TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kärte Pärend 164627

INIMESTE TUVASTAMINE JA LOKALISEERIMINE PUNKTIPILVEST VOXELNETI BAASIL

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2019

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: Kärte Pärend

26.05.2019

Annotatsioon

Käesoleva töö eesmärgiks on kontrollida, kas firmas Apple Inc. töötavate teadlaste Y. Zhou ja O. Tuzeli poolt väljapakutud tehisnärvivõrgu VoxelNet kahte mitteametlikku implementatsiooni on võimalik modifitseerida nii, et tõsta inimeste tuvastamise ja lokaliseerimise täpsust. Lisaks on püütud nende poolt kirjutatud artiklis esitatud tulemusi taastada.

Töö praktiliseks väärtuseks on saada teada, kas VoxelNeti mitteametlikke implementatsioone, mis on mõeldud autode lokaliseerimiseks, on võimalik praktikas rakendada piisava täpsusega, et punktipilvest leitaks üles kõik inimesed.

Töö käigus otsiti närvivõrkude treenimiseks sobiv keskkond ja loodi seal virtuaalkeskkonnad. Täiendati kahte VoxelNeti mitteametlikku implementatsiooni ja treeniti mudeleid. Närvivõrkude treenimise tulemusel leiti, et paremaid tulemusi saadi antud töö juhendaja Martin Rebase loodud VoxelNeti algoritmiga.

Töös tõestati, et VoxelNetiga on võimalik punktipilvest inimesi tuvastada.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 5 peatükki, 22 joonist, 5 tabelit.

Abstract

Using VoxelNet to Detect People from Point Cloud

The purpose of this thesis is to validate if an unofficial implementation of a neural network algorithm VoxelNet, that is created by scientists Y. Zhou and O. Tuzel at Apple Inc., can be modified to improve detecting people from pointcloud.

The practical purpose of this work is to find out, if two unofficial implementations of VoxelNet that have been made to detect cars, can be used to detect people from pointcloud with a good precision.

To get the results an appropriate environment was found and virtual environments were created. After that two VoxelNet implementations were modified and models were trained. After getting the training results it was found that the second algorithm, that was created by the supervisor of this thesis, gave better results than the first one.

It was proved that VoxelNet can be used to detect people from pointcloud.

The thesis is in Estonian and contains 41 pages of text, 5 chapters, 22 figures, 5 tables.

Lühendite ja mõistete sõnastik

Ahendamine	Tunnuskaartide (ehk maatriksite) dimensioonide vähendamine, et alles jääks kõige olulisem informatsioon (*pooling*)
Aktivatsioonifunktsioon	Funktsioon, mille sisendiks on eelmiste kihtide sisendite kaalutud keskmine ja mille väljund on järgmise kihi sisendiks. Enamasti kasutatakse ReLU, tanh või sigmoidfunktsiooni (*activation function*)
Andmete rikastamine	Olemasolevate andmete muutmine, et luua suuremat andmestikku (*data augmentation*)
Ankurkast	RPN algoritmi poolt objekti asukoha ennustamiseks loodav kast (a*nchor box*)
Ankurpunkt, ankur	RPN algoritmis kasutatava liikuva akna keskpunkt (*anchor point, anchor*)
Filter, kernel	Maatriks, mille abil on võimalik saada sisendpildist kõige olulisemad omadused
Gradientlaskumine, järsima laskumise meetod	Meetod, mille abil püütakse vähendada kaofunktsiooni väärtust (g*radient descent, steepest descent method*)
Ground truth	Õiged tulemused, mille alusel närvivõrku treenitakse
Hüperparameetrid	Parameetrid, mida muudetakse närvivõrgu treenimisel, et saada paremat mudelit (*hyperparameters*)
IoU	Kahe kujundi ühisosa (*intersection*) jagatud nende ühendiga (*union*). Kasutatakse ruumiliste ennustuste hindamiseks (*intersection over union*)
KITTI	Autonoomse juhtimise valdkonnas vabalt kasutatav etalonandmestik [1]
Kaofunktsioon	Funktsioon, mille väärtused näitavad, kui kaugel on mudeli tehtud ennustused õigetest tulemustest. Mida väiksemad on väärtused, seda parem on treenitud mudel (*loss function*)
LiDAR	Laserskaneerimisseade
Miniplokk	Kogu andmestikust juhuslikult võetud väikseim osa (*mini-batch*)
Mittenegatiivne lineaarfunktsioon, ReLU	Aktivatsioonifunktsioon, mille väärtus on null, kui sisend on negatiivne või null ja mille väärtus on võrdne sisendi väärtusega, kui sisend on positiivne (*Rectified Linear Unit*)
Märgend	Andmekirje klassikuuluvust või omadusi näitav informatsioon;

	oodatav tulemus (*label*)
Peidetud kihid	Kihid sisend- ja väljundkihtide vahel (*hidden layers*)
Piirikast	Kast, mis lisatakse objekti ümber selle mõttelise piiri tähistamiseks (*bounding box*)
Plokk	Andmestikust võetud osa, mida kasutatakse ühes iteratsioonis (*batch*)
Punktipilv	LiDARi skaneering. Ruumis laiali olevate punktide kogum
Pärilevi	Sõlmede väljundite arvutamine (*forward propagation*)
Regulariseerimine	Meetod ülesobituse vältimiseks
RPN	Algoritm, mida kasutatakse objekti asukoha ennustamiseks (*Region Proposal Network*)
Sisendkiht	Närvivõrgu esimene kiht. Kiht, millele antakse sisendandmed (*input layer*)
Stohhastiline gradientlaskumine, SGD	Gradientlaskumise algoritm, milles on miniploki suuruseks üks
Sügavõpe	Masinõpe, kus kasutatakse närvivõrke, milles on rohkem kui üks peidetud kiht (*deep learning*)
Tagasilevi	Gradientlaskumise rakendamine närvivõrgule (*backpropagation*)
Tensor	Ühte tüüpi andmeid sisaldav N-dimensiooniline andmestruktuur
Testandmestik	Mudeli tehtud ennustuste valideerimiseks kasutatav andmestik
Treeningandmestik	Treenitud mudeli valideerimiseks kasutatav andmestik
Tunnus	Sisendmuutuja, mida kasutatakse ennustuste tegemisel (*feature*)
Tunnusruum	„Kõigi võimalike karakteristikuvektorite hulk, mida saab kasutada pildi karakteristikute esitamiseks [2]“ (*feature space*)
Tunnuste kaart, tunnuskaart	Sisaldab sisendpildi kõige olulisemaid omadusi (*feature map*)
VFE kiht	VoxelNetis olev tunnuseid õppiv närvivõrk (*Voxel Feature Extraction*)
Voksel	Vähim kolmemõõtmeline element, mis väljendab kindlat väärtust kolmemõõtmelises ruumis olevas võrgus (*volume pixel, voxel*)
VoxelNet	Närvivõrk 3D punktipilvest objektide tuvastamiseks ja lokaliseerimiseks
Väljundkiht	Närvivõrgu viimane kiht. Väljastab tulemused (*output layer*)
Võrgusõlm	Närvivõrgu kihis olev sõlm, mis võtab tavaliselt mitu sisendit ja väljastab ühe väljundi (*node, neuron*)
Võrgusõlmede	Regulariseerimismeetod, mille puhul „lülitatakse“ osad

väljalülitamise meetod	võrgusõlmed välja (*dropout*)
Õppimismäär	Hüperparameeter, mida muudetakse, et saada treenides paremat mudelit. Väärtus, mis näitab, kui palju igas iteratsioonis võrgusõlmede kaalusid muudetakse (*learning rate*)
Ülesobitus, ülesobitamine	Sellise mudeli loomine, mis on treeningandmed ära õppinud nii, et ei suuda teha üldistusi uutel andmetel (*overfitting*)

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva töö eesmärgiks on uurida inimeste tuvastamist ja lokaliseerimist isesõitva auto trajektooril. Tänapäeval koguvad aina enam populaarsust isesõitvad sõidukid. Selleks, et sõiduk saaks ilma kõrvalise abita liikuda, on tal vaja erinevate sensorite abil ümbritsevast maailmast informatsiooni koguda, seda töödelda ning siis nende andmete põhjal otsuseid teha [3]. Töödeldud andmete põhjal õigete otsuste tegemiseks kasutatakse masinõpet, eriti tehisnärvivõrke. Viimasel ajal on populaarsust kogunud just sügavad närvivõrgud (*deep networks*), milles kasutatakse rohkem kui ühte peidetud kihti. Selliste närvivõrkude kasutamine on saanud võimalikuks, kuna tänu digitaliseerimisele on olemas suur kogus andmeid, arvutuskiirus on suurenenud ning on täiustunud ka närvivõrkude algoritmid [4].

Isesõitvad sõidukid saavad ümbritsevast maailmast informatsiooni sõidukile paigaldatud sensorite, näiteks kaamera, radari, infrapunasensorite või laserskaneerimisseadme LiDAR abil. Reaalelulistes süsteemides kasutatakse tihti kaamera ja LiDARi kombinatsiooni [5]. Kaameraga tehtud pildid annavad lisaks LiDARist saadud andmetele palju juurde, kuid alati ei saa loota sellele, et on võimalik mõlemat komponenti kasutada või et mõlemad komponendid töötavad. 2018. aasta kevadel toimunud Uberi isesõitva auto juhtumi puhul oleks kindlasti abi olnud sellest, kui sensorandmeid töötlev süsteem oleks suutnud jalgratast käekõrval lükkavat jalakäijat õigel ajal õigesti tuvastada ja tema asukohta määrata [3]. Suure tõenäosusega oli LiDAR ainukene sensor, mis oli suuteline liikuvat jalakäijat õigesti tuvastama ja lokaliseerima, kuna arvatavasti jäi jalgrattur kaamera pimedasse alasse (*blindspot*). Lisaks on LiDARi andmed pimedas usaldusväärsemad kui kaamera omad [6], [7].

Töö eesmärgiks on kontrollida, kas firmas Apple Inc. töötavate teadlaste Y. Zhou ja O. Tuzeli poolt väljapakutud tehisnärvivõrku VoxelNet [8] on võimalik modifitseerida nii, et tõsta jalakäijate tuvastamise ja lokaliseerimise täpsust. Lisaks on püütud nende poolt kirjutatud artiklis esitatud tabelis olevaid tulemusi taastada. Töö praktiliseks väärtuseks on saada teada, kas VoxelNeti mitteametlikku implementatsiooni, mis on mõeldud

autode lokaliseerimiseks, on võimalik praktikas rakendada piisava täpsusega, et kõik jalakäijad leitaks punktipilvest üles.

Töös on kolm peatükki, millest esimeses kirjeldatakse probleemi tausta ja töös kasutatavat sisendandmestikku, teises analüüsitakse VoxelNeti arhitektuuri ja selle mõistmiseks vajalikke tehisnärvivõrkude komponente ning kolmandas on toodud autori eksperimendid VoxelNetiga ja eksperimentidest saadud tulemused.

1.1 Töö käik

Töö eesmärgi täitmiseks kasutatakse Githubis olevat VoxelNeti ühte mitteametlikku[1] implementatsiooni ja antud töö juhendaja Martin Rebase poolt loodud VoxelNeti implementatsiooni. Mõlemad on mõeldud autode tuvastamiseks ja lokaliseerimiseks. Kõigepealt leitakse närvivõrkude käivitamiseks sobiv keskkond ja luuakse seal vajalikke pakette sisaldavad virtuaalkeskkonnad. Järgmisena testitakse mudeli töötamist autode tuvastamise puhul. Hiljem kohandatakse mudel jalakäijaid tuvastama.

Närvivõrgu treenimiseks kasutatakse KITTI[2] andmestikku [1], mis on autonoomse juhtimise valdkonnas kasutatav etalonandmestik.

Töös on püütud korrata Y. Zhou ja O. Tuzeli artiklis [8] leheküljel 4496 olevas 3D objektide tuvastuse ja lokaliseerimise tabelis olevaid tulemusi. Teised *open source* projektid on seda ainult autode peale replikeerinud. Lõputöös saadud programm kordab jalakäijate tuvastamist.

[1] https://github.com/skyhehe123/VoxelNet-pytorch

[2] https://github.com/bostondiditeam/kitti/blob/master/resources/devkit_object/readme.txt

2 Probleemi taust

Selles peatükis antakse ülevaade sellest, kuidas punktipilvedest ruumilisi objekte tuvastatakse. Nagu on öeldud Zhou ja Tuzeli artiklis, on punktipilvedel põhinev 3D objektide tuvastamine oluline komponent paljudes reaalse maailma rakendustes [8]. Seda kasutatakse näiteks isesõitvate autode, majapidamisrobotite, liitreaalsuse ja virtuaalreaalsuse puhul [8].

Võrreldes punktipilvedel põhinevat objektide tuvastust ja lokaliseerimist piltidel põhineva objektide tuvastuse ja lokaliseerimisega, on LiDARist saadud punktipilved hõredad, kuna punktipilvede tegemise meetod ei garanteeri, et igast ruumi osast salvestatakse ühepalju punkte [8], näiteks tühjas ruumis ei leidugi punkte, mida oleks võimalik salvestada. Lisaks võivad sensorid olla ebatäpsed, objektid võivad olla varjatud (*occlusion*) või LiDAR võib olla ebasobivalt asetatud [8]. Selliste probleemidega toimetulekuks on loodud mitmeid meetodeid punktipilvede käsitlemiseks. „Paljud meetodid projitseerivad punktipilved teise vaatesse ja rakendavad piltidel põhinevaid tunnuste eraldamise tehnikaid. Teised meetodid muudavad punktipilve 3D vokselitest koosnevaks võrgustikuks ja kodeerivad iga vokseli [8].“ Ühte sellist tehisnärvivõrgu arhitektuuri, mida kasutatakse 3D ruumi kaardistamiseks, nimetatakse VoxelNetiks. Selle meetodi kõige unikaalsem panus on tunnuseid õppiv VFE kiht (*Voxel Feature Encoding Layer*) koos täielikult ühendatud kihiga (*fully connected network*) [8].

2.1 Sisendandmestik

Punktipilvede saamiseks kasutatakse LiDARit, mis on laserskaneerimisseade, mille abil on näiteks helikopteritel või isesõitvatel autodel võimalik saada ümbritsevast kõrgekvaliteedilist kolmedimensioonilist informatsiooni [7]. Kuna tänapäeval on enamasti kasutusel pöörlevad LiDARid, siis on nende abil võimalik saada ümbritsevast ülevaade 360 kraadi ulatuses [7]. LiDAR väljastab punktipilvi (*point cloud*).

Kuna närvivõrgu treenimiseks vajamineva sisendandmestiku kogumine on ajamahukas ja väga kallis, on antud töös kasutatud KITTI andmestikku. Kuigi on olemas ka teisi avalikke andmestikke, mis pakuvad LiDARi andmeid ja milles on ka jalakäijaid, näiteks *Ford Campus Vision and Lidar Data Set* [9] või nuScenes andmestik [10], on otsustatud kasutada just KITTI andmestikku, kuna tegemist on ühe suurima ja enim kasutatud andmestikuga isesõitvate autode jaoks.

2.1.1 KITTI andmestik

The KITTI Vision Benchmark Suite on Karlsruhe Tehnoloogiaülikooli ja Chicagos asuva Toyota Tehnoloogia Instituudi poolt loodud andmestik [11]. Selle loomiseks on kasutatud autot, mille katusele on paigaldatud 2 kõrgresolutsioonilist stereo kaamerat (mustvalge ja värvikaamera), Velodyne laserskaneerija (enim kasutatud LiDAR) ja GPS süsteem [11]. Andmestik on kogutud keskmise suurusega linnas, maapiirkondades ja maanteedel ning on loodud keskendudes just objektide tuvastamiseks ja lokaliseerimiseks loodud tehisnägemise (*computer vision*) algoritmidele ja 3D orientatsiooni hinnangule [1]. KITTI andmestikus on märgistatud järgnev: autod, pakiautod, veoautod, jalakäijad, istuvad inimesed, jalgratturid, trammid, mitmesugune ja muu [1].

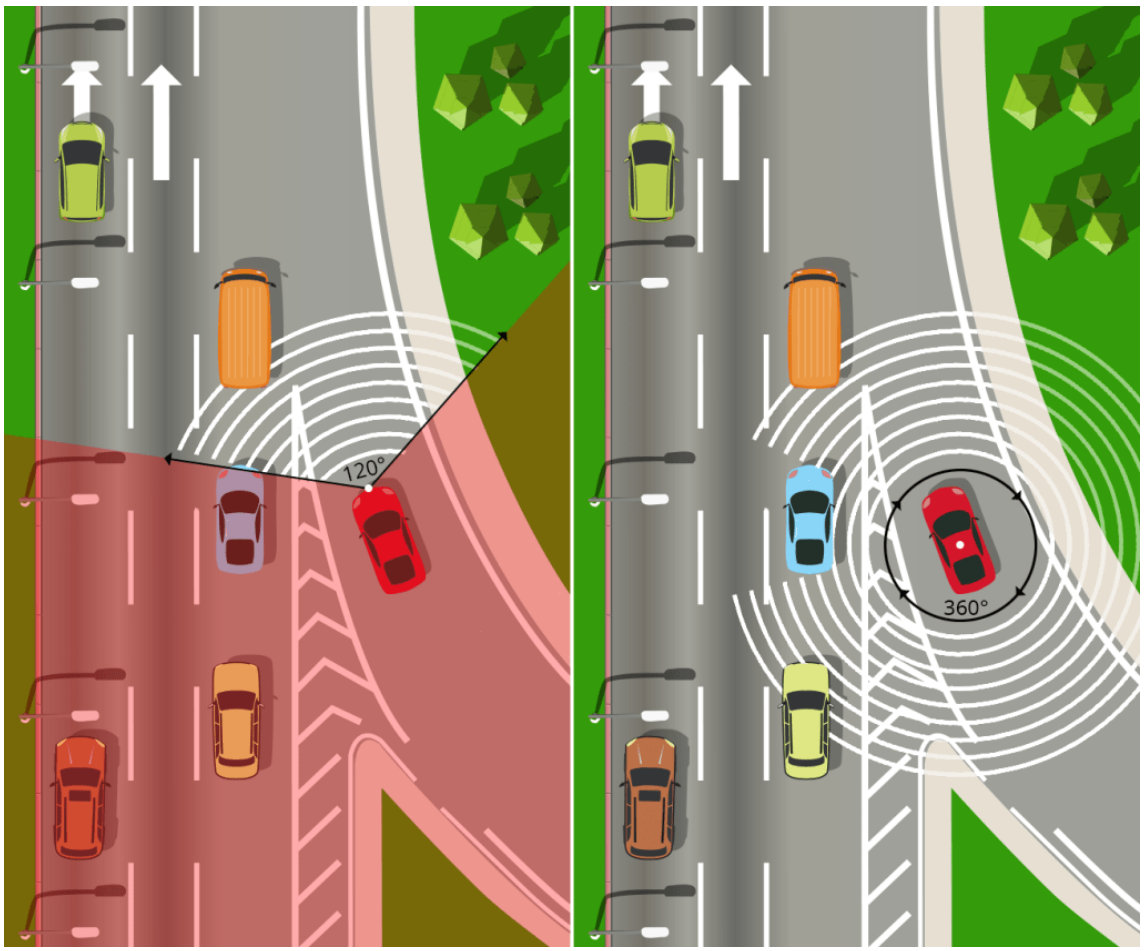
KITTI andmestik sisaldab 7 481 treeningpilti/punktipilve ja 7 518 testimise pilti/punktipilve [12]. Kokku on andmestikus 80 256 märgistatud objekti [12].

Käesolevas töös on kasutatud juhendaja poolt pakutud osa KITTI andmestikust, kus on treenimiseks 3712 töödeldud punktipilve ja nendele vastavad tekstifailid õigete tulemustega (*ground truth*), et närvivõrgu tulemusi valideerida. Sellest andmestikust on loodud väiksem andmestik, mis sisaldab 1210 punktipilve, milles kõikides on inimesi. Andmestiku loomiseks on kasutatud sama jaotust, mida kasutasid Chen *et al* ja mis on kujunenud *de facto* standardiks. Jaotuse kirjeldus on ka Internetist allalaetav[1]. Kasutatud jaotust on kirjeldatud artiklis „*Multi-View 3D Object Detection Network for Autonomous Driving*“[2].

[1] https://xiaozhichen.github.io/

[2] https://arxiv.org/abs/1611.07759

360 kraadi ulatuses tehtud punktipilvedel on üleliigne osa eemaldatud ehk alles on jäetud vaid osa, mis jääb ka sõiduki kaamera vaatevälja. Teoreetiliselt oleks võimalik närvivõrgule anda sisendiks ka 360 kraadi ulatuses tehtud punktipilvi, kui muuta närvivõrgu kihtide dimensioone, aga antud andmestiku puhul on takistuseks 360 kraadi ulatuses tehtud piltide puudumine. Seetõttu on jagatud punktipilv kolmeks osaks ja võetud vaid see osa, mis jääb ka kaamera vaatevälja (Joonis 1). Lisaks on punktipilved konverteeritud kaamera koordinaatsüsteemi.



Joonis 1. Vasakpoolsel pildil on valge viirutusega kujutatud kaamera vaatevälja jääv osa. Parempoolsel pildil on valge viirutusega kujutatud LiDARi poolt horisontaalselt nähtavat [7].

Antud töös kasutatud andmed on jagatud treeningandmeteks ja valideerimisandmeteks. Valideerimisandmete peal valideeritakse treenimistulemusi iga epohhi või teatud arvu epohhide järel. Antud juhul ei loodud testimisandmestikku, mille peal kontrollida, kas mudel on valmis.

Treeningandmete kaustas on punktipilved (velodyne kaust), tekstifailid, milles on märgitud, mis objekt on antud punktipilves või sellele vastaval pildil ning informatsioon objekti asukoha kohta (label_2 kaust) ning valideerimiseks kasutatavad pildid (image_2

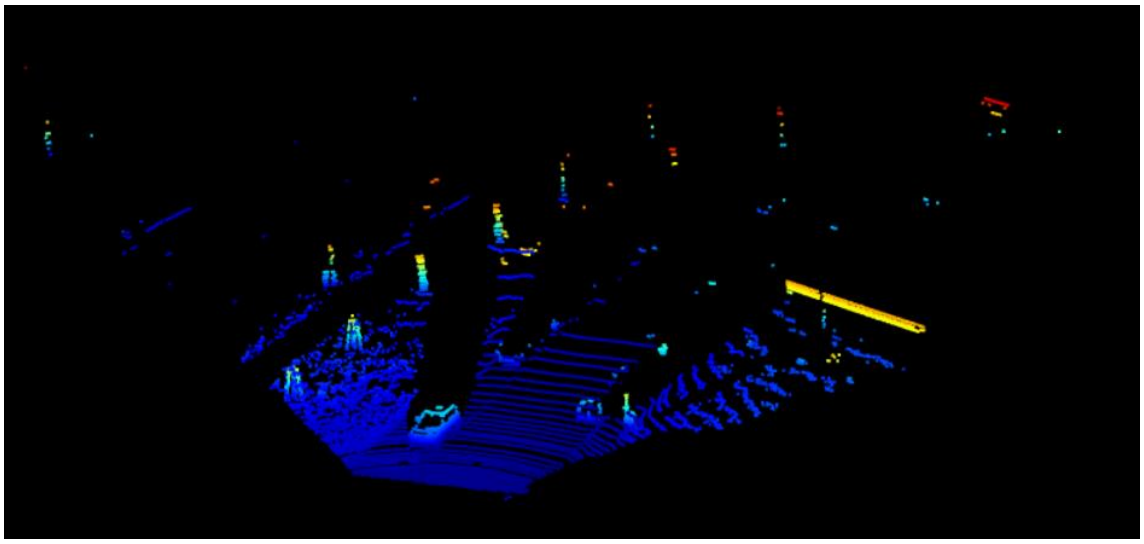
kaust). Kuna iga sensor (LiDAR või kumbki kaamera) on oma koordinaatsüsteemis (*reference frame*), siis võib vaja minna ka andmeid, antud juhul maatrikseid, et saaks mingit punkti ühest koordinaatsüsteemist teise teisendada. See on vajalik näiteks selleks, kui on vaja teada saada, kus mõne pildi peal asuv *ground truth* piirikast asuks 3D LiDARi skaneeringul ehk punktipilves. Sel põhjusel on kasutatud andmestikus kaasas ka andmed, mida ei ole veel ühest koordinaatsüsteemist teise teisendatud (calib kaust). Valideerimisandmete kaustas on vaid punktipilvede, tekstifailide ja piltide kaust. Lisaks on kaasas ka andmestiku jaotuse failid.

Märgenditega tekstifailides on punktipilvedes esinevate objektide kohta kirjas järgmised andmed: objekti tüüp, kas objekt väljub pildi piiridest, objekti nähtavus, objekti vaatlemise nurk, objekti ümbritseva 2D piirikasti koordinaadid (*top-left* ja *bottom-right*), 3D objekti mõõtmed (kõrgus, laius ja sügavus meetrites), 3D objekti asukoht X, Y, Z kaamera koordinaatides (antud meetrites), objekti pöördenurk ümber y-telje kaamera koordinaatides [1]. Tekstifailides on märgendid järgmiste objektide/piirkondade kohta: autod, pakiautod, veoautod, jalakäijad, istuvad inimesed, jalgratturid, trammid ja objektid, mis kuuluvad küll eelpoolnimetatud klassidesse, kuid on kaamera suhtes liiga kaugel, et neid tulemuste hindamisel arvesse võtta.

2.1.2 Punktipilved

„Punktipilv on ruumis laiali olevate punktide kogum, mis on saadud laserskaneerimise või fotogramm-meetria teel [13].“ Igal ruumis oleval punktil on X, Y ja Z koordinaadid. Ruumis olev punktide kogum kirjeldab tavaliselt mingi objekti kuju [13]. Järgneval lehel toodud joonisel (Joonis 2) on toodud üks antud töös kasutatavast andmestikust pärit punktipilv. Joonisel olev punktipilv on saadud kasutades selleks brauseris töötavat LiDARi punktipilvede visualiseerimise jaoks loodud tööriista[1], mis võtab sisendiks punktipilvi, mis on tekstifailides X Y Z formaadis või ASPRS LAS1.2 formaadis. KITTI andmestikus olevaid binaarformaadis punktipilvi saab teisendada X Y Z formaati kasutades selleks antud töös kasutatud koodiga kaasas olevat pcd_conv.py failis olevat koodi.

[1] http://lidarview.com/



Joonis 2. Autori visualiseering antud töös kasutatavas KITTI andmestikus olevast punktipilvest, millest on võetud ainult 120 kraadiline vaade.

Närvivõrkudes kasutatavad andmed võib klassifitseerida struktureeritud ja struktureerimata andmeteks [4]. Struktureeritud andmed on sellised, millel on kindel tähendus, näiteks hind või vanus. Struktureerimata andmed on näiteks pildid, heli, tekst [4]. Punktipilved kuuluvad struktureerimata andmete hulka.

3 Tehisnärvivõrgud ja VoxelNet

Antud peatükis räägitakse esmalt tehisnärvivõrkudest üldiselt ning kirjeldatakse närvivõrkude treenimist ja konvolutsioonilisi närvivõrke. Seejärel räägitakse töös kasutatud VoxelNeti arhitektuurist ja tööpõhimõttest.

3.1 Närvivõrgud

Tehisnärvivõrk on õppimisalgoritm, mille loomiseks on inspiratsiooni saadud bioloogilise aju töötamisest [4]. Närvivõrk koosneb kihtidest. Närvivõrgul on sisendkiht (*input layer*), enamasti ka üks või rohkem peidetud kihti (*hidden layers*) ja väljundkiht (*output layer*). Närvivõrgu kihtides olevaid elemente nimetatakse sõlmedeks või neuroniteks. Närvivõrke, milles ühe kihi väljund on teise kihi sisendiks, nimetatakse pärilevivõrkudeks (*feedforward networks*) [14]. Sellistes närvivõrkudes ei ole tsükleid ja info liigub alati ühes suunas – sisenditest väljundi poole. Närvivõrke, milles võib olla ka tsükleid, nimetatakse rekurrentseteks närvivõrkudeks [14]. Töös kasutatav VoxelNet kuulub pärilevivõrkude hulka.

Masinõppe algoritmide eeliseks on nende võime õppida ja parandada enda ennustusi [15]. Õppimine tähendab närvivõrkude puhul seda, et kaalud ja vabaliikmed valitakse lõpuks sellised, et närvivõrk suudab väljastada õige tulemuse või õigele küllaltki lähedase tulemuse [15]. Kaalud väljendavad seda, kui tugevad on neuronite vahelised ühendused. Vabaliikmed näitavad, kui kaugel on tehtud ennustused õigetest tulemustest. Närvivõrgu treenimise tulemusena saadakse õpitud mudel [16].

Närvivõrgu treenimise alguses toimub kaalude initsialiseerimine. Kaalud initsialiseeritakse enamasti juhuslikult, kuna kui kaalud initsialiseerida alguses nullidega, siis toimub närvivõrgu treenimine aeglasemalt kui kaalude juhusliku initsialiseerimise puhul [4].

Järgmisena toimub pärilevi (*forward propagation*), mille jooksul arvutatakse kaofunktsiooni (*loss function*) väärtus, ja tagasilevi (*backward propagation*), et uuendada kaalusid [4]. Tagasilevi puhul kasutatakse ka gradientlaskumise meetodit, et

vähendada kaofunktsiooni väärtust [16]. Pärast seda korratakse sama protsessi kuni kaofunktsiooni väärtus on piisavalt väike. Kaofunktsiooni kasutatakse, et arvutada ennustuse ja *ground truth* vahelist erinevust ehk väljundi viga [14]. Mida väiksem on kaofunktsiooni väärtus, seda parem, sest see tähendab, et ennustamisel tehtud viga on vähenenud.

Väljundi vea vähendamiseks tahetakse leida sellised kaalud/kordajad (*weights*) ja vabaliikmed (*bias*), et kaofunktsiooni väärtus oleks võimalikult väike [14]. Selleks kasutatakse gradientlaskumise ehk järsima laskumise meetodit (*gradient descent*), mis püüab väikeste sammudega liikuda "õigemate" väärtuste poole, kuni tulemus sellest paraneb [17]. Gradientlaskumise puhul töödeldakse kogu andmestik korraga [18]. Kuna kogu andmestiku korraga töötlemine ei ole efektiivne, siis kasutatakse optimeerimisalgoritmina gradientlaskumise asemel enamasti miniplokkide stohhastilist gradientlaskumist (*mini-batch stochastic gradient descent*), kus võetakse igas iteratsioonis juhuslikult väike kogus treeningandmeid (*mini-batch*) ja treenitakse selle peal. Seda korratakse kuni kõik treeningandmed on läbi käidud ehk on läbitud üks epohh (*epoch*) [14]. Vajadusel alustatakse pärast ühte epohhi uue treeningepohhiga [14]. Mõnikord kasutatakse ka väljendit iteratsioon, kuid see ei tähenda alati epohhi.

Närvivõrgu tulemuste parandamiseks kasutatakse ka teisi optimeerimisalgoritme. Üks enamkasutatavaid optimeerimisalgoritme on ka Adam algoritm, mis kuulub adaptiivse stohhastilise gradientlaskumise meetodite hulka ja mille puhul vähendatakse miinimumi poole liikumisel kiirust, et mitte miinimumist mööduda [18].

Närvivõrgu tulemuste parandamiseks on võimalik muuta ka hüperparameetrite (*hyperparameters*) väärtuseid. Muudetavad hüperparameetrid on enamasti treening epohhide (*epochs*) arv, miniploki suurus (*mini-batch size*), õpisamm (*learning rate*) [14] Lisaks on hüperparameetriteks ka peidetud kihtide arv, peidetud kihtides olevate võrgusõlmede arv ja aktiveerimisfunktsiooni (*activation function*) valik [4].

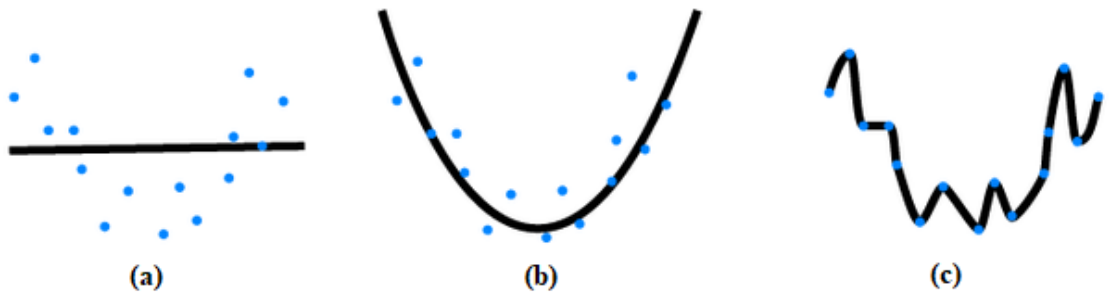
Närvivõrgu treenimise puhul on närvivõrgu töö hindamiseks mitmeid meetrikaid. Neid saab kategoriseerida rahuldavateks (*satisficing*) ja optimeerivateks (*optimizing*) meetrikateks [4]. Närvivõrgu headust saab hinnata vaadates kaofunktsiooni väärtuseid ja leides erinevad täpsused: õigsus (*accuracy*), täpsus (*precision*), saagis (*recall*) ja F1 skoor (*F1 score*) [4]. Veel on võimalik leida näiteks ka algoritmi töö aeg. Täpsus on

optimeeriv meetrika, sest soovitakse, et närvivõrk annaks võimalikult täpse tulemuse. Algoritmi töö aeg on rahuldav meetrika, sest tavaliselt lihtsalt soovitakse, et algoritmi töö jääks kindlatesse ajalistesse piiridesse [4].

3.2 Ülesobitus

Närvivõrku, millel on palju vabu parameetreid ehk hüperparameetreid (*hyperparameters*), saab kasutada paljude erinevate andmestikega, näiteks ühe ülesande ja andmestiku jaoks mõeldud närvivõrku on võimalik hüperparameetreid muutes kasutada ka mõne teise ülesande ja andmestiku puhul [14]. Närvivõrku treenides „võib juhtuda, et mudel töötab hästi olemasolevate andmetega, kuid ei suuda kohaneda uute andmetega [14].“ Seda nimetatakse ülesobituseks (*overfitting, overtraining*) ning see on närvivõrgu mudelite treenimise puhul suureks probleemiks. Ülesobituse vastupidiseks probleemiks on alasobitus, millega on tegemist siis, kui treenitud mudel on liiga lihtne [19] ehk mudel ei suuda olemasolevaid andmeid modelleerida ega kohaneda ka uute andmetega. Mõlemat olukorda on kirjeldatud joonisel (Joonis 3).

Et mudeleid efektiivselt treenida ja mitte üle treenida, on vaja teha kindlaks, millal ülesobitamine aset leiab. Seda on võimalik närvivõrgu treenimise ajal tuvastada näiteks graafikutelt. Kui on näha, et täpsus testandmetel enam ei parane, tuleks treenimine lõpetada [14].



Joonis 3. Närvivõrgu treenimisel tekkivad olukorrad: (a) alasobitus (*underfitting*), (b) sobiv olukord (*just right*), (c) ülesobitus (*overfitting*) [20].

Enamasti jagatakse närvivõrkude puhul kasutatav andmestik treeningandmeteks, valideerimisandmeteks ja testandmeteks. Sellisel juhul tuleks esiteks jälgida nii kaofunktsiooni väärtusi treeningandmetel kui ka protsentides olevat täpsust testandmetel, sest kaofunktsiooni graafik võib näidata, et mudel läheb iga uue epohhiga aina paremaks (kaofunktsiooni väärtus kahaneb), kuid testandmete puhul võib täpsuse

kasvamine mingil hetkel peatuda või jääda enam-vähem ühele tasemele ehk ei toimu enam õppimist [14].

Teiseks viitab ülesobitusele ka see, kui treeningandmete puhul kaofunktsiooni väärtus ainult kahaneb, kuid testandmete puhul kaofunktsiooni väärtus alguses väheneb ja mingi aja pärast hakkab suurenema [14].

Kolmandaks võib ülesobitusele viidata ka see, kui treeningandmete puhul jõuab mudeli täpsus 100 protsendini ehk mudel on korrektselt kõik pildid klassifitseerinud, kuid testandmete puhul näiteks ainult 80 protsendini [14]. See tähendab, et närvivõrk jätab meelde treeningandmed, kuid ei saa andmetest piisavalt hästi aru, et üldistada neid teadmisi testandmetele [14]. Üldiselt võib öelda, et kui täpsus testandmetel ei parane, siis võiks treenimise lõpetada [14].

Ülesobituse ennetamiseks on mitmeid meetodeid. Üks meetod ülesobituse ennetamiseks on kasutada varast lõpetamist (*early stopping*), mis tähendab seda, et mudeli treenimine lõpetatakse siis, kui mingi aja jooksul ennustustäpsus ei parane [14], [16]. Näiteks võib piirduda 10 epohhiga ja kui täpsus ei ole paremaks muutunud, võib lõpetada. Varase lõpetamise puhul arvutatakse ennustustäpsus iga epohhi lõpus enamasti valideerimisandmetel ja kui täpsus enam ei suurene, lõpetatakse treenimine [14]. Üldiselt kasutataksegi ülesobituse vältimiseks just valideerimisandmeid [14]. Närvivõrgu parameetrid valitakse valideerimisandmete põhjal, sest kui valida need testandmete põhjal, võib jälle toimuda ülesobitumine ehk närvivõrk töötab hästi ainult testandmetel ja ei oska üldistada end teistele andmetele [14]. Parameetrid valitakse valideerimisandmete põhjal ja testandmeid kasutatakse, et lõplikult hinnata närvivõrgu täpsust [14].

Ülesobituse vältimiseks on mitmeid viise. „Üks parimaid viise ülesobitumise vähendamiseks on suurendada treeningandmete hulka [14].“ See ei pruugi aga alati võimalik olla, sest andmete saamine võib olla keeruline ja kallis.

Veel on ülesobitumist võimalik vähendada regulariseerimise (*regularization*) abil. Üks levinumaid regulariseerimise meetodeid on L2 regulariseerimine (*weight decay, L2 regulariation*) [14], kus kaofunktsioonile liidetakse juurde regulariseerimise liige (*regularization term*). See liige sisaldab regulariseerimisparameetrit λ (*regularization parameter*). Regulariseerimise tulemus on see, et närvivõrk eelistab minimeerida

kaofunktsiooni (lambda on väike) või eelistab väikeseid kaalusid (lambda on suur) [14]. Pole kindlalt teada, miks regulariseerimine hästi töötab, kuid regulariseeritud närvivõrgud suudavad tavaliselt paremini üldistada kui mitteregulariseeritud närvivõrgud [14].

On olemas ka teisi regulariseerimistehnikaid, näiteks L1 regulariseerimine, väljajätumeetod (*dropout*) ja tehislikult treeningandmete suurendamine andmete rikastamise (*data augmentation*) teel [14]. L1 regulariseerimise puhul liidetakse samuti kaofunktsioonile regulariseerimise liige, kuid teistsugune L2 regulariseerimise puhul kasutatavast liikmest.

Väljajätumeetodi puhul aga „lülitatakse“ igas peidetud kihis pooled neuronid välja. Iga miniploki puhul „lülitatakse välja“ erinevad juhuslikult valitud neuronid [14]. Väljajätumeetodit on hea kasutada sügavate närvivõrkude treenimisel.

Andmete rikastamise puhul muudetakse närvivõrgule antavaid andmeid kasutades selleks erinevaid tehnikaid. Piltide puhul näiteks pööratakse, peegeldatakse või lõigatakse pilte. VoxelNetis on punktipilvede rikastamiseks kolm viisi: *ground truth* piirikastide lokaalne pööramine, *ground truth* piirikastide suuruse muutmine ja/või kogu punktipilve pööramine [8].

3.3 Hüperparameetrite valik

Sobivate hüperparameetrite valikuks on erinevaid viise. Õpisammu väärtused valitakse tavaliselt vahemikust 0.001–1.0. Kui õpisamm on liiga suur, võib algoritm miinimumist mööduda või võivad olla suured võnkumised miinimumi poole liikudes [14], [21]. Kui õpisamm on aga liiga väike, on õppimine liiga aeglane [21]. Tuleks leida selline õpisamm, mille puhul kaofunktsiooni väärtus kahaneb, mitte ei võngu ega ei kasva [14].

Õpisammu lävendi leidmiseks tuleks alguses valida õpisammule mingi väärtus, mille puhul kaofunktsiooni väärtus kahaneb algusest peale [14]. Pärast hakatakse õpisammu vähendama või suurendama ja jälgitakse kaofunktsiooni graafikut. Õpisammu võib suurendada niikaua, kuni hakkavad toimuma võnkumised või funktsiooni väärtus hakkab kasvama [14]. Lõpuks valitakse õpisammu lävendiks suurim väärtus, mille puhul kaofunktsiooni väärtus kahanes paari esimese epohhi jooksul [14]. Närvivõrgu treenimisel kasutatav õpisammu väärtus ei tohiks valitud lävendist suurem olla. Nielseni

kirjutatud raamatus [14] pakutakse, et õpisammu valikul võiks kasutada treeningandmeid, kuid teiste parameetrite valikul valideerimisandmeid.

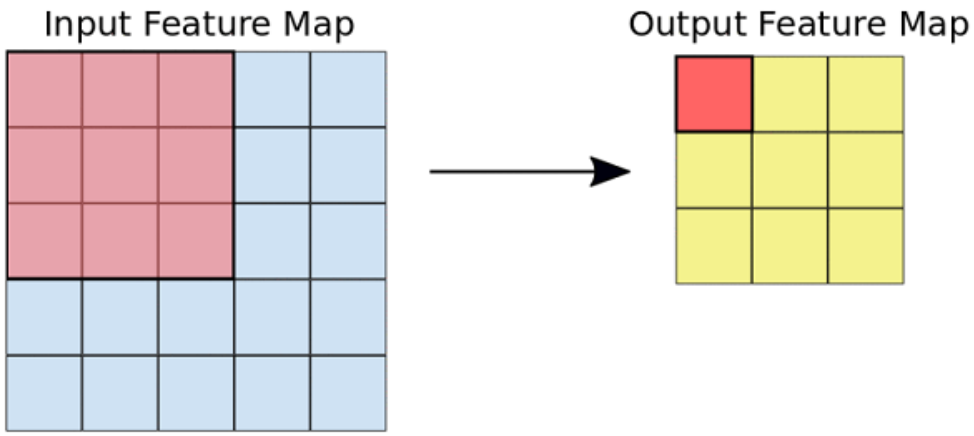
Miniploki suurus on enamasti 10–1000 näidist. Nielseni raamatus soovitatakse kõigepealt optimeerida teisi parameetreid ja alles siis proovida erinevaid väärtuseid miniploki jaoks [14].

Aktivatsioonifunktsioonide puhul kasutatakse sigmoid funktsiooni asemel enamasti ReLU või tanh aktivatsioonifunktsiooni [14].

Parimate hüperparameetrite valikuks sobib ka *grid search*, mille puhul otsitakse hüperparameetrite kombinatsioonide võrgustikust süstemaatiliselt sobivaimat kombinatsiooni [14].

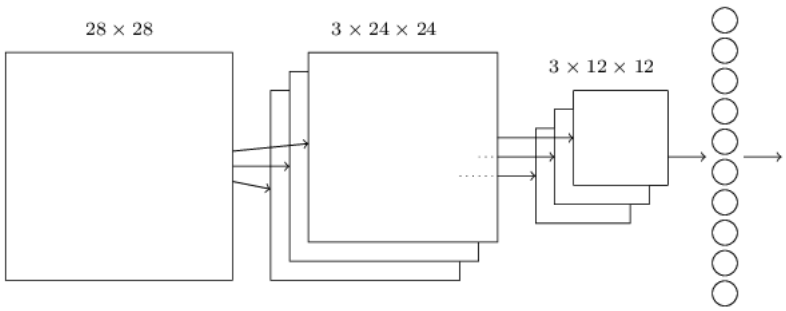
3.4 Konvolutsiooniline närvivõrk

Piltidega seotud ülesannete puhul kasutatakse tavaliselt konvolutsioonilisi närvivõrke (*convolutional networks*). Ka VoxelNeti üheks osaks on konvolutsioonilsed närvivõrgud. Erinevalt täielikult ühendatud närvivõrgust võtab konvolutsiooniline närvivõrk arvesse ka piltide ruumilist struktuuri [14]. Konvolutsiooniliste närvivõrkude puhul vähendatakse piltide dimensioone, et jätta alles vaid sisendpildi kõige olulisemad tunnused (*features*) [4]. Selleks rakendatakse sisendpildile järjest kernelit. Järgneval joonisel (Joonis 4) on toodud näide kerneli rakendamisest.



Joonis 4. Kerneli rakendamine [22].

Konvolutsiooniline närvivõrk koosneb konvolutsioonilistest kihtidest, ahendamise kihtidest (*pooling layers*) ja täielikult ühendatud kihist [4]. Sellise närvivõrgu puhul (Joonis 5) antakse sisendpilt konvolutsioonilistesse kihtidesse, kus rakendatakse sisendpildile korduvalt kernelit ja saadakse tunnuste kaart (*feature map*). Pärast konvolutsioonilisi kihte tulevad ahenduskihid (*pooling layers*), mis võtavad iga tunnuste kaardi ja vähendavad neis neuronite arvu [14]. Üks ahendamise levinumaid viise on kasutada meetodit *max-pooling*, mis tähendab, et tunnuste kaardi mingist regioonist väljastatakse vaid maksimaalne väärtus [14]. Näiteks kui konvolutsiooniliste kihtide väljundi suurus oli 24x24 ja iga kord rakendatakse *max-pooling*-t 2x2 suurusega alale, siis pärast ahendamist jääb alles 12x12 neuronit [14]. Igale tunnuste kaardile rakendatakse *max-pooling*-t eraldi. L2 ahendamist rakendades aga jäetakse alles alale jäävate aktivatsioonide ruutude summast ruutjuur [14].



Joonis 5. Konvolutsioonilise võrgu struktuur [14]. Lihtsustuse jaoks on näidatud vaid üks nool, selle asemel, et kõiki ühendusi näidata. Sisendiks on (a) 28x28 maatriks, järgnevad (b) konvolutsiooniline kiht, mille tulemuseks on 3x24x24 kiht, (c) *max-pooling*, mille tulemuseks on 3x12x12 peidetud tunnustega kiht ja (d) täielikult ühendatud kiht [14].

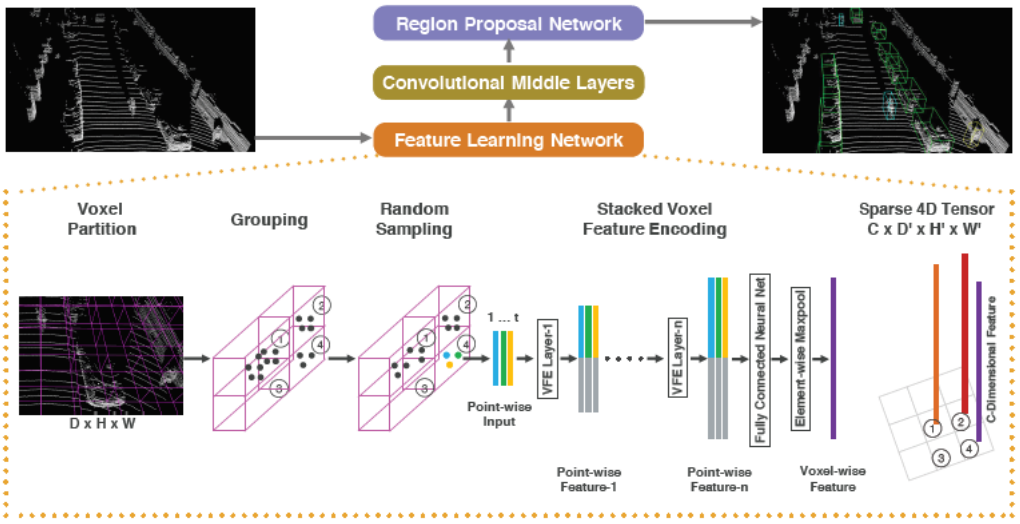
Konvolutsioonilise närvivõrgu lõpus on üks või rohkem täielikult ühendatud kihti, mille eesmärgiks on konvolutsioonide rakendamisel kogutud tunnuste põhjal läbi viia klassifitseerimine [22]. Tavaliselt kasutatakse viimases täielikult ühendatud kihis *softmax* aktivatsioonifunktsiooni, mis väljastab iga klassifitseeritava klassi kohta tõenäosuse [22]. Konvolutsioonilistes närvivõrkudes tuvastavad eespool olevad kihid lihtsamaid tunnuseid (näiteks servi), tänu sellele suudavad tagapool olevad kihid tuvastada keerulisemaid tunnuseid (näiteks näo osa) [4].

3.5 VoxelNet

VoxelNet on närvivõrk 3D punktipilvest objektide tuvastamiseks ja lokaliseerimiseks [8]. VoxelNet on *end-to-end* närvivõrk. *End-to-end deep learning* tähendab seda, et kasutatakse närvivõrku, mis koosneb erinevate ülesannete lahendamiseks mõeldud osadest, kuid närvivõrku vaadeldakse kui üht suurt närvivõrku, millele antakse sisend ja saadakse sealt ka väljund, ilma et peaks erinevaid osasid kasutama eraldi iga vajaliku komponendi arvutamiseks [23]. Seda ei ole võimalik kasutada kõikide probleemide puhul, kuna üldjuhul on selleks vaja suur hulka märgistatud andmeid [23]. Kuna sellisel süsteemil on vaja palju märgistatud andmeid, siis suudab ta tunnused ise ära õppida [23].

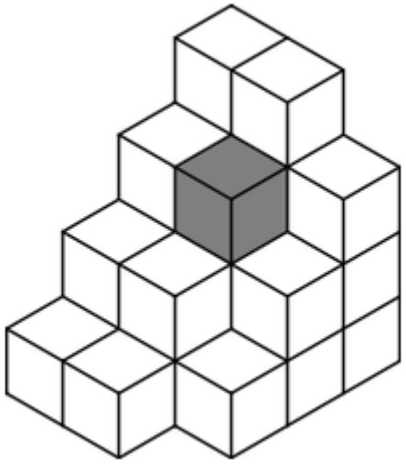
VoxelNeti miinuseks on aga see, et ei võeta arvesse objektide piire, kuna nende teadasaamiseks on vaja 3D ruumist (punktipilvest) eelnevaid teadmisi, mida VoxelNetil ei ole [24]. Kuigi VoxelNetis jagatakse 3D punktipilv vokseliteks, mis katavad terve ruumi, siis saadud vokselite suurused ja asukoht vokselite võrgus ei arvesta leitava objekti piiridega [24].

VoxelNet koosneb kolmest osast: tunnuseid õppiv närvivõrk (*Feature Learning Network*), konvolutsioonilised vahekihid (*Convolutional Middle Layers*) ja objekti asukohta ennustav RPN närvivõrk (*Region Proposal Network*). Teadlaste Y. Zhou ja O. Tuzeli poolt kirjutatud artiklis [8] on öeldud, et tunnuseid õppiv närvivõrk jagab punktipilve võrdsete suurustega 3D vokseliteks ja esitab SVFE (*Stacked Voxel Feature Encoding*) kihtide ehk tunnuseid õppivate kihtide abil iga vokseli sees olevat punktigruppi ühtse tunnusena [8]. VoxelNeti arhitektuuri kirjeldava joonise (Joonis 6) alumises osas on kirjeldatud üldiselt tunnuseid õppiva kihi osasid. Järgmisena tulevad konvolutsioonilised vahekihid ja RPN kiht, mis tuvastab ja lokaliseerib objekte [8].



Joonis 6. Zhou ja Tuzeli artiklis kirjeldatud VoxelNeti arhitektuur [8].

VoxelNeti puhul kasutusel olev mõiste voksel (*volume pixel*, *voxel*) tähendab vähimat kolmemõõtmelist elementi, mis väljendab kindlat väärtust kolmemõõtmelises ruumis olevas võrgus (Joonis 7) [25]. Erinevalt pikslitest, mis paiknevad 2D ruumis, ei sisalda vokselid endas informatsiooni enda asukoha kohta 3D ruumis [25]. Vokseleid võib kirjeldada kui kuupe, millest saab ehitada suuremaid objekte. Vokselid hoiavad endas informatsiooni enda suhtelise asukoha kohta teiste vokselite suhtes ja neid vaadeldakse kolmemõõtmelises ruumis kui punkte [25]. Vokselid võivad endas sisaldada ühte või mitut skalaarset väärtust, näiteks väärtust tiheduse, läbipaistvuse või värvi kohta [25].



Joonis 7. Näide vokselitest koosnevast võrgustikust, kus halliga on värvitud üks voksel [26].

Zhou ja Tuzeli artiklis on mainitud, et „KITTI andmestikul tehtud katsed on näidanud, et VoxelNet suudab andmestikust paremini autosid tuvastada ja lokaliseerida kui teised

LiDARil põhinevad 3D tuvastusmeetodid [8].“ Lisaks väidetakse artiklis, et VoxelNet suudab ainult LiDARi andmeid kasutades lisaks autodele ka jalakäijaid ja jalgrattureid tuvastada ja lokaliseerida [8].

Nagu on näha KITTI andmestiku kodulehelt [12], on VoxelNeti erinevaid teostusi kasutatud nii autode, inimeste kui ka jalgratturite tuvastamiseks ja lokaliseerimiseks. Kodulehel on toodud viited vaid nendele implementatsioonidele, kus autode puhul on saadud piirikastide kattuvuseks vähemalt 70% ning jalakäijate ja jalgratturite puhul vähemalt 50% [12]. 2017. aasta seisuga oli autode tuvastamisel üks VoxelNeti mitteametlik implementatsioon tabelis 64. kohal ning üks A-VoxelNetiga autode tuvastamine 24. kohal [12]. Jalakäijate tuvastamisel on A-VoxelNeti erinevad implementatsioonid aga 1. kohal ja jalgratturite tuvastamisel 9. kohal [12].

VoxelNet aga ei ole parim meetod 3D objektide tuvastamiseks ja lokaliseerimiseks punktipilvedest, sest paremaid tulemusi on saadud Frustum ConvNetiga [24], kus on kasutatud fokaalset kadu (*focal loss*). Fokaalset kadu kasutatakse tavaliselt selleks, et klasside arvu tasakaalutust lahendada, näiteks sellisel juhul, kui positiivseid näiteid on palju vähem kui negatiivseid [27].

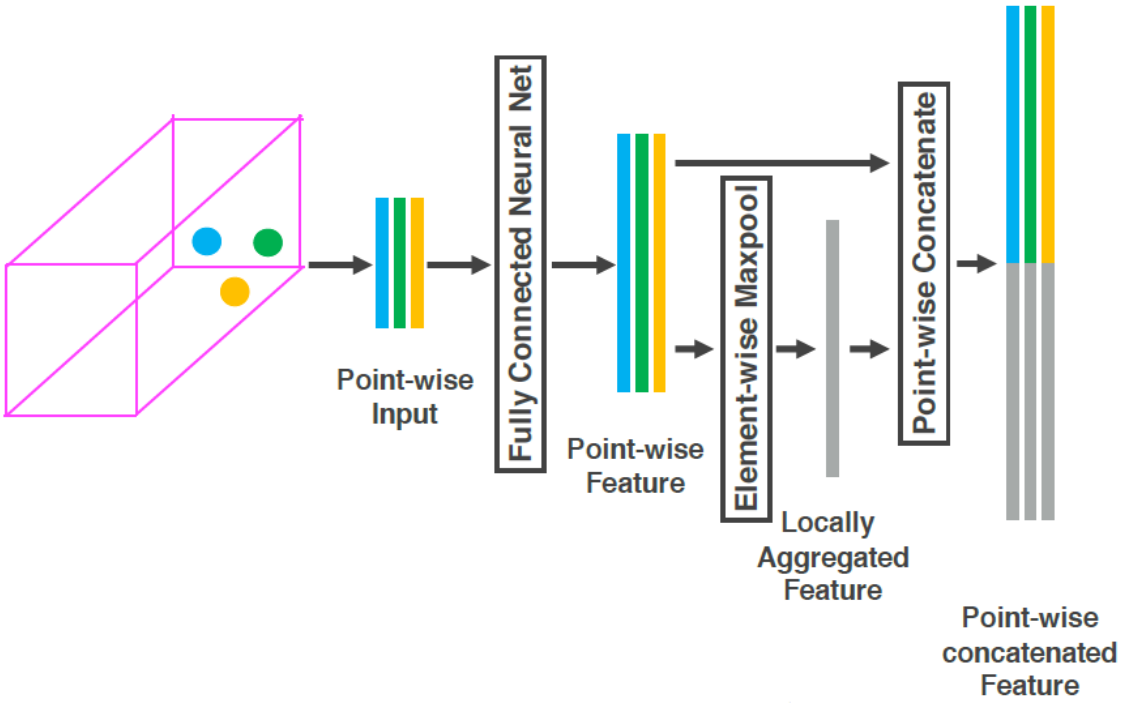
3.5.1 VFE kiht

Nagu oli eelnevalt mainitud, koosneb VoxelNet kolmest osast, millest esimene on tunnuseid õppiv närvivõrk. Kuna punktipilvede puhul on tegemist suure mahuga struktureerimata andmetega ja nende otse töötlemine on väga arvutusmahukas, siis kasutataksegi VoxelNeti mudelis esimese kihina tunnuste eraldamise närvivõrku, kus vähendatakse punktipilve dimensioone ja muudetakse punktipilv tihedaks tensorite struktuuriks [8]. Kui aga konvolutsiooniliste kihtide abil sisendite dimensioone sammhaaval vähendada, siis ei jätku selleks enamasti piisavalt mälu. Selleks on mudelis kasutusel tunnuste eraldamise kiht, et säilitada jõudlus ja kvaliteet. Tunnuste eraldamise kihti kasutatakse ka paljude teiste sarnaste probleemide puhul, kus on kasutusel suuremahulised andmed.

Tunnuste eraldamiseks jagatakse punktipilv kõigepealt võrdsete suurustega vokseliteks, grupeeritakse punktid selle järgi, millises vokselis nad asuvad (*random sampling*) [8]. Üks punktipilv sisaldab enamasti ligikaudu 100 000 punkti, see tähendab, et üle 90% vokselitest on tavaliselt tühjad [8]. Kuna punktipilv on hõre ja punktid paiknevad

kolmemõõtmelises ruumis laiali, siis jääb pärast grupeerimist igasse vokselisse erinev arv punkte [8]. Järgmisena võetakse nendest vokselitest, kus on rohkem kui T punkti, juhuslikult T punkti. T on kindlaks määratud või juhuslikult kindlaks määratud arv, mis määrab, kui palju punkte võib igasse vokselisse maksimaalselt alles jääda [8]. Veel on selle eesmärgiks see, et tehtaks vähem arvutusi ja et vähendada erinevate vokselite vahel olevat punktide arvu tasakaalutust [8].

Lõpuks tulevad järjestikulised tunnuste kodeerimise (VFE) kihid, mille väljundiks ongi tensorid. Saadava tensorite struktuuri dimensioonid on KxTx7, kus K on maksimaalne arv mittetühjasid vokseleid, T on maksimaalne arv punkte, mis võib igas vokselis olla ja 7 on iga punkti sisendikodeeringu dimensioon [8]. Järgneval joonisel (Joonis 8) on illustreeritud ühe VFE kihi osasid.



Joonis 8. VFE kiht [8].

Zhou ja Tuzeli artiklis väljapakutud VoxelNeti uuendus seisneb selles, et tunnuste eraldamise närvivõrgus kasutatakse mitut järjestikulist VFE kihti, sest see aitab paremini õppida keerulisi tunnuseid, et 3D informatsiooni kirjeldada [8]. Kuigi tunnuste eraldamise tehnika oli ka varem olemas, on VoxelNetis kasutusel olev VFE kiht (*feature extraction*) artikli autorite loodud [8]. Kasutusel oleva tunnuste eraldamise kihi VFE uuendus seisneb selles, et punktipõhised (*point-wise*) tunnused on kombineeritud lokaalselt kogutud tunnustega [8]. See tähendab seda, et vokselis arvutatakse iga punkti

kohta tunnus. Saadud tunnustest leitakse *maxpooling* abil antud vokseli parim tunnus, mida nimetatakse lokaalselt kogutud tunnuseks (*locally aggregated feature*) [8]. Lõpuks lisatakse iga punkti tunnusele juurde lokaalselt kogutud tunnus [8].

3.5.2 Keskmised konvolutsioonilised kihid

VoxelNetis olevates konvolutsioonilistes keskkihtides töödeldakse SVFE kihi väljundiks olnud tensorite struktuuri, et koondada ruumilist informatsiooni ja lisada objekti kuju kirjeldusele rohkem konteksti [8]. Tensorite struktuur sisaldab vokselite kodeeritud informatsiooni. Selleks rakendatakse konvolutsioonilistes keskkihtides järjest 3D konvolutsiooni, miniploki normaliseerimise kihti ehk BN kihti (*batch normalization layer*) ja ReLU kihti. Konvolutsiooniliste keskkihtide osa väljundiks on aga tunnuste kaart [8].

3.5.3 Region Proposal Network

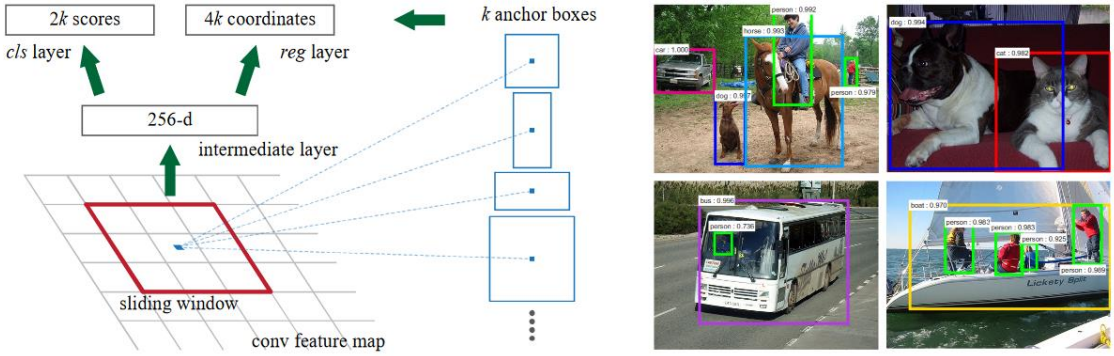
Region Proposal Network (RPN) on algoritm objektide lokaliseerimiseks ehk asukoha ennustamiseks [8]. RPN algoritmi sisendiks olevad andmed peavad olema tihedad ja organiseeritud tensoriteks (näiteks pilt, video) [8].

RPN algoritmi sisendiks on tensoriteks muudetud pilt ning väljundiks hulk tuvastatava objekti ristkülikukujulisi ankurkaste (*anchor box*) ja iga ankurkasti kvaliteedihinnang (*objectness score*), mis näitab, kas objekt kuulub leitavate objektide hulka või mitte [28]. RPN algoritmi implementeerimisel kasutatakse tavaliselt täielikult ühendatud konvolutsioonilist närvivõrku [28].

Objekti asukoha ennustamiseks kasutatakse liikuvat akent (*sliding window*). Liikuva akna keskpunktiks on ankurpunkt (*anchor point*) või ankur [28]. Iga ankurpunkti kohta on number k, VoxelNetis $k = 2$ [8]. Iga ankurpunkti kohta leitakse k ankurkasti, mis võiksid tuvastatavat objekti ümbritseda [28]. Ühe objekti jaoks leitud ankurkastide mõõtmed on samad, kuid erinevus on nende pöördenurgas. Ankurkastid võivad kattuda suuremal või vähemal määral. Närvivõrgu treenimisel leitakse igale ankurkastile IoU, mis näitab, kui hästi ankurkast kattub *ground truth* piirikastiga [28]. RPNi põhiideeks ongi leida, millistel ankurkastidel on suurim IoU kattuvus *ground truth* piirikastiga [28].

RPN algoritmi treenides antakse igale ankurkastile positiivne või negatiivne märgend (*label*), mis ütleb, kas tegemist on objektiga või mitte [28]. Märgendi andmisel

vaadatakse ankurkastidele antud IoU-d. Positiivne märgend antakse ankurkastidele, millel on kõige suurem IoU kattuvus *ground truth* piirikastiga või mille IoU kattuvus on suurem seatud ülemisest piirist (näiteks 0.7) [28]. Negatiivne märgend antakse ankurkastidele, mille kattuvus *ground truth* piirikastiga on väiksem alumisest piirist (näiteks 0.3) [28]. Neid ankurkaste, millele ei ole märgendit antud, ei arvestata treenimisel [28]. Järgnevalt on toodud (Joonis 9) näide RPN närvivõrgust ja ankurkastidest.



Joonis 9. Vasakpoolsel pildil on toodud RPN. Parempoolsel pildil on näited ankurkastidest [28].

VoxelNetis on RPN algoritmi sisendiks konvolutsiooniliste kihtide väljundiks olev tunnuste kaart [8]. Zhou ja Tuzeli artiklis kirjeldatud RPN sisaldab kolme täielikult konvolutsioonilistest kihtidest koosnevat plokki [8]. Iga ploki esimene kiht vähendab tunnuste kaardi suurust poole võrra [8]. Pärast iga konvolutsioonilist kihti rakendatakse miniplokkide normaliseerimist (BN) ja ReLU operatsiooni [8]. Lõpuks saadakse kõrgeresolutsiooniline tunnuste kaart, mis seotakse tõenäosuste skoori kaardiga (*probability score map*) ja regressiooni kaardiga (*regression map*) [8].

4 Tehniline lahendus

Antud peatükis on võetud eksperimentide aluseks Githubist kasutaja Skyhehe123 poolt loodud VoxelNeti mitteametlik realisatsioon ja antud töö juhendaja Martin Rebase loodud VoxelNeti mitteametlik realisatsioon. Kuigi VoxelNetist on erinevaid implementatsioone, siis ametlikku implementatsiooni ei ole. On palju mitteametlikke, kus on näiteks kasutatud masinõppeteeke TensorFlow, Keras või Pytorch. Mõlemad töös kasutatud VoxelNeti implementatsioonid kasutavad masinõppeteeki Pytorch[1].

Antud töös kasutatud VoxelNeti algoritmid on mõeldud LiDARi andmetest autode tuvastamiseks. See, et närvivõrk kohanduks inimeste peale, võib olla keeruline. Selleks võib vaja olla andmete rikastamise tehnikaid, näiteks punktitiheduse muutmist või närvivõrgu muutmist, sest ei saa eeldada, et inimene ja auto on sama hästi tuvastatavad ja lokaliseeritavad. Martin Rebase algoritmi puhul on näiteks osade eksperimentide puhul lisatud juurde täielikult ühendatud kihte, et närvivõrgu sügavust suurendada.

Mõlema töös kasutatud VoxelNeti algoritmi puhul on närvivõrgu treenimisel kasutusel ankurkastid, kuid valideerimisel piirikastid. Nii Skyhehe123 kui ka Martin Rebase loodud algoritmi puhul on ankurkastide kattuvuse ehk IoU ülemiseks piiriks (*positive threshold*) 0,6 ja alumiseks piiriks (*negative threshold*) 0,45, kuna antud väärtuseid on soovitatud VoxelNeti autorite artiklis. Kui kattuvus on suurem ülemisest piirist, siis loetakse ennustus positiivseks ja tegemist on positiivse ankurkastiga, kui alla alumise piiri, siis on tegemist negatiivse ankurkastiga. Kui kattuvus jääb ülemise ja alumise piiri vahele, siis on tegemist neutraalse ankurkastiga, mida ignoreeritakse. Kui valideerimisel on piirikastide puhul oluline ainult IoU ülemine piir, siis treenimisel on oluline ka IoU alumine piir ehk nii neutraalsed ankurkastid, mida ignoreeritakse, kui ka negatiivsed ankurkastid [28].

Kogu töös kasutatud närvivõrgu kood on kirjutatud programmeerimiskeeles Python. Lisaks on kasutatud masinõppe teeki Pytorch ja matemaatilisteks arvutusteks mõeldud

[1] www.pytorch.org

teeki Numpy[1]. Närvivõrkudega saadud tulemuste visualiseerimiseks on kasutatud visualiseerimiseks vajalikku teeki Visdom[2]. Esimese implementatsiooni puhul on kasutatud ka tehisnägemise ja masinõppe teeki OpenCV[3], et salvestada ennustatud piirikastidega pilte.

Järgnevalt on kirjeldatud VoxelNeti koodi käivitamise jaoks virtuaalkeskkonna loomist ning on esitatud mõlema VoxelNeti mitteametliku implementatsiooniga saadud tulemused.

4.1 Närvivõrgu treenimiseks mõeldud keskkonna seadistamine ja konfigureerimine

Tavaliselt sisaldab LiDARit kasutades saadud punktipilv ligikaudu 100 000 punkti ehk punktipilve närvivõrgus kasutamiseks võivad olla kõrged arvutuslikud ja mälu nõuded [7]. Seetõttu tuli närvivõrkude treenimiseks leida sobiv keskkond.

Antud töös prooviti Githubist võetud VoxelNeti koodi käivitada nii enda sülearvutis, Google Colaboratory's, Google Cloudis kui ka Tallinna Tehnikaülikooli arvutusklastris. Kõikides keskkondades koodi käivitamiseks loodi seal teaduslike arvutuste jaoks mõeldud platvormi Anaconda[4] abil virtuaalkeskkonnad, kuhu installeeriti vajalikud paketid. Virtuaalkeskkondade loomise juhend on toodud lisades (Lisa 1). Kuna sülearvutis oli liiga vähe muutmälu (RAM), siis prooviti koodi käivitada Google Colaboratory's, mis on masinõppe jaoks mõeldud brauseris töötav keskkond, kus on võimalik kasutada Jupyter notebook'e [29]. Lisaks CPU-le on seal võimalik kasutada ka GPU-d [30]. Google Colaboratory's oli muutmälu ligikaudu 12 GB [30], kuid mälu eraldamine ebaõnnestus korduvalt ja mudeli treenimine ei olnud võimalik. Teiseks põhjuseks oli see, et Google Colaboratory on mõeldud pigem katsetusteks kui suuremate närvivõrkude treenimiseks. Järgmisena prooviti kasutada Google Cloud'i, kuid kuna treenimiseks oli vaja sadu GPU töötunde, siis eraldatud krediidist ei piisanud.

[1] www.numpy.org

[2] https://github.com/facebookresearch/visdom

[3] www.opencv.org

[4] https://www.anaconda.com/distribution/

Lõpuks otsustati Tallinna Tehnikaülikooli arvutusklastri kasuks, kuid ka seal ilmnesid mõned probleemid. Githubist võetud Skyhehe123 koodi käivitades ei olnud võimalik kasutada GPU-d ja tuli närvivõrku treenida vaid CPU peal. Probleemiks võis olla, et klastris kasutusel olnud CUDA uuem versioon 9.2.88 ei ühildu Pytorch 0.3.1 versiooniga. Õigem on aga kasutada tihedate tensorite struktuuri puhul GPU-sid, sest need on optimiseeritud töötlema tihedaid tensorite struktuure [8]. Lisaks loodi arvutusklastris närvivõrgu treenimise ajal automaatselt kaks faili, millest ühes olid kirjas treenimise jooksul tekkinud vead ja teises väljund. Juhendaja VoxelNeti teostuse baasil loodud lahendust oli võimalik käivitada GPU-sid kasutades ja tänu sellele oli treenimine kiirem.

Tallinna Tehnikaülikoolis olevas arvutusklastris on 232 arvutusmasinat [31]. Igas arvutusmasinas on 24 Inteli 64bit protsessorit ja 48 GB mälu [31]. Närvivõrgu treenimiseks kasutatud masinas on aga 88 CPU tuuma, 400 GB RAM-i ja 5 P100 GPU-d. Koodi jooksutamiseks loodi bash skriptid, mis Pythoni programmikoodi käivitasid. Masinas on kasutusel tööde järjekorra süsteem SGE[1].

Võib öelda, et VoxelNeti implementatsioonide treenimiseks sobiva keskkonna leidmine ja seal virtuaalkeskkondade loomine oli üks ajamahukamaid tegevusi.

4.2 VoxelNeti täiendamine ja treenimisel saadud tulemused

Antud alapeatükis on toodud mõlema VoxelNeti mitteametliku implementatsiooniga saadud tulemused. Töös kasutatud kood on üleval GitLabi repositooriumis[2]. Kasutatud kood käivitati Tallinna Tehnikaülikooli arvutusklastris Anaconda abil loodud virtuaalkeskkonnas, kuhu olid installeeritud vajalikud paketid.

Kuna mõlemad algoritmid on keerulised ning nende treenimiseks on hea, kui on palju positiivseid näiteid, siis koostati närvivõrkude treenimiseks juhendaja poolt pakutud esialgsest andmestikust väiksem andmestik, mis sisaldab ainult selliseid punktipilvi, milles esineb jalakäijaid, istuvaid inimesi ja jalgrattureid. Mõlema algoritmi puhul töödeldakse korraga ühte punktipilve.

[1] https://www.uibk.ac.at/zid/systeme/hpc-systeme/common/tutorials/sge-howto.html

[2] https://gitlab.cs.ttu.ee/kapare/iapb

Zhou ja Tuzeli artiklis hinnati VoxelNeti tööd linnulennulisest vaatest ja kolmemõõtmelisest ruumist objektide tuvastamisel, kasutades selleks vaid LiDARi andmeid [8]. Antud töös on püütud korrata või valideerida artiklis toodud tulemusi. Selleks on muudetud õpisammu, aktivatsioonifunktsiooni, punktitihedust ja vokseli objekti sisalduvuse tõenäosust. Õpisamm on kõige olulisem hüperparameeter, mille leidmiseks peaks kõige rohkem katsetama [32]. Liiga väikese õpisammu puhul võib treenimine võtta liiga kaua aega, kuid liiga suur õpisammu väärtus võib tähendada, et jõutakse mitteoptimaalse tulemuseni või et treenimisprotsess on ebastabiilne [32]. Väiksema õpisammu puhul võib treeningepohhide arv olla suurem, suurema õpisammu puhul aga toimuvad muudatused kiiresti ja võib vaja olla vähem epohhe [32].

Närvivõrguga tehtud ennustused (Skyhehe123 koodi puhul leitud piirikastide arv või Martin Rebase koodi puhul, kas voksel katab otsitavat objektitüüpi) saab jagada järgnevalt: *true positive* (mudel ennustas positiivse ehk otsitava klassi õigesti), *true negative* (mudel ennustas negatiivse klassi õigesti), *false positive* (mudel ennustas positiivse klassi valesti), *false negative* (mudel ennustas negatiivse klassi valesti) [33]. Tabelis on toodud ennustuste jaotamise viis (Tabel 1).

Tabel 1. Närvivõrguga tehtud ennustuste jaotamise viis [33].

Ennustus/klassifitseering **Tegelik**	**Positiivne**	**Negatiivne**
Positiivne	Tõene positiivne (*True positive*)	Väär positiivne (*False positive*)
Negatiivne	Väär negatiivne (*False negative*)	Tõene negatiivne (*True negative*)

Skyhehe123 loodud algoritmi puhul on tegemist piirikastide põhise ja Martin Rebase VoxelNeti algoritmi puhul vokselipõhise meetrikate arvutamisega. Täpsuste arvutamisel tuli arvestada andmestikus olevate klasside arvuga, kuna tegemist on mitut klassi sisaldava klassifitseerimisega (*multi-class classification*), kus klassid ennustatakse

positiivsesse või negatiivsesse klassi. Positiivsesse klassi loetakse järgnevad klassid: jalakäijad, istuvad inimesed ja jalgratturid. Negatiivsesse klassi aga kõikidesse teistesse klassidesse kuuluvad objektid, mida võib piltidelt tuvastada, näiteks autod, veoautod ja muu. Esiteks leiti mudeli õigsus (*accuracy*), mis näitab, kui suur osa mudeli tehtud ennustustest olid õiged. Kuna õigsus ei ole alati parim meetrika, mille abil mudelit valida, siis leiti ka mudeli täpsus (*precision*), saagis (*recall*) ning F1 Skoor (*F1 score*) [33]. Õigsuse arvutamiseks kasutati järgnevat valemit [33]:

$$accuracy = \frac{true\ positives + false\ negatives}{true\ positives + true\ negatives + false\ positives + false\ negatives} \quad (1)$$

Täpsuse abil saab hinnata seda, kas väärate positiivsete hulk on suur ehk kui palju andmetest, mis on ennustatud positiivseks, on ka tegelikult positiivsed [33]. Skyhehe123 närvivõrgu puhul näiteks, kui suur osa õigeks ennustatud piirikastidest on tegelikult õiged. Martin Rebase loodud algoritmi puhul aga, kui suur osa positiivseks ennustatud vokselitest on positiivsed. Täpsuse arvutamise valem [33] on

$$precision = \frac{true\ positives}{true\ positives + false\ positives} \quad (2)$$

Saagis näitab kui suur osa positiivsetest andmetest ennustati õigesti [33]. Saagis arvutati valemiga (3) [33].

$$recall = \frac{true\ positives}{true\ positives + false\ negatives} \quad (3)$$

Selleks, et mudeli efektiivsust hinnata, tuleb aga võrrelda nii täpsust kui ka saagist, sest need on omavahel seotud. Täpsuse parandamine vähendab saagist ja vastupidi [34]. F1 skoori kasutatakse, kui otsitakse tasakaalu täpsuse ja saagise vahel ning kui klassid on jaotatud ebavõrdselt ja suurem osa andmetest kuulub klassi, mida ei tuvastata [33]. Seda arvutatakse järgmise valemiga (4) [33]:

$$F1 = 2 * \frac{(precision * recall)}{precision + \mathrm{recall}} \quad (4)$$

4.2.1 Esimese VoxelNeti implementatsiooniga saadud tulemused

Töö tulemuste saamiseks täiendati Githubist võetud VoxelNeti koodi. Kasutatud realisatsioon on loodud kasutaja Skyhehe123 poolt. Juurde lisati IoU arvutamine, erinevate meetrikate arvutamine, mudelite salvestamine ja Visdomiga graafikute kuvamine, et saaks teha järeldusi treenitud mudelite kohta. Lisaks muudeti ka

punktitihedust ehk vokselites olevat punktide arvu. Kui algselt oli punktitiheduseks 35, kuna algoritm oli loodud autode tuvastamise ja lokaliseerimise jaoks, siis inimeste tuvastamiseks ja lokaliseerimiseks määrati punktitiheduseks 45 nagu on soovitatud VoxelNeti artiklis [8]. Autode puhul määratakse vokselis olevate punktide arvuks enamasti 35 [8]. Veel lisati juurde treeningtulemuste salvestamine failidesse ja täiendati piltide salvestamist. Salvestati nendele punktipilvedele vastavad pildid, millest algoritm tuvastas inimesi. Salvestatud piltidele lisati algoritmi poolt ennustatud piirikastid.

Antud koodi on varem käivitatud autode tuvastamiseks, kuid tulemusi inimeste tuvastamise või lokaliseerimise kohta ei ole. Autorile teadaolevalt ei ole vabalt kasutatavat töötavat ja valideeritud VoxelNeti koodi inimeste tuvastamiseks ja lokaliseerimiseks, sest artiklis väljapakutud VoxelNeti algoritm loodi firmas Apple Inc. ning seda ei ole avalikustatud.

Kuna valitud VoxelNeti teostuses puudus valideerimise osa, siis programmeeriti kahe piirikasti kattuvuse ehk IoU leidmiseks vajalik osa, võttes aluseks töö juhendaja VoxelNeti kood. IoU piiriks määrati 0,5 ehk kui IoU on jalakäijate tuvastamisel üle 50%, siis loetakse ennustatud piirikast õigeks. Autode puhul peaks IoU aga vähemalt 60% olema [8].

Masinõppes kasutatakse algoritmi headuse hindamiseks kaofunktsiooni. Selle abil saab hinnata, kui hästi algoritm modelleerib etteantud andmeid. Mida väiksem on kaofunktsiooni väärtus, seda parem on treenides saadud mudel, sest see tähendab, et ennustusel tehtud viga on väike [35]. Kaofunktsiooni väärtus näitab, kui hästi mudel käitub pärast igat iteratsiooni [35]. Antud töös kasutatud Skyhehe123 VoxelNeti puhul on kogu kao saamiseks arvutatud klassifitseerimise kadu (*classification loss*) ja regressiooni kadu (*regression loss*). *Classification loss* näitab, kui suur on tõenäosus, et üks ankurkast sisaldab objekti, mida soovitakse tuvastada ja lokaliseerida [28]. *Regression loss* on seotud ennustatud objekti koordinaatidega [28]. Klassifitseerimise kadu on jagatud kaheks: *positive cases loss* ja *negative cases loss* ehk kas tegemist on objektiga või mitte [28]. Et *positive cases loss*-le suurem kaal anda, on kadu kaheks jagatud. Regressiooni kao jaoks on kasutatud funktsiooni *smooth L1* [28]. Regressioonikadu arvutatakse vaid positiivse märgendiga ankurkastide puhul [28].

Skyhehe123 poolt loodud algoritmi treeniti CPU peal, kuna koodil puudus mitme GPU kasutamise tugi ning ühe GPU kasutamisel tekkisid veateated. Kuna treeningandmestikus oli 1210 punktipilve ja algne minipartii suurus on 2, siis otsustati iteratsioonide arvuks valida 605. Minipartii suurus 2 tähendab, et ühes iteratsioonis töödeldakse järjest 2 punktipilve. Järgnevalt (Tabel 2) on toodud Skyhehe123 närvivõrguga tehtud katsetes kasutatud hüperparameetrid.

Tabel 2. Skyhehe123 loodud VoxelNeti mitteametliku implementatsiooniga tehtud katsed.

Katse nr	**Iteratsioonide arv**	**Maksimaalne punktide arv vokselis**	**Miniploki suurus**	**Aktivatsioonifunktsioon**	**Õppimismäär**
1	605	45	2	SGD	0,01
2	605	45	2	SGD	0,1
3	605	45	2	Adam	0,01
4	605	45	2	SGD	1,0

Järgnevalt on esitatud närvivõrgu treeningute tulemused (Tabel 3). Teist, kolmandat ja neljandat katset ei olnud võimalik läbi viia, kuna nende katsete puhul oli mälukasutus närvivõrku treenides peale viiendat iteratsiooni nii suur, et Tallinna Tehnikaülikooli klastris olevast 400 GB-st mälust tarbiti ligikaudu 80% ja teiste kasutajate säästmiseks ei olnud võimalik algoritmi edasi treenida.

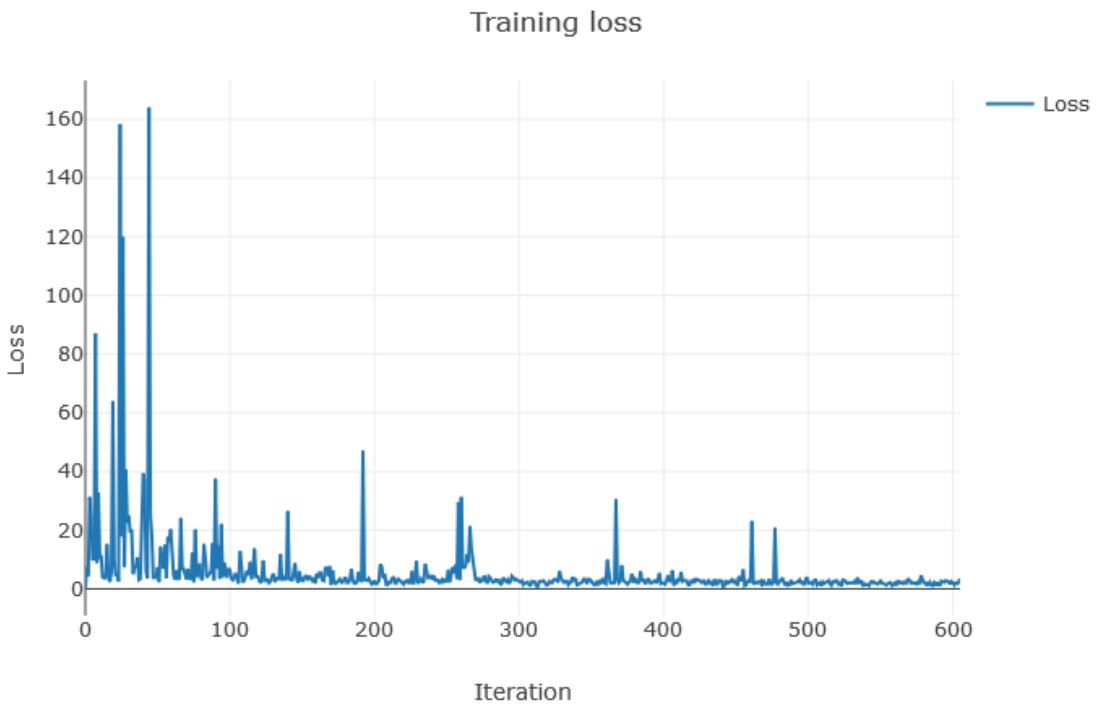
Tabel 3. Skyhehe123 loodud VoxelNeti mitteametliku implementatsiooniga saadud tulemused.

Katse nr	**Treenimise aeg tundides**	**Kadu (*loss*)**	**Õigsus (*accuracy*)**	**Täpsus (*precision*)**	**Saagis (*recall*)**	**F1 skoor (*F1 score*)**
1	17,9	0,73850	0,99995	0,99995	0,02797	0,02914

Andmestikus oli 2992 märgistatud (*ground truth*) piirikasti jalakäijate, istuvate inimeste ja jalgratturite tuvastamiseks ja lokaliseerimiseks. Algoritm ennustas 12989 piirikasti, millest 107 olid õigesti ennustatud. Minimaalseks kao väärtuseks saadi 0,73850 ehk ligikaudu 74%. Võib öelda, et mudel tegi kohati väga täpseid ennustusi, kuna nii õigsus kui ka täpsus olid vahepeal ligikaudu 99%. Saagis oli aga ligikaudu 2,8%, kuigi ennustatud piirikastide hulka ja neist õigesti ennustatud piirikastide arvu vaadates oleks see pidanud olema ligikaudu 3,8%, see tähendab, et viga võis olla ennustatud piirikastide arvu leidmisel või saagise arvutamisel.

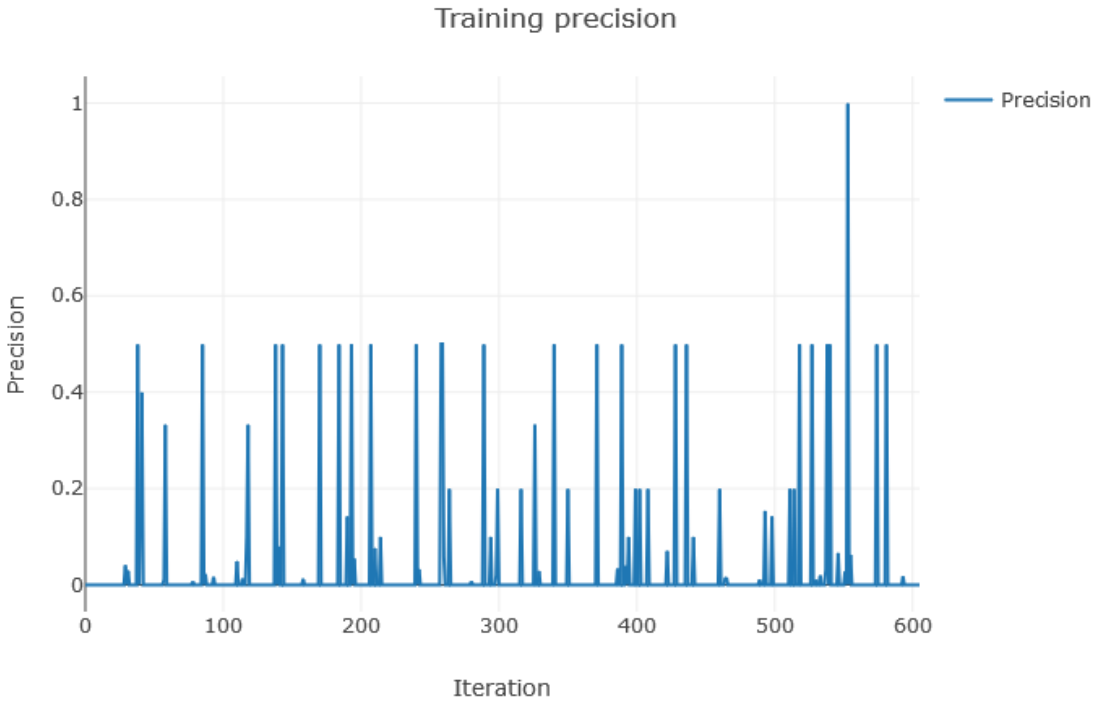
Järgnevalt on toodud esimesel treeningul saadud kaofunktsiooni graafik (Joonis 10). Võib öelda, et esimesel katsel valitud õpisamm (0,01) ei olnud kõige sobilikum, kuna graafikul on näha, et kaofunktsiooni väärtus võnkus küllaltki palju. Lisaks võiks iteratsioonide arv suurem olla, et graafikul oleks täpselt näha, kuhu graafik koondub.

Veel on esitatud esimesel treeningul saadud õigsuse graafik (Joonis 11). Graafikut vaadates võib öelda, et närvivõrku võiks treenida rohkem kui 600 iteratsiooni, kuna joon võngub tugevalt. Peale kao ja õigsuse kujutati graafikutel veel täpsust (Joonis 12), saagist (Joonis 13) ja F1 skoori (Joonis 14).

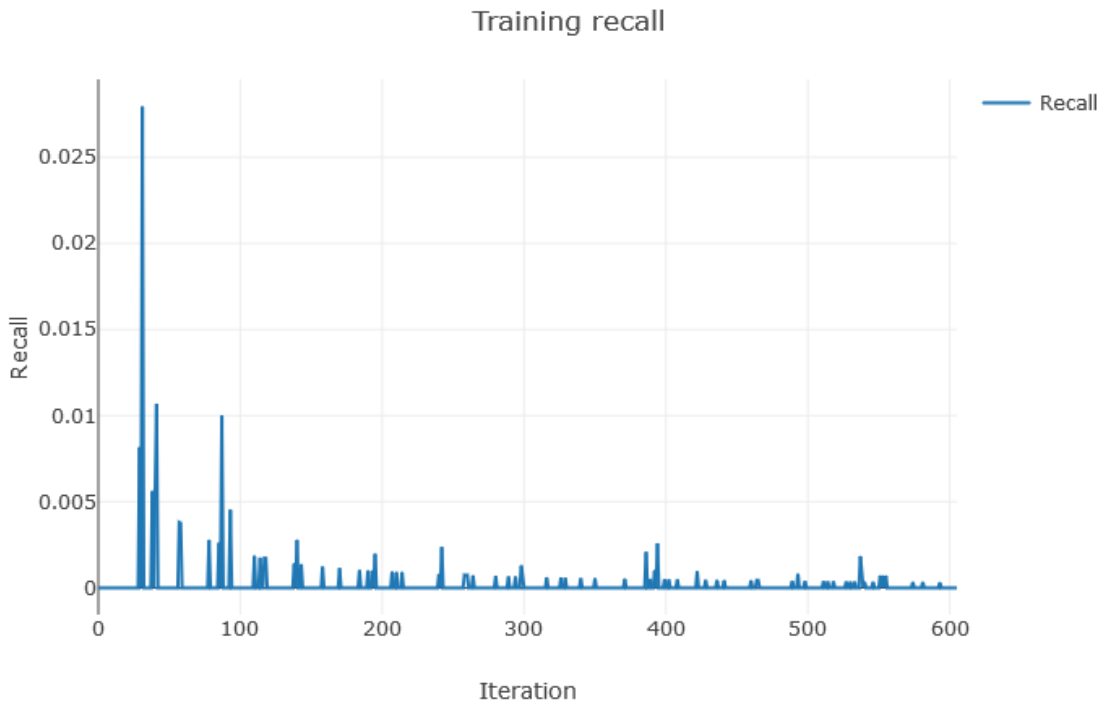


Joonis 10. Skyhehe123 poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud kaofunktsiooni väärtused.

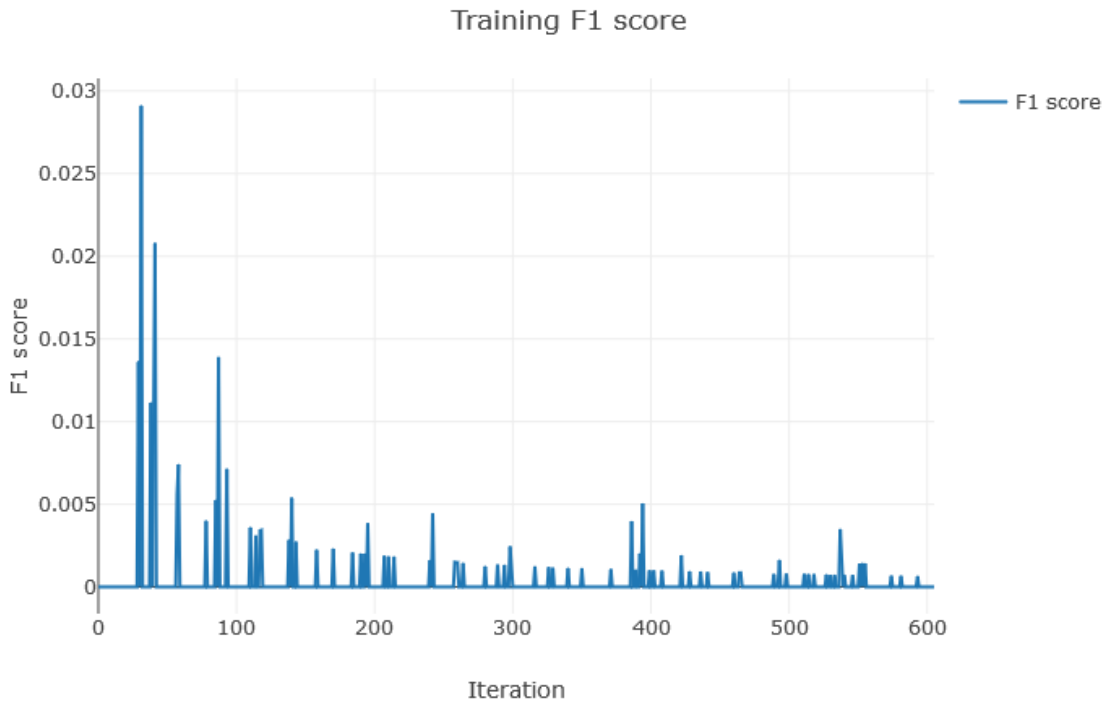
Joonis 11. Skyhehe123 poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud õigsuse (*accuracy*) väärtused.



Joonis 12. Skyhehe123 poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud täpsuse (*precision*) väärtused.



Joonis 13. Skyhehe123 poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud saagise (*recall*) väärtused.

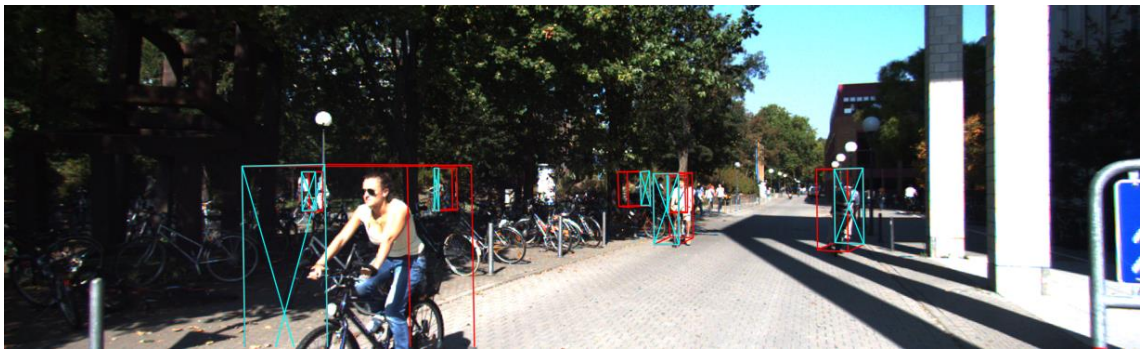


Joonis 14. Skyhehe123 poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud F1 skoori (*F1 score*) väärtused.

Võib öelda, et esimese eksperimendi tulemused olid kesised, kuna paljudes punktipilvedes, milles algoritm suutis inimese tuvastada ja lokaliseerida, olid piirikastid lisatud valideerimiseks mõeldud piltidel valesse kohta. Närvivõrgu treenimisel läbiti ainult 1 epohh ja selle jooksul suutis antud realisatsioon väga vähe inimesi tuvastada ja lokaliseerida, kuna saagis ehk õigete objektide tuvastamise hulk oli vaid 3%. Allpool on toodud mõned näited esimese treeningu jooksul salvestatud piltidest, millele lisati Skyhehe123 närvivõrguga treenimisel õigesti ennustatud piirikastid.

Joonis 15. Näide õigesti leitud piirikastist.

Joonis 16. Näide valesti leitud piirikastist.



Joonis 17. Näide õigesti leitud piirikastidest.

4.2.2 Teise VoxelNeti implementatsiooniga saadud tulemused

Teiseks kasutati juhendaja poolt loodud VoxelNeti koodi. Lisaks kao graafiku kuvamisele lisati juurde täpsuste graafikute loomine ja kuvamine ning treenimisel saadud mudelite salvestamine. Lisaks arvutati peale õigsuse ka täpsus, saagis ja F1 skoor. Juhendaja poolt antud algoritmi treeniti GPU-de peal. Juhendaja poolt pakutud VoxelNetis on tegemist vokselipõhise (*voxelwise*) meetrikate arvutamisega. Objekt loetakse leituks, kui vähemalt üks objekti sees olevatest vokselitest on klassifitseeritud seda objekti sisaldama ehk piirikaste otsitakse vokselite kaupa. See tähendab, et piirikast loetakse leituks, kui sellest on leitud vähemalt üks voksel. Taolise lahenduse puhul aga on keeruline arvutada, kuidas leida valesti leitud piirikastide arvu.

Eksperimenteeriti ka närvivõrgu arhitektuuriga, et testida, kas vokselipõhist lähenemist saab täiustada. Selleks katsetati erinevaid arhitektuure, näiteks lisati juurde vahepealseid täielikult ühendatud kihte.

Treenimisel toimub iga punktipilve kasutamise järel valideerimine. Erinevalt piirikastide põhisest, ei anna vokselipõhine lahendus väärate positiivsete ega tõeste negatiivsete arvu, sest neid ei ole võimalik lihtsa meetodiga taandada vokselite kujult objekti kujule. Seetõttu on kasutusel ainult tõeste positiivsete ja väärate negatiivsete arv, kuigi kaudse meetodiga oleks võimalik leida ka väärate positiivsete arv. Järgnevas tabelis (Tabel 4) on toodud närvivõrku treenides kasutatud hüperparameetrid.

Tabel 4. Martin Rebase loodud VoxelNeti mitteametliku implementatsiooniga tehtud katsed.

Katse nr	**Epohhide arv**	**Maksimaalne punktide arv vokselis**	**Miniploki suurus**	**Optimeerimisfunktsioon**	**Õppimismäär**	**Tõenäosus, et voksel sisaldab objekti**	**Täielikult ühendatud vahekihtide (FCN) suurused**
1	600	45	1	SGD	0,01	0,7	(128, 32) (32, 8) (8, 1)
2	600	45	1	SGD	0,1	0,7	(128, 32) (32, 8) (8, 1)
3	600	45	1	Adam	0,01	0,7	(128, 32) (32, 8) (8, 1)
4	600	45	1	SGD	0,01	0,6	(128, 32) (32, 8) (8, 1)
5	600	45	1	SGD	0,01	0,8	(128, 32) (32, 8) (8, 1)
6	600	45	1	SGD	0,01	0,7	(128, 64) (64, 32) (32, 8) (8, 1)
7	600	45	1	SGD	0,01	0,7	(128, 64) (64, 32) (32, 16) (16, 8) (8, 1)

Allpool (Tabel 5) on toodud närvivõrguga saadud tulemused. Kasutatud VoxelNeti teostuses kasutati vokselipõhise kao arvutamisel (*voxelwise loss*) binaarset ristentroopiakahju (*binary cross-entropy*).

Tabel 5. Martin Rebase loodud VoxelNeti mitteametliku implementatsiooniga saadud tulemused.

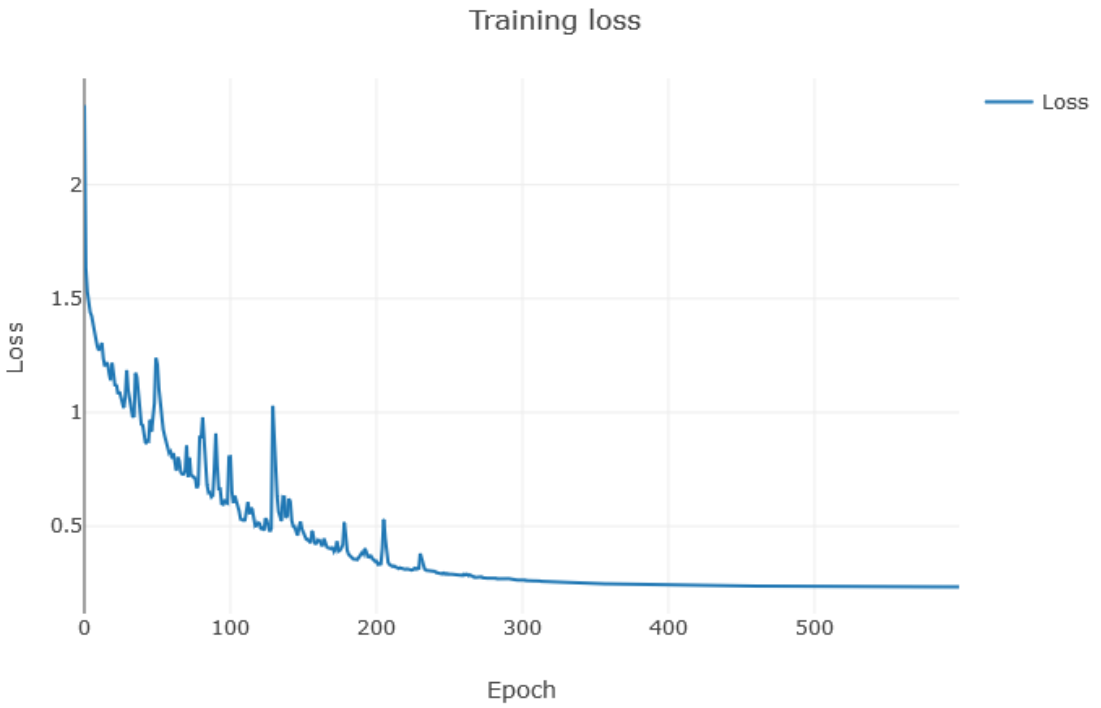
Katse nr	**Treenimise aeg minutites**	**Kadu (*loss*)**	**Õigsus (*accuracy*)**	**Täpsus (*precision*)**	**Saagis (*recall*)**	**F1 skoor (*F1 score*)**
1	19,0	0,25	0,96	0,25	0,86	0,39
2	18,9	0,23	0,95	0,25	0,87	0,38
3	19,1	0,23	0,95	0,27	0,85	0,38
4	18,9	0,23	0,95	0,24	0,86	0,37
5	18,9	0,20	0,96	0,28	0,87	0,42
6	19,2	0,20	0,96	0,26	0,88	0,39
7	19,8	0,23	0,96	0,26	0,86	0,39

Võib öelda, et ilma RPN osata VoxelNeti implementatsiooniga saadud tulemused on küllaltki head. Näiteks on kao väärtused väiksed. Kõikide katsete puhul jäi kadu lõpuks ligikaudu 23% lähedale. Õigsuse väärtused on üle 95%. Tabelis olevad täpsused on 0,25 lähedal ehk 25%, mis tähendab, et iga 1 õige vokseli kohta leitakse 4 vale vokselit. Lisaks on algoritmi saagis küllaltki kõrge.

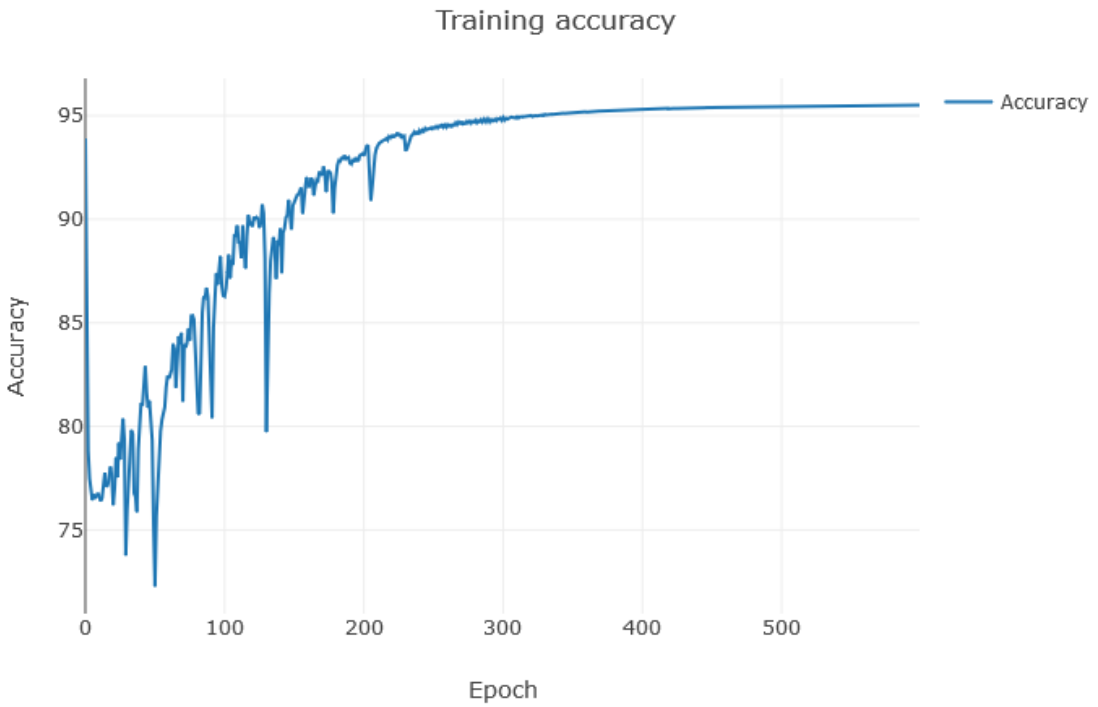
Töös katsetatud VoxelNeti arhitektuuri muudatused ei parandanud närvivõrgu tulemusi oluliselt. Ka katsed 6 ja 7 näitavad, et antud närvivõrgule kihtide lisamine ehk närvivõrgu sügavuse suurendamine ei parandanud tulemusi märkimisväärselt.

Järgnevalt on toodud mudeli kao (Joonis 18), õigsuse (Joonis 19), täpsuse (Joonis 20), saagise (Joonis 21) ja F1 skoori (Joonis 22) graafikud. Saagise graafikult on näha, et saagis tõuseb küllaltki kiiresti. Sellest võib järeldada, et alguses märgitakse suur osa punkte sisaldavatest vokselitest objekti sisaldavateks ja alles siis eemaldatakse üleliigsed vokselid, mis tegelikult ei sisalda objekti. See võib olla põhjustatud kaalufunktsioonist, kus positiivse vokseli kaal on alguses 300 ja seda püütakse kiirelt

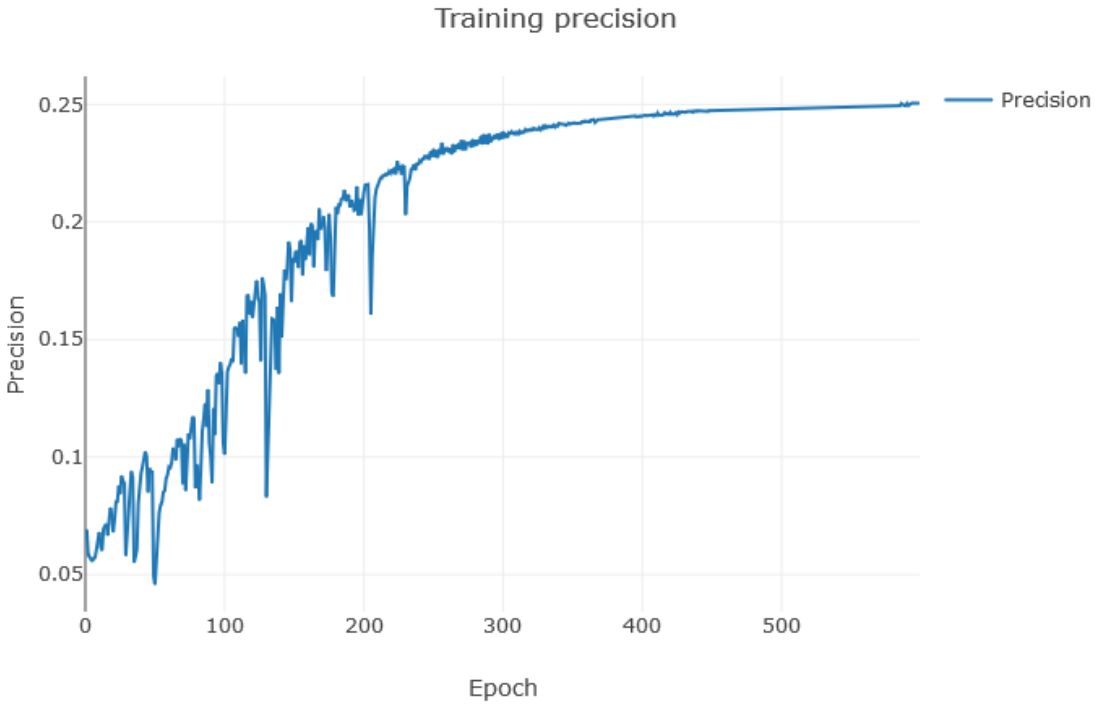
vähendada, et eemaldada üleliigsed vokselid. Üleliigsete vokselite eemaldamine algab samal ajal, kui täpsuse graafik hakkab tõusma ja kao graafik järk-järgult vähenema.



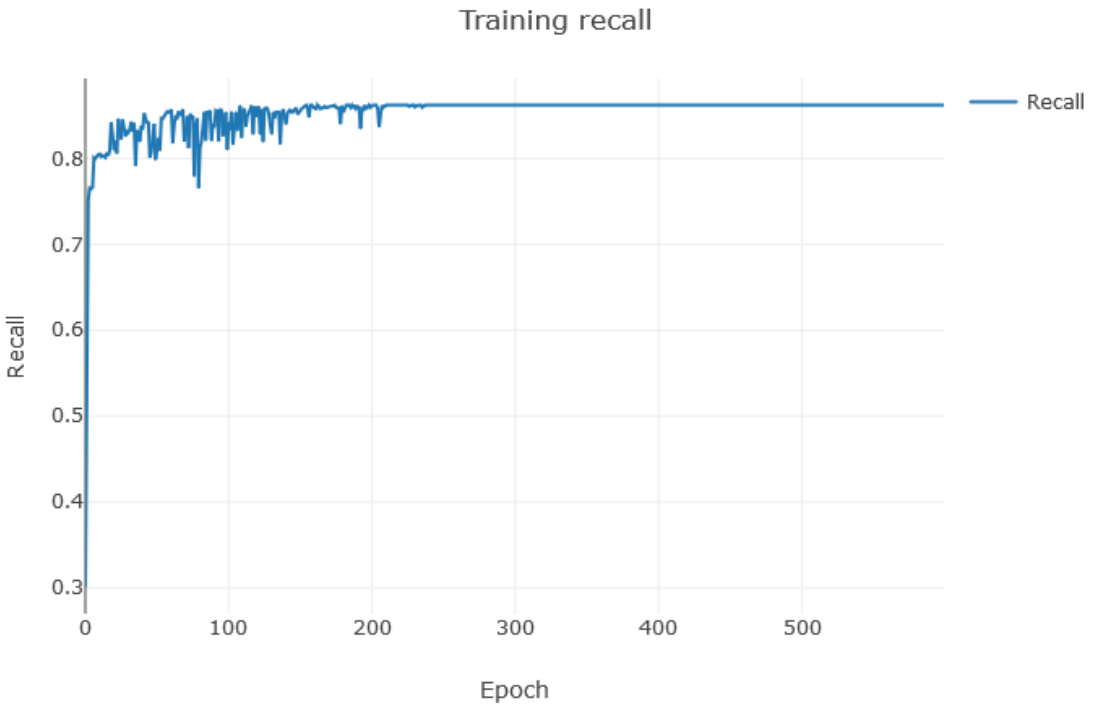
Joonis 18. Martin Rebase poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud kaofunktsiooni väärtused.



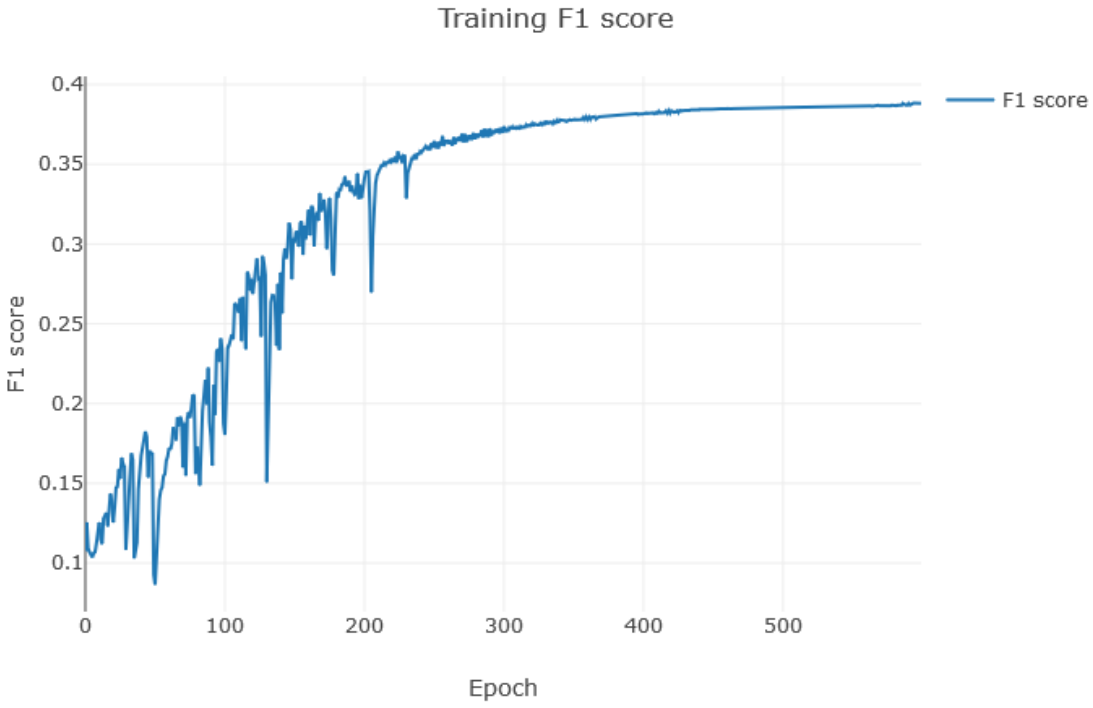
Joonis 19. Martin Rebase poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud õigsuse (*accuracy*) väärtused.



Joonis 20. Martin Rebase poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud täpsuse (*precision*) väärtused.



Joonis 21. Martin Rebase poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud saagise (*recall*) väärtused.



Joonis 22. Martin Rebase poolt loodud VoxelNeti implementatsiooniga esimesel treenimisel saadud F1 skoori (*F1 score*) väärtused.

4.2.3 Järeldused inimeste tuvastamise ja lokaliseerimise kohta

Antud töös kasutati inimeste tuvastamiseks ja lokaliseerimiseks punktipilves kahte VoxelNeti mitteametlikku implementatsiooni. Tuvastatavateks klassideks olid jalakäijad, istuvad inimesed ja jalgratturid. Võib öelda, et mõlemast katsetatud VoxelNeti realisatsioonist saadi paremaid tulemusi Martin Rebase poolt loodud algoritmiga. Lisaks toimus treenimine kiiremini võrreldes Skyhehe123 algoritmiga, kuid selle põhjuseks võis olla RPN osa puudumine.

Kui Skyehe123 algoritmiga saadi kao väärtuseks esimesel treeningul ligikaudu 74%, siis Martin Rebase närvivõrguga vaid 23%, mis on palju parem, sest närvivõrke treenides soovitakse saada lõpuks võimalikult väike kao väärtus. Õigsus oli mõlemal algoritmil küllalti hea, üle 95%. Teise algoritmi puhul olid aga täpsuse väärtused küllalki madalad kõikide treeningute puhul. Esimese algoritmiga saadi küll täpsuse väärtuseks 99%, kuid antud tulemus ei ole eriti usaldusväärne. Kui Skyhehe123 närvivõrguga saadi saagiseks ligikaudu 3% ehk õigete objektide tuvastamise hulk oli väike, siis teise algoritmiga olid saagise väärtused üle 80%, mis on väga hea tulemus.

F1 skoor ehk täpsuse ja saagise tasakaal oli esimese närvivõrgu puhul ligikaudu 2,9%, mis tähendab, et tasakaal ei olnud paigas, mida võib öelda ka täpsuse ja saagise väärtuste põhjal, kuna täpsus oli 99%, kuid saagis kõigest ligikaudu 3%. Teise puhul aga jäid F1 skoori väärtused 39% lähedale, mis on parem kui esimese närvivõrgu puhul, kuid ka siin olid väärtused tasakaalust väljas. Täpsuse väärtused jäid ligikaudu 25% lähedale, aga saagis oli üle 80%. Esimene närvivõrk andis võrreldes teisega täpsemaid tulemusi, kuid saagis oli väike. Teine algoritm oli aga ebatäpsem, kuid saagis oli kõrgem.

4.3 Edasiarenduse võimalused

Käesoleva töö tulemusel leiti, et töös kasutatud punktipilvedest objektide tuvastamiseks mõeldud närvivõrk VoxelNet ei ole kõige parem algoritm inimeste tuvastamiseks ja lokaliseerimiseks punktipilvest. Leidub ka paremaid võimalusi inimeste lokaliseerimiseks punktipilves, näiteks PointNet, mis tegelikult oli üks esimesi punktipilvest objektide tuvastamiseks mõeldud närvivõrke ja inspireeris VoxelNeti loomist [36]. VoxelNet oli üks esimesi taolisi närvivõrke, milles kasutatakse VFE kihi loomisel PointNeti ideed luua tunnuseid täielikult ühendatud võrguga. Lisaks sobivad

punktipilves objektide tuvastamiseks ja lokaliseerimiseks ka Wang ja Jia poolt pakutud Frustrum Convnet[1] ja Lang *et al* poolt pakutud PointPillars[2] närvivõrgud.

Võib öelda, et töös kasutatud Skyhehe123 algoritmi ei ole soovitatav edasi arendada, kuna algoritm ei ole kõige paremini üles ehitatud ning võib kohati jääda arusaamatuks. Lisaks on Githubis mainitud, et algoritmil puudub valideerimise osa, kuid see võib tähendada, et valideerimiseks võis kasutusel olla KITTI valideerimise C++ kood või mõni teine väline koodibaas. Algoritmil puudub ka mitme GPU tugi ja autori hinnangul ei ole selle lisamine triviaalne.

Töös kasutatud Martin Rebase loodud VoxelNeti algoritmile aga tuleks juurde lisada RPN osa, et tegemist oleks Zhou ja Tuzeli artiklis kirjeldatud VoxelNeti algoritmiga. Hetkel on seal vaid VFE kiht ja 3 täielikult ühendatud kihti. Lisaks võib lisada võimaluse *ground truth* piirikastidega pilte salvestada.

Kuna VoxelNeti puhul otsitakse objekti 3D ruumist ning see ruum, millest objekti otsitakse, on suur ja antud töös otsitav objekt on väike, siis on ruumis see osa, kus objekt leidub, samuti väike, seetõttu võib proovida parandada tulemusi, kasutades selleks fokaalset kadu. Fokaalne kadu aitab paremini leida esiplaanil olevaid objekte kui tausta on rohkem kui esiplaanil olevaid objekte [27]. Algoritmile võib juurde lisada ka väärate tõeste ligikaudse arvutamise.

Kindlasti võiks töö tulemusena saadud treenitud mudeleid testida enda kogutud andmete peal ja katsetada päriselus.

[1] https://arxiv.org/abs/1903.01864

[2] https://arxiv.org/abs/1812.05784

5 Kokkuvõte

Käesolevas töös uuriti, kui hästi suudab punktipilvest objektide tuvastamiseks ja lokaliseerimiseks loodud tehisnärvivõrk VoxelNet tuvastada ja lokaliseerida inimesi. Kuna VoxelNeti originaalne lähtekood on salastatud ja seetõttu on selle kordamine keeruline, siis võeti aluseks kaks autode tuvastamiseks ja lokaliseerimiseks loodud VoxelNeti mitteametlikku implementatsiooni, täiendati neid ja püüti nende abil mudeleid treenida.

Kuna punktipilvede töötlemine on arvutusmahukas, siis otsiti kõigepealt närvivõrkude käivitamiseks sobiv keskkond ja loodi valitud keskkonnas vajalikke pakette sisaldavad virtuaalkeskkonnad. Järgmisena täiendati mõlemat VoxelNeti algoritmi. Juurde lisati näiteks valideerimine, kus algoritmi väljastatud regressiooni tulemuseks olevatest 3D deltaväärtustest arvutati piirikastide kattuvus ehk IoU. Täiendati ka piirikastide visualiseerimist 2D ruumis. Lõpuks katsetati erinevaid hüperparameetrite kombinatsioone, et saada mõlema närvivõrguga paremaid tulemusi.

Tulemustest leiti, et aluseks võetud Martin Rebase poolt loodud närvivõrguga saadi inimeste tuvastamisel paremaid tulemusi, kui Githubist aluseks võetud VoxelNeti ühe implementatsiooniga. Lisaks leiti, et töös kasutatud mitteametlike VoxelNeti implementatsioonide käivitamiseks loodavates virtuaalkeskkondades võib tekkida probleeme närvivõrgu tööks vajalike teekide ühildumisel uuemate CUDA platvormidega.

Kokkuvõtteks võib öelda, et töös kasutatud VoxelNeti mitteametlike implementatsioonidega on võimalik punktipilvest inimesi tuvastada. Töö põhieesmärk, milleks oli teha kindlaks, kas VoxelNeti artiklis toodud tulemusi on võimalik taastada või parandada, sai mingil määral täidetud, sest saadi teada, et antud algoritmiga on võimalik inimesi tuvastada.

Kasutatud kirjandus

[1] A. Geiger, P. Lenz, ja R. Urtasun, „Are we ready for autonomous driving? The KITTI vision benchmark suite“, *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI, 2012, lk 3354–3361.

[2] „[ITS] IT terministandardi sõnastik“. [Online]. Saadaval: http://www.eki.ee/dict/its/index.cgi?Q=bounding+box&F=M&C06=et&C10=1. (12.05.2019)

[3] T. S, „Why Uber’s self-driving car killed a pedestrian“, *The Economist*, 29.05.2018.

[4] A. Ng, K. Katanforoosh, Y. B. Mourri, ja M.-A. Valiquette, „Neural Networks and Deep Learning - deeplearning.ai“, *Coursera*. [Online]. Saadaval: https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome. (10.04.2019)

[5] M. El Ansari, R. Lahmyed, ja A. Tremeau, „A Hybrid Pedestrian Detection System based on Visible Images and LIDAR Data“:, *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, Funchal, Madeira, Portugal, 2018, lk 325–334.

[6] T. Luettel, M. Himmelsbach, ja H. Wuensche, „Autonomous Ground Vehicles—Concepts and a Path to the Future“, *Proceedings of the IEEE*, kd 100, nr Special Centennial Issue, lk 1831–1839, mai 2012.

[7] „How LiDAR Technology Enables Autonomous Cars to Operate Safely“, *Velodyne Lidar*, 20.09.2018.

[8] Y. Zhou ja O. Tuzel, „VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection“, *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, 2018, lk 4490–4499.

[9] „PeRL: The Perceptual Robotics Laboratory at the University of Michigan | Ford Campus Vision and Lidar Data Set“, *PeRL: The Perceptual Robotics Laboratory at the University of Michigan*. [Online]. Saadaval: http://robots.engin.umich.edu/SoftwareData/Ford. (11.04.2019)

[10] „nuScenes dataset - Overview“. [Online]. Saadaval: https://www.nuscenes.org/overview. (12.05.2019)

[11] „The KITTI Vision Benchmark Suite“. [Online]. Saadaval: http://www.cvlibs.net/datasets/kitti/. (10.04.2019)

[12] „3D Object Detection Evaluation 2017“. [Online]. Saadaval: http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d. (10.04.2019)

[13] „What is Point cloud | IGI Global“. [Online]. Saadaval: https://www.igi-global.com/dictionary/point-cloud/36879. (11.04.2019)

[14] M. A. Nielsen, „Neural Networks and Deep Learning“, 2015.

[15] E. Inzaugarat, „Understanding Neural Networks: What, How and Why?“, *Towards Data Science,* 30.10.2018. [Online]. Saadaval: https://towardsdatascience.com/understanding-neural-networks-what-how-and-why-18ec703ebd31. (16.05.2019)

[16] „Machine Learning Glossary“, *Google Developers*. [Online]. Saadaval: https://developers.google.com/machine-learning/glossary/. (14.05.2019).

[17] „ProgAlused2N1.pdf“, *Tartu Ülikooli arvutiteaduse instituut*. [Online]. Saadaval: https://courses.cs.ut.ee/LTAT.TK.001/2017_spring/uploads/Main/ProgAlused2N1.pdf. (14.05.2019)

[18] Sirena, „Optimizers for Training Neural Networks“, *Data Driven Investor*, 28.07.2018.

[19] „Overfitting in Machine Learning: What It Is and How to Prevent It“, *EliteDataScience*, 07.09.2017. [Online]. Saadaval: https://elitedatascience.com/overfitting-in-machine-learning. (19.05.2019)

[20] J. Despois, „Memorizing is not learning! — 6 tricks to prevent overfitting in machine learning.“, *Hacker Noon*, 22.03.2018. [Online]. Saadaval: https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42. (11.04.2019)

[21] „Reducing Loss: Learning Rate | Machine Learning Crash Course“, *Google Developers*. [Online]. Saadaval: https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate. (16.05.2019)

[22] „ML Practicum: Image Classification | Machine Learning Practica“, *Google Developers*. [Online]. Saadaval: https://developers.google.com/machine-learning/practica/image-classification/convolutional-neural-networks. (16.05.2019)

[23] A. Ng, K. Katanforoosh, ja Y. B. Mourri, „What is end-to-end deep learning?“, *Coursera*. [Online]. Saadaval: https://www.coursera.org/lecture/machine-learning-projects/what-is-end-to-end-deep-learning-k0Klk. (12.05.2019)

[24] Z. Wang ja K. Jia, „Frustum ConvNet: Sliding Frustums to Aggregate Local Point-Wise Features for Amodal 3D Object Detection“, *arXiv:1903.01864 [cs]*, märts 2019.

[25] „What is a Volume Pixel (Volume Pixel or Voxel)? - Definition from Techopedia“, *Techopedia.com*. [Online]. Saadaval: https://www.techopedia.com/definition/2055/volume-pixel-volume-pixel-or-voxel. (13.04.2019)

[26] „3D voxel Display in matlab“, *Stack Overflow*. [Online]. Saadaval: https://stackoverflow.com/questions/11642426/3d-voxel-display-in-matlab. (13.04.2019)

[27] T.-Y. Lin, P. Goyal, R. Girshick, K. He, ja P. Dollár, „Focal Loss for Dense Object Detection“, *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, lk 2999–3007.

[28] S. Ren, K. He, R. Girshick, ja J. Sun, „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks“, *IEEE Trans. Pattern Anal. Mach. Intell.*, kd 39, nr 6, lk 1137–1149, juuni 2017.

[29] „Colaboratory – Google“. [Online]. Saadaval: https://research.google.com/colaboratory/faq.html. (09.05.2019)

[30] O. Bar-El, „Getting the Most Out of Your Google Colab“, *Ori Bar-El*, 13.10.2018.

[31] T. I. teenused, „IT teenused > Arvutusklaster“. [Online]. Saadaval: https://www.ttu.ee/tugistruktuur/it-teenused/juhendid-1/arvutusklaster/. (11.05.2019)

[32] J. Brownlee, „Understand the Impact of Learning Rate on Model Performance With Deep Learning Neural Networks“, *Machine Learning Mastery*, 24.01.2019.

[33] K. P. Shung, „Accuracy, Precision, Recall or F1?“, *Towards Data Science*, 15-märts-2018. [Online]. Saadaval: https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. (11.05.2019)

[34] „Classification: Precision and Recall | Machine Learning Crash Course“, *Google Developers*. [Online]. Saadaval: https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall. (25.05.2019)

[35] „neural network - How to interpret ‘loss’ and ‘accuracy’ for a machine learning model“, *Stack Overflow*. [Online]. Saadaval: https://stackoverflow.com/questions/34518656/how-to-interpret-loss-and-accuracy-for-a-machine-learning-model. (11.05.2019)

[36] R. Q. Charles, H. Su, M. Kaichun, ja L. J. Guibas, „PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation“, *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, lk 77–85.

Lisa 1 – Juhend Anaconda virtuaalkeskkonna loomiseks enda arvutis

Antud lisas on toodud Anaconda platvormi abil virtuaalkeskkondade loomiseks kasutatud käsud. Kõigepealt tuleb installeerida Anaconda platvorm, mille abil on võimalik luua Anaconda virtuaalkeskkond ja installeerida teeke. Juhised selle jaoks leiab Anaconda[1] leheküljelt.

Järgnevalt on toodud Skyhehe123 poolt loodud VoxelNeti algoritmi käivitamiseks loodud keskkonna jaoks kasutatud käsud. Loodud virtuaalkeskkonnas kasutati Pythonit versiooniga 3.5. Lisaks lisati virtuaalkeskkonda VoxelNeti tööks vajalikud teegid Cython, Shapely, Mayavi, Matplotlib ja Visdom.

```
conda create -n venv-iapb-voxelnet-pytorch-with-visdom python=3.5

conda install pytorch=0.3.1 -c pytorch
conda install -c anaconda cython
conda install --channel https://conda.anaconda.org/menpo opencv3
conda install -c conda-forge shapely
conda install -c menpo mayavi
conda install -c conda-forge matplotlib
conda install -c conda-forge visdom
```

Järgnevalt on toodud Martin Rebase poolt loodud VoxelNeti algoritmi käivitamiseks loodud keskkonna käsud.

```
conda create -n venv-iapb-voxelnet-pytorch-2-with-visdom python=3.5

conda install pytorch=1.0 -c pytorch
conda install matplotlib descartes shapely h5py
conda install -c conda-forge visdom
conda install pytorch torchvision cudatoolkit=9.0 -c pytorch
```

[1] www.anaconda.com