}')

        imagelabel.configure(image=realimg)
        imagelabel.image = realimg
        samplelabel.configure(image=realsample)
        samplelabel.image = realsample
        print("Images handled")


#test-Grab IMG
try:
    current_Image = Image.open(f"{IMAGEDIR}\{CURRNODE.get_BDT()}_.jpg")
    print(f"Set current images to {CURRNODE.get_BDT()}")
except FileNotFoundError:
    print(f"\n\nImage for {CURRNODE.get_BDT()} isn't avaliable!")
    current_Image = Image.open(f"placeholder.png")
finally:
    current_Sample = Image.open(f"{SAMPLEDIR}\{CURRNODE.get_BDT()}.jpg")
    realimg = ImageTk.PhotoImage(current_Image.resize((640, 480),
Image.NEAREST))
    realsample = ImageTk.PhotoImage(current_Sample.resize((640, 480),
Image.NEAREST))
    print("Images handled")


####
#Traversal Functions
####
def get_next():
```

```
    global CURRNODE
    if DEBUG:
        print("next")
    if CURRNODE.get_nref() == None:
        print("We are at the last event of the day, next event is in
next day")
        imagegrab()
    else:
        identif = CURRNODE.get_nref()
        CURRNODE = maindict[identif]
        imagegrab()

def get_prev():
    global CURRNODE
    if DEBUG:
        print("back")
    if CURRNODE.get_pref() == None:
        print("We are at the first event of the day, previous event is
in previous day")
        imagegrab()
    else:
        identif = CURRNODE.get_pref()
        CURRNODE = maindict[identif]
        imagegrab()


def get_first():
    global CURRNODE

    CURRNODE = firstnode
    #ImageC.loadimage()


def find_BDT():
    global CURRNODE

    try:
        BDT = retrieve_input()
        CURRNODE = maindict[BDT]
        imagegrab()
    except KeyError:
        print("BDT not found.")

def find_first_unlabeled():
    global firstnode
    global CURRNODE
    checknode = firstnode
    while True:
```

```
        if checknode.labelled is False:
            CURRNODE = maindict[checknode.get_BDT()]
            imagegrab()
            break
        else:
            if checknode.get_nref() == None:
                print("All nodes are labelled!")
                CURRNODE = maindict[checknode.get_BDT()]
                imagegrab()
                break
            else:
                checknode = maindict[checknode.get_nref()]

def classify(classific):
    global CURRNODE

    with open(f"{classific}.out", 'a') as classed_file:
        if CURRNODE.labelled is False:
            classed_file.write(CURRNODE.get_text())
            CURRNODE.labelled = f"{classific}"
            print(f"Labelled img as {classific}")
            imagegrab()
            classed_file.close()
            savefile = open(classified_samples_file, 'a')
            savefile.write(f"{CURRNODE.get_BDT()}:{classific}\n")
            savefile.close()
        else:
            print("This node is already labelled.")
            classed_file.close()

    if NEXT_POST_CLASSIFY:
        get_next()

def retrieve_input():
    gotten_BDT = BDT_IN.get()
    return gotten_BDT

####
#Initialize Tkinter
####TODO: PACK -> GRID
nodetext = tk.StringVar()
infotext = tk.StringVar()
infotext.set("Copy-pastable BDT")
nodetext.set(f"{CURRNODE.get_text().rstrip()}")
labeltext = tk.StringVar()
labeltext.set(f"LABEL: {CURRNODE.labelled}")
```

```
imageframe = tk.Frame(window)
imageframe.grid(row=1, column=0, columnspan=4, rowspan=5)
imagelabel = tk.Label(imageframe, image=realimg)
imagelabel.pack()

sampleframe = tk.Frame(window)
sampleframe.grid(row=1, column=4, columnspan=4, rowspan=5)
samplelabel = tk.Label(sampleframe, image=realsample)
samplelabel.pack()

nodedataframe = tk.Frame(window)
nodedataframe.grid(row=0, column=0, columnspan=7, rowspan=1)
nodedatalabel = tk.Label(nodedataframe, textvariable=nodetext,
wraplength=1000)
nodedatalabel.pack()


BDTframe = tk.Frame(window)
BDTtext = tk.Label(BDTframe, textvariable=infotext)
BDTtext.pack(side=tk.TOP)
BDT = tk.Entry(BDTframe)
BDT.insert(tk.END, f'{CURRNODE.get_BDT()}')
BDT.pack()
button_unlab = tk.Button(BDTframe, text="First Unlabelled Event",
command = find_first_unlabeled)
button_unlab.pack(side=tk.BOTTOM)
BDTtext2 = tk.Label(BDTframe, textvariable=labeltext)
BDTtext2.pack(side=tk.BOTTOM)
BDTframe.grid(row=0, column=7, sticky=tk.N+tk.S+tk.E+tk.W)

#TODO-MAKE BUTTONS
button1 = tk.Button(window,text="<",command = get_prev)
button1.grid(row=6, column=0, sticky=tk.N+tk.S+tk.E+tk.W)
BDT_IN = tk.Entry(window)
BDT_IN.grid(row=6, column=1, sticky=tk.N+tk.S+tk.E+tk.W)
button2 = tk.Button(window,text="FIND BDT",command = find_BDT)
button2.grid(row=6, column=2, sticky=tk.N+tk.S+tk.E+tk.W) #->MAKE SURE
TO GIVE IT AN INPUT
button3 = tk.Button(window,text="ANIMAL",command = lambda:
classify("ANIMAL"))
button3.grid(row=6, column=3, sticky=tk.N+tk.S+tk.E+tk.W)
Button4 = tk.Button(window,text="VEHICLE",command = lambda:
classify("VEHICLE"))
button4.grid(row=6, column=4, sticky=tk.N+tk.S+tk.E+tk.W)
button5 = tk.Button(window,text="HUMAN",command = lambda:
classify("HUMAN"))
button5.grid(row=6, column=5, sticky=tk.N+tk.S+tk.E+tk.W)
```

```
button6 = tk.Button(window,text="UNKNOWN",command = lambda:
classify("UNKNOWN"))
button6.grid(row=6, column=6, sticky=tk.N+tk.S+tk.E+tk.W)
button7 = tk.Button(window,text=">",command = get_next)
button7.grid(row=6, column=7, sticky=tk.N+tk.S+tk.E+tk.W)

if __name__ == "__main__":
    window.mainloop()
```

Lisa 3 - Skaleerimisfunktsioon

```
#SCALE PARAMS (dependent on wished scale minimum and maximum):
scalemax = b = 1
scalemin =  a = -1

#INPUT PARAMS (dependent on possible input values):
maxx = 255
minx = 0

def normalize_value(x):
    """Input: one sample val
    output: normalized sample val"""
    normx = (b - a) *  ( (x - minx) / (maxx - minx) ) + a
    return normx
```

Lisa 4 - LSTM mudeli loomise ja treenimise kood

```
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

#MODEL TRAINING PARAMETERS
epochs = 1000
batch_size = 16

classlist = ["ANIMAL", "HUMAN", "VEHICLE"]
class_conversion = [0, 1, 2]

def readucr(filename):
    data = np.loadtxt(filename)
    y = data[:, 0]
    x = data[:, 1:]
    return x, y.astype(int)

x_train, y_train = readucr(f"Sander3LabelResults/train/result.txt")
x_test, y_test = readucr(f"Sander3LabelResults/test/result.txt")

#Normalize to a one-channel multivariate instead of an univariate one.
#No longer one channel per timeseries example, so our model could be fit
to
#multivariate time series if such and update would be neccessary.
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))
#Categorical crossentropy requires the class count beforehand
num_classes = len(np.unique(y_train))


#Shuffle the traiing set -> Technically done already when generating a
training
#set, but doing it aditionally for every execution can't hurt.
idx = np.random.permutation(len(x_train))
x_train = x_train[idx]
y_train = y_train[idx]
#THE MODEL - LSTM 1-layer Variation
def make_model(input_shape):
    input_layer = keras.layers.Input(input_shape)

    LSTMLayer1 = keras.layers.LSTM(units=50, return_sequences =
True)(input_layer)
    DropLayer1 = keras.layers.Dropout(0.25)(LSTMLayer1)
```

```
    gap = keras.layers.GlobalAveragePooling1D()(DropLayer1)

    output_layer = keras.layers.Dense(num_classes,
activation="softmax")(gap)

    return keras.models.Model(inputs=input_layer, outputs=output_layer)


model = make_model(input_shape=x_train.shape[1:])
keras.utils.plot_model(model, show_shapes=True)

callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50,
verbose=1),
]
model.compile(
    optimizer="adam",
    loss="SparseCategoricalCrossentropy",
    metrics=["sparse_categorical_accuracy"],
)
history = model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks,
    validation_split=0.2,
    verbose=1,
)


print("%%%%%%MODEL EVALUATION%%%%%%")
#EVALUATE MODEL
model = keras.models.load_model("best_model.h5")


test_loss, test_acc = model.evaluate(x_test, y_test)

print("Test accuracy", test_acc)
print("Test loss", test_loss)

#Plot the model's training and validation loss.
```

```
metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric])
plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large")
plt.xlabel("epoch", fontsize="large")
plt.legend(["train", "val"], loc="best")
plt.show()
plt.close()
```

Lisa 5 - CNN mudeli loomise ja treenimise kood

```
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt

#MODEL TRAINING PARAMETERS
epochs = 1000 #Assume callback early stopping way before reaching 1000
batch_size = 16
classlist = ["ANIMAL", "HUMAN", "VEHICLE"]
class_conversion = [0, 1, 2]

def readucr(filename):
    data = np.loadtxt(filename)
    y = data[:, 0]
    x = data[:, 1:]
    return x, y.astype(int)

x_train, y_train = readucr(f"Sander3LabelResults/train/result.txt")
x_test, y_test = readucr(f"Sander3LabelResults/test/result.txt")

#Normalize to a one-channel multivariate instead of an univariate one.
#No longer one channel per timeseries example, so our model could be fit
to
#multivariate time series if such an update would be neccessary.
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))
#Categorical crossentropy requires the class count beforehand
num_classes = len(np.unique(y_train))
#print(num_classes)


#Shuffle the traiing set -> Technically done already when generating a
training
#set, but doing it aditionally for every execution can't hurt.
idx = np.random.permutation(len(x_train))
x_train = x_train[idx]
y_train = y_train[idx]

#THE MODEL - FCN Variation
def make_model(input_shape):
    input_layer = keras.layers.Input(input_shape)

    conv1 = keras.layers.Conv1D(filters=64, kernel_size=3,
padding="same")(input_layer)
    conv1 = keras.layers.BatchNormalization()(conv1)
```

```
    conv1 = keras.layers.ReLU()(conv1)

    conv2 = keras.layers.Conv1D(filters=64, kernel_size=3,
padding="same")(conv1)
    conv2 = keras.layers.BatchNormalization()(conv2)
    conv2 = keras.layers.ReLU()(conv2)

    conv3 = keras.layers.Conv1D(filters=64, kernel_size=3,
padding="same")(conv2)
    conv3 = keras.layers.BatchNormalization()(conv3)
    conv3 = keras.layers.ReLU()(conv3)

    gap = keras.layers.GlobalAveragePooling1D()(conv3)

    output_layer = keras.layers.Dense(num_classes,
activation="softmax")(gap)

    return keras.models.Model(inputs=input_layer, outputs=output_layer)


model = make_model(input_shape=x_train.shape[1:])
keras.utils.plot_model(model, show_shapes=True)

callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50,
verbose=1),
]
model.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["sparse_categorical_accuracy"],
)
history = model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks,
    validation_split=0.2,
    verbose=1,
)
```

```
print("%%%%%%MODEL EVALUATION%%%%%%")
#EVALUATE MODEL
model = keras.models.load_model("best_model.h5")

test_loss, test_acc = model.evaluate(x_test, y_test)

print("Test accuracy", test_acc)
print("Test loss", test_loss)

#Plot the model's training and validation loss.
metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric])
plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large")
plt.xlabel("epoch", fontsize="large")
plt.legend(["train", "val"], loc="best")
plt.show()
plt.close()
```

Lisa 6 - ResNet mudeli loomise ja treenimise kood

```
from tensorflow import keras, Tensor
import numpy as np
import matplotlib.pyplot as plt

#MODEL TRAINING PARAMETERS
epochs = 1000
batch_size = 16

classlist = ["ANIMAL", "HUMAN", "VEHICLE"]
class_conversion = [0, 1, 2]

def readucr(filename):
    data = np.loadtxt(filename)
    y = data[:, 0]
    x = data[:, 1:]
    return x, y.astype(int)

x_train, y_train = readucr(f"Sander3LabelResults/train/result.txt")
x_test, y_test = readucr(f"Sander3LabelResults/test/result.txt")

#Normalize to a one-channel multivariate instead of an univariate one.
#No longer one channel per timeseries example, so our model could be fit
to
#multivariate time series if such and update would be neccessary.
x_train = x_train.reshape((x_train.shape[0], x_train.shape[1], 1))
x_test = x_test.reshape((x_test.shape[0], x_test.shape[1], 1))
#Categorical crossentropy requires the class count beforehand
num_classes = len(np.unique(y_train))


#Shuffle the traiing set -> Technically done already when generating a
training
#set, but doing it aditionally for every execution can't hurt.
idx = np.random.permutation(len(x_train))
x_train = x_train[idx]
y_train = y_train[idx]

#THE MODEL - ResNet Variation
def relu_bn(inputs: Tensor) -> Tensor:
    relu = keras.layers.ReLU()(inputs)
    bn = keras.layers.BatchNormalization()(relu)
    return bn
```

```
def residual_block(x: Tensor, downsample: bool, filters: int,
kernel_size: int = 3) -> Tensor:
    y = keras.layers.Conv1D(kernel_size=kernel_size,
               strides= (1 if not downsample else 2),
               filters=filters,
               padding="same")(x)
    y = relu_bn(y)
    y = keras.layers.Conv1D(kernel_size=kernel_size,
               strides=1,
               filters=filters,
               padding="same")(y)

    if downsample:
        x = keras.layers.Conv1D(kernel_size=1,
                   strides=2,
                   filters=filters,
                   padding="same")(x)
    out = keras.layers.Add()([x, y])
    out = relu_bn(out)
    return out

def make_model(input_shape):

    input_layer = keras.layers.Input(shape=(input_shape))
    num_filters = 64

    t = keras.layers.BatchNormalization()(input_layer)
    t = keras.layers.Conv1D(kernel_size=3,
               strides=1,
               filters=num_filters,
               padding="same")(t)
    t = relu_bn(t)

    block_counts = [2, 5, 5, 2]
    for i in range(len(block_counts)):
        num_blocks = block_counts[i]
        for j in range(num_blocks):
            t = residual_block(t, downsample=(j==0 and i!=0),
filters=num_filters)
        num_filters *= 2

    t = keras.layers.AveragePooling1D(4)(t)
    t = keras.layers.Flatten()(t)
    output_layer = keras.layers.Dense(num_classes,
activation='softmax')(t)

    return keras.models.Model(inputs=input_layer, outputs=output_layer)
```

```
model = make_model(input_shape=x_train.shape[1:])
keras.utils.plot_model(model, show_shapes=True)

callbacks = [
    keras.callbacks.ModelCheckpoint(
        "best_model.h5", save_best_only=True, monitor="val_loss"
    ),
    keras.callbacks.ReduceLROnPlateau(
        monitor="val_loss", factor=0.5, patience=20, min_lr=0.0001
    ),
    keras.callbacks.EarlyStopping(monitor="val_loss", patience=50,
verbose=1),
]
model.compile(
    optimizer="adam",
    loss="SparseCategoricalCrossentropy",
    metrics=["sparse_categorical_accuracy"],
)
history = model.fit(
    x_train,
    y_train,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks,
    validation_split=0.2,
    verbose=1,
)


print("%%%%%%MODEL EVALUATION%%%%%%")
#EVALUATE MODEL
model = keras.models.load_model("best_model.h5")


test_loss, test_acc = model.evaluate(x_test, y_test)

print("Test accuracy", test_acc)
print("Test loss", test_loss)

#Plot the model's training and validation loss.
metric = "sparse_categorical_accuracy"
plt.figure()
plt.plot(history.history[metric])
plt.plot(history.history["val_" + metric])
plt.title("model " + metric)
plt.ylabel(metric, fontsize="large")
plt.xlabel("epoch", fontsize="large")
```

```
plt.legend(["train", "val"], loc="best")
plt.show()
plt.close()
```

Lisa 7 - ResNet mudeli struktuur

Kuna loodud mudeli pilt on dimensioonidega 971x11693, on tema visuaalne esitamine ilma intensiivse pilditöötluseta esitatavas dokumendis võimatu. Lahendusena on töö autor laadinud loodud pildi ülesse saidile Imgur. Pilt on saadaval järgnevalt lingilt: https://imgur.com/a/dsQKoCe