TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Priit Käärd 164061IAIB

3D TEHISNÄRVIVÕRKUDE INTERPRETEERIMINE VOXELNETI NÄITEL

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2020

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: Priit Käärd

18.05.2020

Annotatsioon

Käesoleva töö eesmärgiks on analüüsida ja visualiseerida tunnuseid, mille alusel teevad otsuseid 3D objektituvastuseks arendatud tehisnärvivõrgud. Analüüsi näideteks kasutatakse Apple teadlaste poolt välja töötatud VoxelNet tehisnärvivõrku.

Töö praktiliseks väärtuseks on saada teada, milliste tunnuste alusel otsustab mudel, et teatud asukohas asub objekt. Nende tunnuste alusel, saab manipuleerida mudeli hüperparameetreid ja arhitektuuri eesmärgiga tõsta mudeli ennustamise õigsust ja täpsust.

Töö käigus kirjeldati tehisnärvivõrkude funktsionaalsust ning erinevaid meetodeid ja algoritme keeruliste tehisnärvivõrkude analüüsimiseks.

Töö praktilise tööna muudeti VoxelNet mudeli implementatsiooni nii, et see toetaks PyTorch raamistiku peale ehitatud Captum tarkvarateeki, mis pakub enimtuntuid algoritme tehisnärvivõrkude interpreteerimiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 5 peatükki, 33 joonist, 0 tabelit.

Abstract

Interpreting VoxelNet neural network

The purpose of this paper is to analyze and visualize features, based on which 3D object detection networks make decisions. Analysis are based on VoxelNet, an object detection neural network developed by Apple Inc scientists.

The practical purpose of this work is to find out, which features are used to identify an object. Based on that, model hyperparameters and architecture can be manipulated to increase precision and accuracy of the model.

During the work, author gave an overview of artificial neural networks and described some widely used methods and algorithms for interpreting artificial neural networks.

As a practical work, implementation of the VoxelNet model was modified to support Captum, a library built on top of PyTorch for interpreting neural networks.

The thesis is in Estonian and contains 28 pages of text, 5 chapters, 33 figures, 0 tables.

Lühendite ja mõistete sõnastik

Interpreteerimine	Otsuste kirjeldamine inimesele arusaadaval kujul
Kalle	*Gradient*; Ennustuse vea minimeerimist kirjeldav väärtuste hulk
Konvolutsioon	Matemaatiline operatsioon, mis ahendab andmestruktuure madalamale kujule
Neuron	Väikseim komponent tehisnärvivõrgus
Omistused	*Attributions*; Tehisnärvivõrgu komponentidele arvutatavad mõjutegurid ennustatavale tulemusele
Pakk	*Batch*; Kogum näiteid
Punktipilv	Kolme dimensiooniline andmestruktuur, mis koosneb fikseeritud koordinaatidega punktidest
Pärilevi	Operatsioon, kus tehisnärvivõrgule antakse sisend, mis kantakse läbi peidetud neuronite väljundini.
RPN	*Region Proposal Network*; Tehisnärvivõrk, mis pakub objekti asukohti 2D sisendil
Shapley väärtus	Meeskondlikust mänguteooriast tulenev väärtust, mis kirjeldab võitude ja kaotuste võrdset jaotust mängijate vahel
SVFE	*Stacked Voxel Feature Encoding*; VoxelNeti arhitektuuris kirjeldatud kiht abstraktsete tunnuste õppimiseks sisendandmetest
Tagasilevi	Operatsioon, kus tehisnärvivõrgu väljundist arvutatakse kaldeid läbi peidetud neuronite sisendini.
Tarkvarateek	Valmis olev tarkvara osa, mida saab erinevates programmides kasutada
Voksel	Loogiline plokk punktipilvest

Sisukord

Jooniste loetelu

1 Sissejuhatus

Käeolevas lõputöös on kirjeldatud firmas Apple Inc töötavate teadlaste Yin Zhou ja Oncel Tuzel poolt loodud 3D objektituvastuseks mõeldud tehisnärvivõrgumudelit VoxelNet. Lisaks on täiendatud doktorant Martin Rebase poolt loodud VoxelNet'i implementatsiooni nii, et see toetaks Facebook'i poolt välja töötatud vabavaralist Captum tarkvarateeki, mis pakub erinevaid algoritme keeruliste tehisnärvivõrkude interpreteerimiseks. Viimasena on visualiseeritud 3D punktipilvi ja erinevate mudeli komponentide mõju kasutades vabavaralisi Open3D ja matplotlib tarkvarateeke.

Teema valik tulenes käesoleva töö juhendaja soovist interpreteerida tehisnärvivõrgu mudelit eesmärgiga suurendada mudeli implementatsiooni õigsust ja täpsust. Antud implementatsioon tuvastab korrektselt inimesi ja autosid ainuüksi punktipilvelt, kuid leidub olukordi, kus kõrge tõenäosusega märgitakse autoks näiteks kaljusein. Sellised olukorrad on reaalajasüsteemides turvalisuse aspektist kriitilised. Antud töö põhieesmärgiks on võimaldada interpreteerida selliseid olukordi ja saada aru, mille alusel mudel vahel ootamatuid otsuseid teeb.

Töö esimeses peatükis kirjeldab autor üldiselt tehisnärvivõrkude olemust ja tehnilisi detaile. Lisaks kirjeldab autor detailsemalt VoxelNet tehisnärvivõrgu arhitektuuri ja komponente.

Teises peatükis kirjeldab autor erinevaid tehnikaid, kuidas tänapäeval keerulisi tehisnärvivõrgumudeleid interpreteeritakse ning kirjeldab tuntuid algoritme nende tehnikate kasutamiseks.

Kolmandas peatükis kirjeldab autor kasutatud keskkonda, ettetulnud probleeme ning ja Captumi algoritmite toetamiseks ja visualiseerimiseks tehtud samme. Lisaks on visualiseeritud stseene ning erinevate algoritmidega arvutatud stseeni punktide mõju ennustatavale tulemusele.

Neljandas peatükis visualiseerib autor punktide mõju vigaste ennustuse puhul ning kirjeldab, mida sellelt võib välja lugeda, et mudeli õigsust parandada.

2 Taust

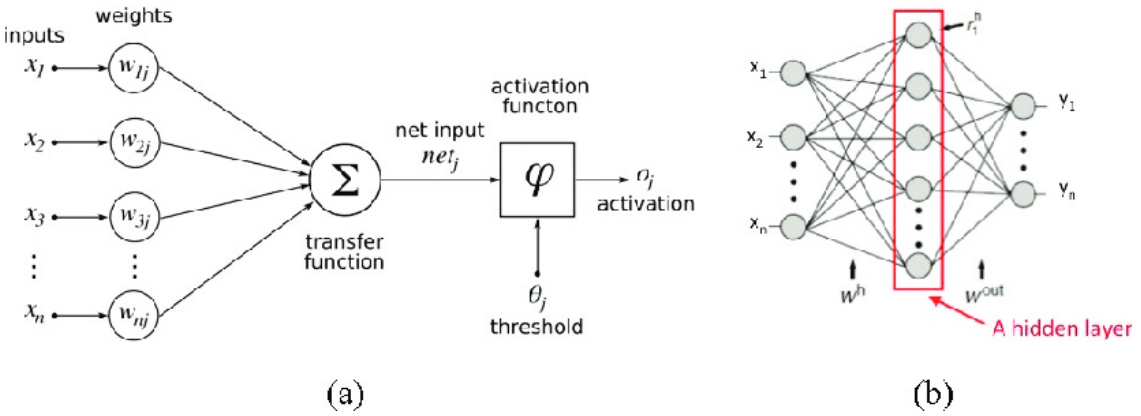
Tänu kiirele tehnoloogia arengule ja teadustöödele on masinõpe viimase 20 aasta jooksul palju populaarsust kogunud. Masinõppe valdkonnast tuntud tehisnärvivõrkude abil lahendatakse tänapäeval keerulisi probleeme, mida varem ei ole suudetud programmikoodiga lahendada. Näiteks inimesed suudavad eristada piltidelt kasse ja koeri, kuid kui oleks see vaja kirjutada käskude jadana, siis oleks see väga keeruline kui mitte võimatu.

Keeruliste probleemide lahendamiseks on vaja ka keerulisi tehisnärvivõrke, mis toob kaasa keerukuse mudeli mõistmisel. Viimase 10 aasta jooksul on avaldatud palju teaduslikke töid tehisnärvivõrkude interpreteerimisese meetoditest ja algoritmidest.

2.1 Tehisnärvivõrgud

Tehisnärvivõrgud on inspireeritud bioloogilistest närvivõrkudest ning suudavad sarnaselt inimestele õppida kogemustest või näidetest ilma, et peaks ette andma konkreetsed juhiseid ülesande lahendamiseks [1].

Tehisnärvivõrgu arhitektuure on erinevaid, kuid kõige tavalisemaks on pärilevivõrk (*feed-forward network*), mis on jaotatud kihtideks ja millest esimene on sisendkiht, viimane väljundkiht ning vahepealsed on peidetud kihid. Joonisel 1 on paremal pool näha täielikult ühendatud pärilevivõrk (FCN). Lisaks pärilevivõrgule on tuntuimad näiteks rekurrentsed ja konvolutsioonilised närvivõrgud, Boltzmanni masina võrgud ja Hopfieldi võrgud. Käesolevas töös on käsitletud peamiselt täielikult ühendatud pärilevi- ja konvolutsioonilisi närvivõrke.

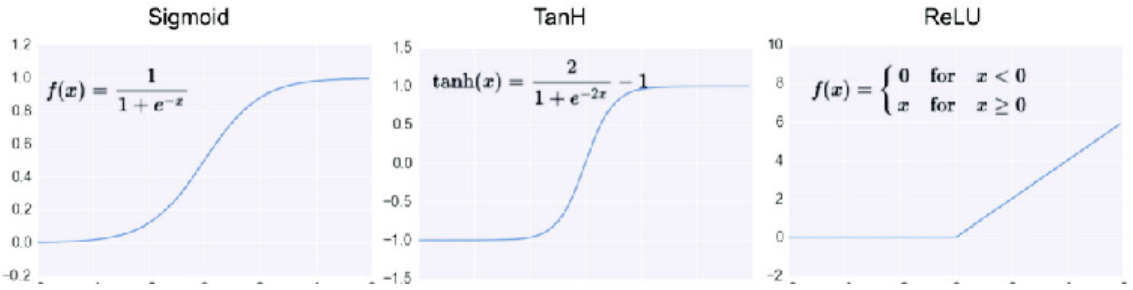


Joonis 1. a) Tehisnärvivõrgu komponendid ja b) mitmekihiline tehisnärvivõrk [2].

Tehisnärvivõrgud õpivad suurel hulgal näidete põhjal. Treenimise ajal antakse võrgule sisendiks ükshaaval või pakkide kaupa näiteid ning arvutatakse pärilevi tulemusest viga. Viga defineeritakse kaofunktsiooni alusel, mis on ette defineeritud võrgu implementeerimisel. Tehisnärvivõrgu treening seisneb arvutatud vea minimeerimisel pärilevi jooksul. Tehisnärvivõrgu treeningu tulemuseks on parameetrite kogum, mida nimetatakse mudeliks. Treenimise ajal viiakse läbi tagasilevi ja arvutatakse kalle (gradient), mis kirjeldab, kuidas peaks mudeli parameetreid muutma, et ennustatava tulemuse viga kahaneks.

2.1.1 Neuron

Tehisnärvivõrgud koosnevad neuronitest. Täielikult ühendatud pärilevivõrgu puhul on selle neuronid on jaotatud kihtideks, erinevate kihtide neuronid on omavahel ühendatud ning annavad edasi signaale järgmise kihi neuronitele. Igal neuronil on üks või rohkem sisendit ja väljundit. Neuroni sisendid on summeeritud ja korrutatud läbi parameetriga, mida nimetatakse kaaluks ja mida tehisnärvivõrk õpib treenimise ajal. Lisaks on summeeritud eelnevale summale juurde järgmine parameeter, mida nimetatakse kallakuks (*bias*). Neuroni väljund sõltub tema mitte-lineaarsest aktivatsioonifunktsioonist, mis saab sisendiks eelnevalt kirjeldatud summa. Joonisel 2 on välja toodud graafikud tuntuimatest aktivatsioonifunktsioonidest - Sigmoid, TanH ja ReLu.



Joonis 2. Aktivatsioonifunktsioonid - Sigmoid, Tanh ja ReLu [3].

2.1.2 Konvolutsioonilised tehisnärvivõrgud

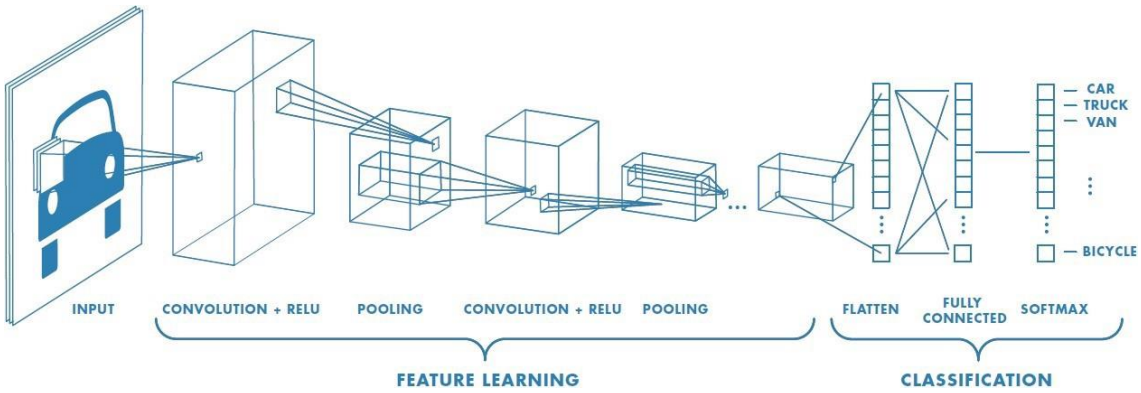
Konvolutsioonilised tehisnärvivõrgud (CNN) on üks tehisnärvivõrkude arhitektuurist, mida kasutatakse põhiliselt visuaalsete andmete analüüsimiseks. Selle nimi viitab matemaatilisele operatsioonile nimega konvolutsioon.

Konvolutsioon on integraalteisendus, mis seab kahele funktsioonile *f* ja *g* vastavusse kolmanda funktsiooni, mida tähistatakse harilikult sümboliga *f***g*. See on integraal funktsioonide *f* ja *g* korrutisest, kus ühe funktsiooni argumenti on peegeldatud ja nihutatud [4].

Visuaalselt kujutatakse piltide puhul konvolutsiooni ette maatriksina, kus iga elemendi asemel on parameetriks kaal, mida tehisnärvivõrk treenimise ajal õpib. Kujuteldav maatriks käib üle pildi ning korrutab läbi iga piksli väärtuse maatriksi elemendi kohal oleva kaaluga. Seejärel võetakse tavaliselt tulemusest maksimaalne väärtus (*max pooling*). Kui maatriks on käinud üle kogu pildi, saadakse tulemuseks madalama resolutsiooniga maatriks. Üldiselt mõeldakse piltide all just värvilisi pilte, mis koosnevad kolmest kihist ehk kanalist – punane, roheline ja sinine.

Konvolutsioonile antakse tehisnärvivõrkude implementeerimisel ette käsitsi parameetreid, milleks on näiteks sisendkanalite arv, väljundkanalite arv, kerneli suurus, sammude suurus (*stride*) ja täienduse suurus (*padding*). Kernel on eelnevalt kirjeldatud maatriks, millega liigutakse üle konvolutsiooni sisendi. Kerneli suuruseks on tema pikkus, laius ja 3D konvolutsiooni puhul ka kõrgus. Sammude suurus ütleb üle mitme elemendi peaks kernel konvolutsiooni sisendi puhul hüppama, vaikimisi on see üks. Täienduse suurus lisab sisendi külgedele null väärtusega elemente, mis võimaldab kernelil liikuda üle sisendi ääre, vaikimisi on padding null.

Joonisel 3 on näiteks toodud kõige tavalisem konvolutsioonilise tehisnärvivõrgu arhitektuur klassifitseerimiseks.



Joonis 3. Näide konvolutsioonilise tehisnärvivõrgu arhitektuurist [5].

2.1.3 Region Proposal Network (RPN)

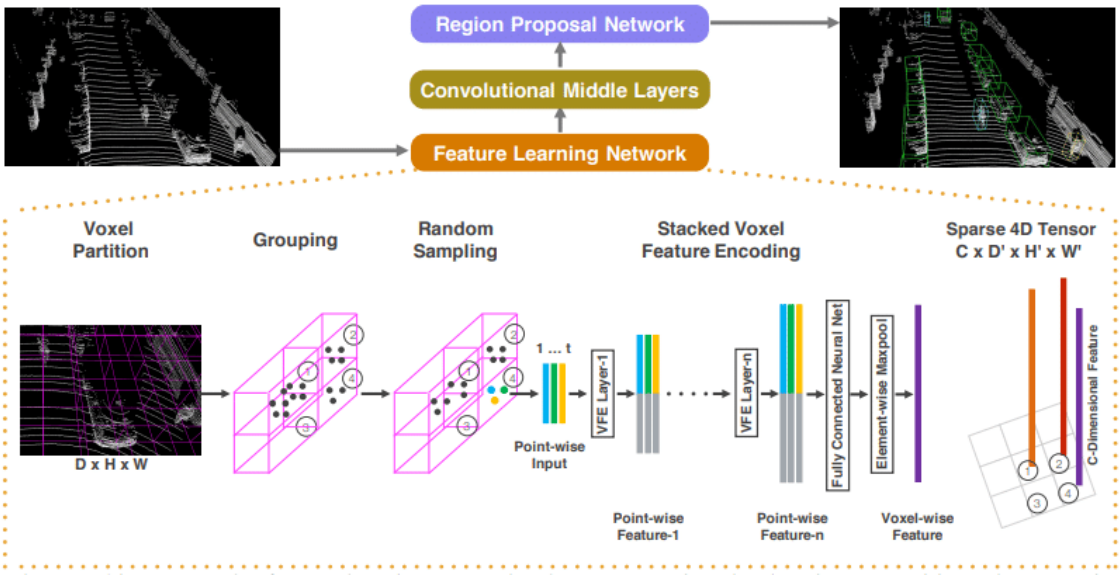
Region Proposal Network (RPN) on täielikult konvolutsiooniline tehisnärvivõrk, mis jagab ühisosa erinevate objektide tuvastuseks loodud tehisnärvivõrkude vahel [6]. RPN saab sisendiks juhuslikus suuruses pildi ning väljastab hulga ristkülikukujulistest objekti regioonidest pildil koos tõenäosustega [6]. Neid ristkülikukujulisi regioone nimetatakse piirkastideks või ankurkastideks, kus ankur viitab piirkasti keskpunktile.

RPN sisend liigub läbi mitme järjestikuse konvolutsioonilise kihi, seejärel jagatakse võrk kaheks ning vahepealne tulemus liigub kahte täielikult ühendatud kihti, millest ühe tulemuseks on piirkasti regressiooni maatriks ja teise tulemuseks on selle sama objekti klassifitseerimismaatriks. Regressiooni maatriksi suuruseks on 4k, kus k tähistab võimalike piirkastide arvu ja 4 väärtust iga piirkasti kohta kirjeldavad 2D pildi puhul vastavalt ankru koordinaate (x, y) ja piirkasti suurused (pikkus ja laius). Klassifitseerimismaatriksi suurus on 2k, mis ütleb kui suure tõenäosusega piirkastis on objekt ja kui suure tõenäosusega ei ole [6].

2.2 VoxelNet

Täpne objektide tuvastus on üks põhiprobleeme autonoomsete sõidukite, kodumasinate ja liitreaalsuse puhul. Selleks, et RPN abil väga hõredast 3D punktipilvest objekte tuvastada, on varasemalt olnud vajalik manuaalselt eraldada tunnuseid sisendandmetest ja need tehisnärvivõrgule sisendiks anda. Apple Inc teadlaste Yin Zhou ja Oncel Tuzel poolt välja töötatud VoxelNet tehisnärvivõrk lahendab selle probleemi ning ühendab tunnuste ja objekti tuvastamise ühte treenitavasse tehisnärvivõrku üldiseks 3D objektide tuvastuseks [7].

VoxelNet koosneb kolmest funktsionaalsest osast – *Stacked Voxel Feature Encoding* (edaspidi SVFE) kiht, keskmistest konvolutsioonilistest kihtidest ja veidi muudetud kujul *Region Proposal* võrgust (edaspidi RPN) [7]. Joonisel 4 on ülevaade kogu VoxelNet tehisnärvivõrgu arhitektuurist.



Joonis 4. Ülevaade VoxelNet'i arhitektuurist [7].

2.2.1 Sisendandmed

Tavalised pildid või videod ei ole objektituvastuse perspektiivis piisavalt täpsed, et neid kasutada reaalajasüsteemides, näiteks autonoomsetel autodel. Selle asemel on kasutusele võetud LiDAR sensorid, mille abil on võimalik ümbruskonnast genereerida hõredaid 3D punktipilvi. Erinevalt kaamerast on võimalik punktipilve abil määrata punkti kaugus sensorist ja asetust teiste punktide suhtes.

VoxelNeti treenimiseks on kasutatud autonoomse juhtimise valdkonnast vabalt kasutatavat KITTI etalonandmestikku, mis sisaldab 7481 pilti ja 3D Velodyne (LiDAR) punktipilve treeninguks ning 7518 testimiseks. Kokku on KITTI andmestikus 80256 märgistatud objekti [8].

2.2.2 SVFE

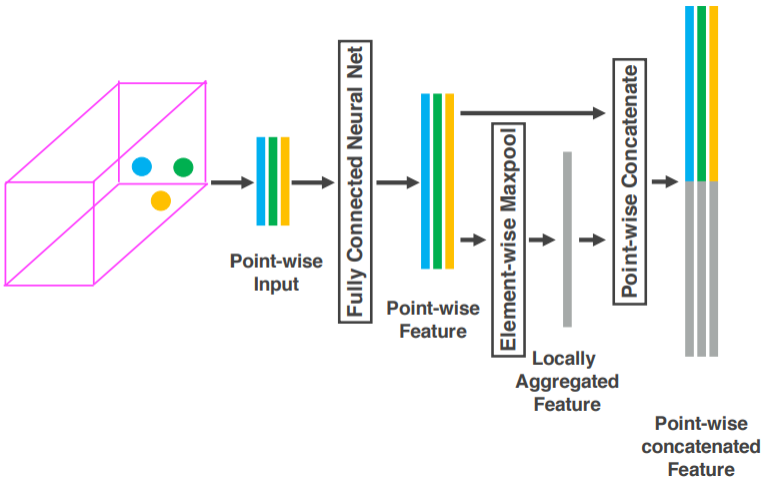
Stacked Voxel feature encoding (SVFE) kiht on põhiline osa VoxelNetist, mis automatiseerib tunnuste eraldamise sisendandmetest. SVFE koosneb neljast etapist – vokseliteks jagamine, grupeerimine, juhuslik valik ja järjestikune vokseli tunnuste kodeering ehk *Stacked Voxel Feature Encoding* kiht.

Vokseliteks jagamise etapis võetakse sisendiks 3D punktipilv ja jagatakse see eelnevalt fikseeritud ühesuurusteks blokkideks, mida nimetatakse vokseliteks. Grupeerimise etapis kogutakse kokku igasse vokselisse kuuluvad punktid. Kuna LiDAR punktipilved hõredad ja punktide tihedus ei ole stabiilne, siis ühes vokselis paiknevate punktide arv varieerub

ja see võib tekitada objektituvastusel probleeme. LiDAR punktipilved koosnevad umbes kümnest tuhandest punktist, mis võib viia ebavajaliku ressursimahuni. Neid probleeme lahendab juhusliku valiku kiht [7].

Juhusliku valiku kihis valitakse sõltuvalt ülesandele arv T ja võetakse kõik vokselid, milles on rohkem kui T punkti. Voksel haaval, valitakse nendest punktidest suvalised T punkti eesmärgiga vähendada vajalikku mälumahtu ja ühtlustada ebastabiilset punktide tihedust punktipilves. Viimasena valitakse iga vokseli kohta 7 tunnust: x, y, z koordinaadid punktipilves, punkti peegeldus r ja x', y', z' relatiivsed koordinaadid vokseli suhtes. Iga punkti tunnuste vektor antakse edasi täielikult ühendatud võrku (FCN), mis koosneb ühest lineaarsest kihist, paki normaliseerimise kihist ja ReLu kihist. Tulemuseks saadakse punktipõhised tunnused. Edasi võetakse iga punkti kohta maksimaalne tunnus, mida nimetatakse lokaalselt agregeeritud tunnuseks. Viimasena ühendatakse kokku punktipõhised tunnused ja lokaalselt agregeeritud tunnused ning saadakse punktipõhine ühendatud tunnus. VFE tulemuseks on hõre 4D tensor kujul C, D, H, W, kus C on õpitud abstraktsete tunnuste arv ning D, H, W on vokselite koordinaadid ruudustikus [7].

Joonisel 5 on visuaalselt näidatud ühe VFE kihi funktsionaalsus. Tavaliselt kasutatakse SVFE kihis järjestikuliselt mitut VFE kihti.



Joonis 5. Voxel Feature Encoding kiht [7].

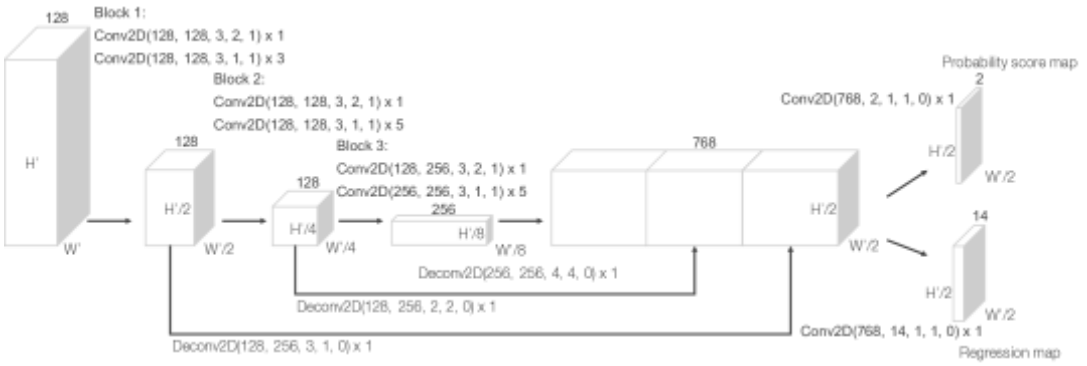
2.2.3 Keskmised konvolutsioonilised kihid

Keskmistes konvolutsioonilistes kihtides kasutatakse iga konvolutsioonilise kihi kohta järjestikuseid 3D konvolutsioonilisi (Conv3D), paki normaliseerimise (BatchNorm3D) ja

ReLu kihte. Need agregeerivad vokselipõhiseid tunnuseid ja õpivad ära tundma nendest mustreid. Iga konvolutsiooniline kiht vaatab tunnuseid üldisemalt kui eelmine [7].

2.2.4 RPN

VoxelNeti muudetud kujul *Region Proposal Network* (RPN) sisendiks on keskmistest konvolutsioonilistest kihtidest õpitud abstraktsete tunnuste maatriks (Joonis 6). RPN koosneb kolmest täielikust konvolutsioonilisest kihist. Iga kihi tulemusena on tunnuste maatriks 2 korda väiksem. Järgmiseks suurendatakse kõik 3 konvolutsioonilise kihi väljundit üheks fikseeritud suuruseks ja ühendatakse kokku. Tulemuseks on kõrge resolutsiooniga tunnuste maatriks. Viimasena on tulemus viidud soovitavale kujule – regressioonimaatriksiks ja vastavate piirkastide tõenäosuste maatriksiks [7].



Joonis 6. Muudetud kujul Region Proposal Network [7].

3 Tehisnärvivõrkude interpreteerimine

Captum[1] on vabavaraline tarkvarateek, mis on loodud tehisnärvivõrkude interpreteerimiseks. Captum on arendatud Facebooki[2] poolt ja on sügavalt integreeritud töötama vaid PyTorch[3] masinõppe raamistikuga [9]. Captum on töö kirjutamise hetkel beeta versioonil ning võib tulevikus kaasa tuua mitteühilduvaid muudatusi.

Captum sisaldab endas erinevaid tänapäevaseid algoritme tehisnärvivõrgu mudelite interpreteerimiseks ning need on jagatud kolme põhilisse kategooriasse - üldised omistused (*general attributions*), kihtide omistused (*layer attributions*) ja neuronite omistused (*neuron attributions*). Captum sisaldab juba implementeeritud algoritme, mida on võimalik kasutada ilma nende tehnilisse implementatsiooni süvenemata. Järgnevalt on välja toodud algoritmid, mida käesoleva töö raames on katsetatud ja millisel põhimõttel need töötavad.

Erinevad algoritmid omistuste leidmiseks on jagatud peamiselt kahte kategooriasse – kaldepõhised (*gradient-based*) ja segaduspõhised (*perturbation-based*) algoritmid. Kaldepõhised algoritmid kasutavad otseselt või kaudselt mudeli kalde arvutuse valemeid. Segadusepõhised algoritmid arvutavad omistusväärtuseid mõõtes müra mõju võrgu sisendis ennustatavale tulemusele [10].

3.1 Üldised omistused

Üldised omistused mõõdavad iga sisendi tunnuse olulisust mudeli väljundile [11].

3.1.1 Integrated Gradients

Integrated Gradients on 2017 aastal M. Sundararajan, A. Taly ja Q. Yan poolt välja pakutud aksiomaatiline algoritm tehisnärvivõrkude interpreteerimiseks ning see on üks populaarsemaid meetodeid tänaseni. Integrated gradients võimaldas ühe esimese algoritmina leida üldisi omistusi vältides muudatusi mudeli implementatsioonile,

[1] https://captum.ai/

[2] https://www.facebook.com/

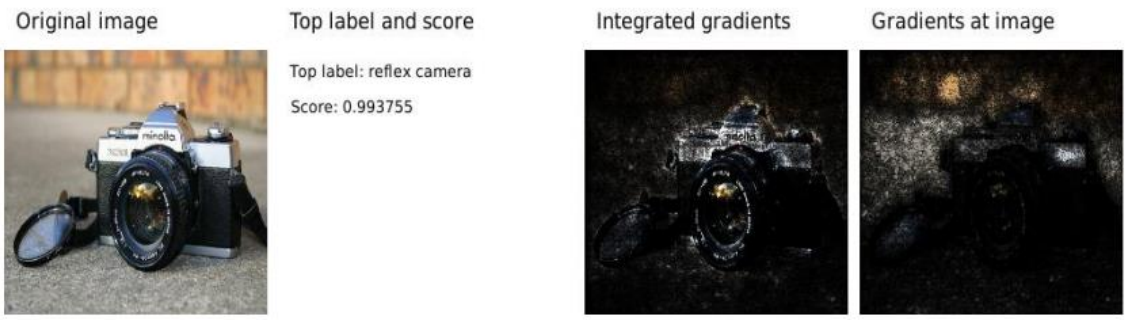
[3] https://pytorch.org/

kasutades vaid kalde operaatori funktsioone [12]. Integrated gradients on tagasilevipõhine (*back-propagation based*) meetod, mis tähendab, et omistused arvutatakse sisendandmetele ühe pärilevi ja tagasilevi jooksul [13].

Integrated Gradients põhineb kahel aksioomil – tundlikkus ja implementatsiooni muutus. Tundlikkus on rahuldatud, kui iga tunnuse või algtaseme muutuse puhul muutub ka väljund. Implementatsiooni muutus on rahuldatud, kui kahel funktsionaalselt identsel võrgul on samad omistused [12].

Integrated Gradients meetodi kasutamiseks tuleb valida hea algtase (*baseline*) [12]. Käesoleva töö puhul sobib algtasemeks null, sest see tähistab puuduvaid andmeid punktipilve mõistes. Seejärel saab lähendada Integrated Gradients valemi integraali summeerides kaldeid valitud algtaseme ja sisendandmete väärtuste vahelt. Kuna Integrated Gradients valem sisaldab integraali ning integraali arvutamine on ressursimahukas, siis kasutatakse selle puhul Riemmani lähendamist. Selleks tuleks valida arv, mitme sammuga integraali lähendada. Tavaliselt sobib piisavaks täpsuse saavutamiseks 20 kuni 300 sammust [12].

Joonisel 7 on näide Integrated Gradients kasutamisest klassifitseeriva mudeli peal.



Joonis 7. Integrated Gradients tulemuse võrdlus pildi kallete suhtes [12].

3.1.2 Soovituskaardid

Soovituskaardid (*Saliency maps*) on 2014 aastal K. Simonyan, A. Vedaldi ja A. Zissermani poolt välja pakutud kaldepõhine algoritm tehisnärvivõrkude interpreteerimiseks [14]. Piltide puhul vaatab soovituskaartide algoritm iga piksli kohta erinevaid tunnuseid nagu värvide erksust, orientatsiooni ja liikumist. Soovituskaartide algoritmi on võimalik kasutada ka piltide segmenteerimiseks selle asemel, et treenida eraldi mudel segmenteerimiseks [14]. Tagasilevi jooksul pannakse kokku kalde väärtuste

järgi soovituskaart. Soovituskaardil erksamad alad tähistavad kõige rohkem ennustust mõjutavaid ja kõigi kolme aspekti poolt esile jäänud alasid.

Joonisel 8 on näide soovituskaartide kasutamisest klassifitseeriva mudeli peal.

Joonis 8. Soovituskaardid klassifitseeriva tehisnärvivõrgu peal [14].

3.1.3 DeepLIFT

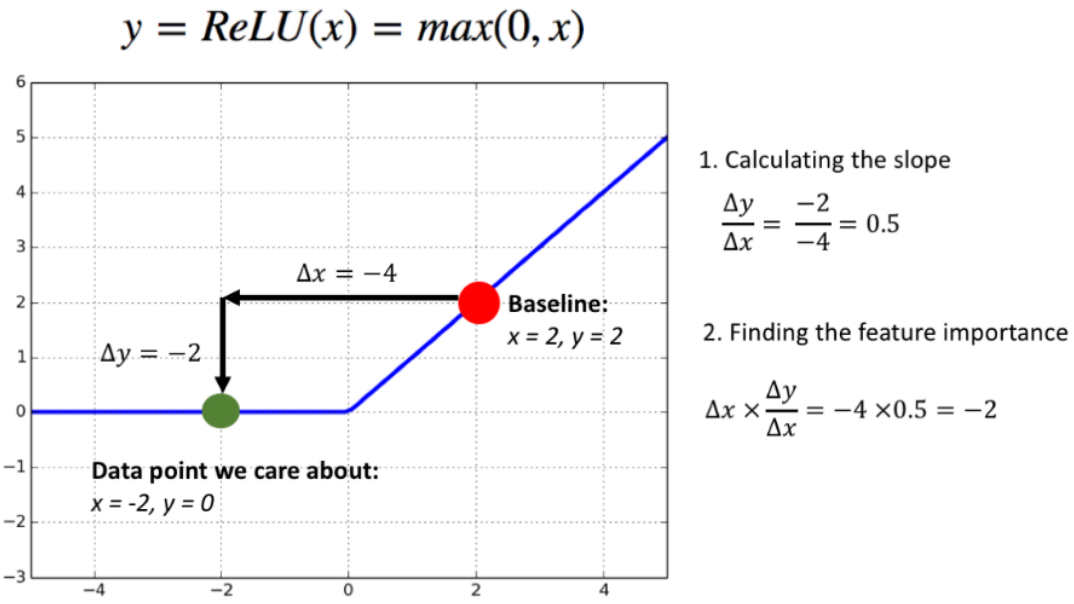
DeepLIFT on järjekordne kaldepõhine algoritm. Kui Integrated Gradients kirjeldab kallet, kuidas väljund muutub kui sisend muutub, siis DeepLIFT kirjeldab kallet, kuidas väljund muutub kui sisend erineb algtasemest [15] (Joonis 9). Käesolevas töös sobib algtasemeks null. DeepLIFTi kontseptsioon on lihtne, kuid seda on keerulisem implementeerida, kuna see muudab kalde arvutamise valemit. Tänu Captumile on DeepLIFTi Rescale reegel juba varasemalt implementeeritud.

Rescale reegel kasutab tehisnärvivõrgus vaid ühe sisendiga operatsioone nagu ReLU, TanH või Sigmoid [16]. Alustades viib algoritm läbi pärilevi, kus sisendiks antakse etteantud algtaseme väärtused, käesoleva töö puhul nullid. Järgnevalt antakse võrgule sisendiks algoritmile etteantud sisend, käesoleva töö puhul punktipilv, ja mõõdetakse uuesti neuronite aktivatsioone etteantud väljundi suhtes. Võrreldes mõlemaid väärtuseid annab algoritm tulemuseks omistusväärtused sisendi kujul [13].

Teiseks pakutud reegliks on RevealCancel reegel, mida Captumi poolt ei ole implementeeritud. Kuna DeepLIFT tugineb kalde arvutustel, siis võib tekkida olukordi, kus nulli lähedaste väärtuste puhul ei ole see piisavalt täpne [16]. Erinevalt Rescale

reeglist vaatab RevealCancel eraldi positiivset ja negatiivset mõju antud ennustuse suhtes, mis lahendab eelneva probleemi [16].

Captum pakub ka DeepLiftShap algoritmi, mis arvutab Shapley väärtused igale väiksemale komponendile. See algoritm eeldab, et kõik komponendid on lineaarsed, mis käesoleva töö puhul ei ole.



Joonis 9. DeepLIFT tunnuse olulisus algtaseme suhtes [17].

3.1.4 Gradient SHAP

SHAP[1] ehk Shapley Additive exPlanations on 2017 aastal S.M. Lundberg ja S. Lee poolt mänguteooriast tuletatud universaalne raamistik tehisnärvivõrgumudelite interpreteerimiseks [18].

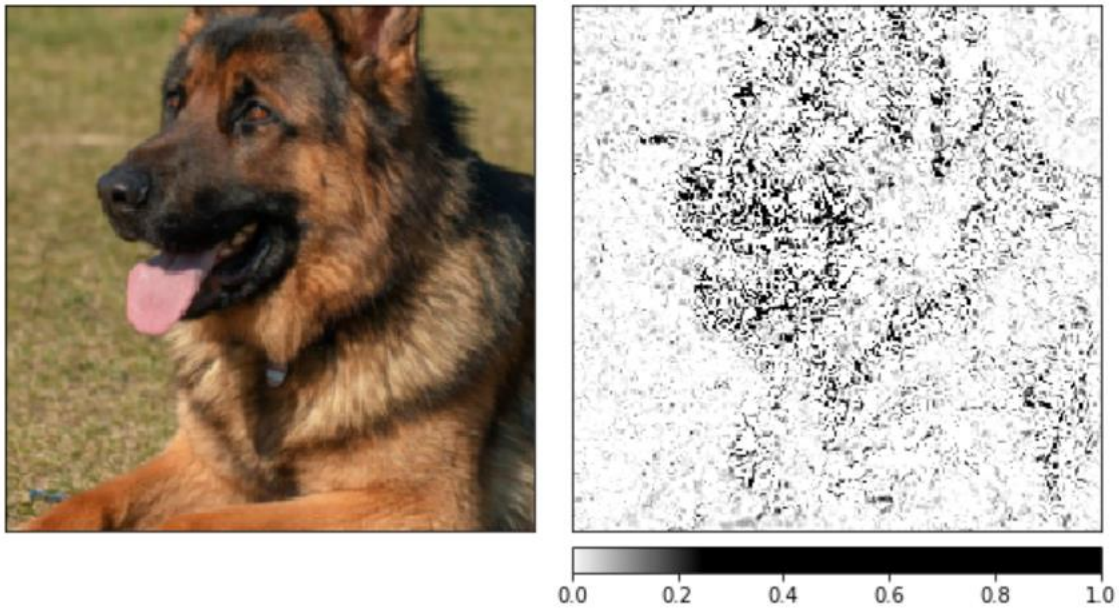
Shapley väärtuse valem tuleb meeskondlikust mänguteooriast ja see kirjeldab võrdset võitude ja kaotuste jaotust erinevate faktorite või mängijate vahel. Shapely väärtuse eesmärk on võidud ja kaotused jagada mängijate vahel sõltuvalt individuaalse mängija panusest. Iga mängija peaks meeskonnas võitma vähemalt nii palju või rohkem kui ta osaleks mängus üksinda [19].

[1] https://github.com/slundberg/shap#deep-learning-example-with-gradientexplainer-tensorflowkeraspytorch-models

SHAP määrab igale tunnusele olulisuse väärtuse etteantud ennustuse suhtes [18]. Shapley väärtuse mõistes on mängijateks iga sisendiks olev tunnus ja auhinnaks ennustuse väärtus – väärtused SHAP selgitusmudelilt [20]. Täpne SHAP väärtuste arvutamine on keerukas, kuid sarnaselt eelnevatele algoritmidele saab ka neid lähendada [18].

SHAP väärtuste valem sõltub eelnevalt mainitud selgitusmudelist. Captum implementeerib erinevaid SHAP algoritme, kuid käesolevas töös on katsetatud GradientSHAP algoritmi, mis põhineb kaldepõhisel selgitusel.

Joonisel 10 on näide Gradient SHAP algoritmi kasutamisest klassifitseeriva mudeli peal.



Joonis 10. Gradient SHAP [21].

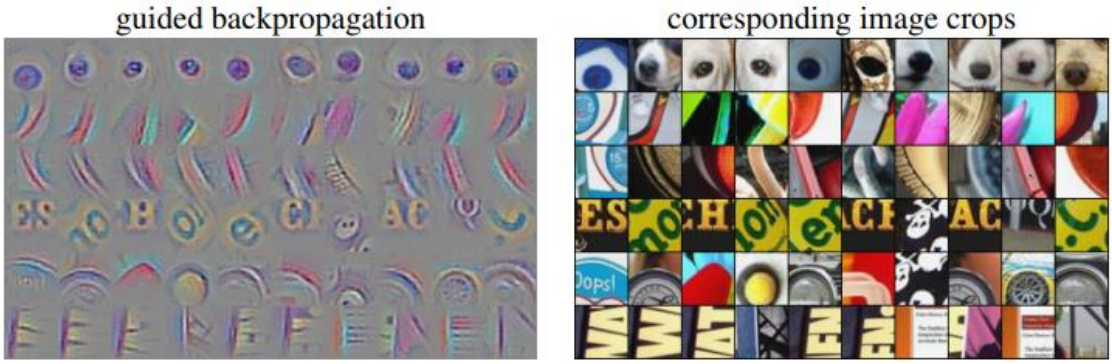
3.1.5 Input X Gradient

Input X Gradient on laiendus eelnevalt kirjeldatud soovituskaartidele. Input X Gradient arvutab väljundi kalded sisendi suhtes ning korrutab läbi sisendi väärtustega. Lineaarse mudeli puhul on kalded lihtsalt iga sisendi kordajad. Kalde ja sisendi korrutis vastab lineaarse mudeli korral sisendi olulisusele antud väljundi jaoks [22].

3.1.6 Juhatatud tagasilevi

Juhatatud tagasilevi ehk *Guided Backpropagation* arvutab etteantud väljundi kalded sisendi suhtes, kuid tagasilevi ajal levivad tagasi vaid mitte-negatiivsed kalded [22]. Juhatatud tagasilevi puhul kasutatakse sisendi kallete peal ReLU funktsiooni [22], ehk negatiivsed kalded nullitakse ära.

Joonisel 11 on näidatud, milliseid tunnuseid vaatab klassifitseeriva mudeli ImageNet sisendist kuues konvolutsiooniline kiht.



Joonis 11. Juhatatud tagasilevi ImageNet kuuenda konvolutsioonilise kihi näitel [23].

3.1.7 Dekonvolutsioon

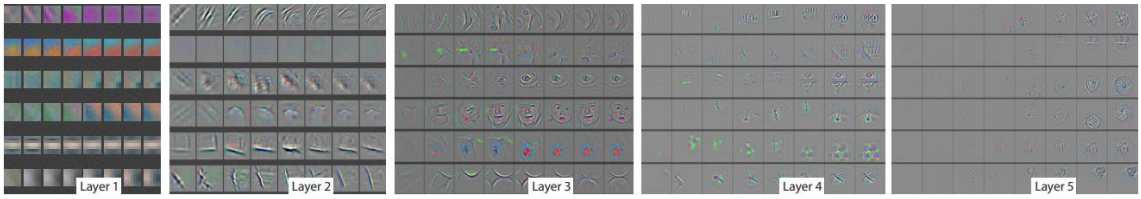
Kirjeldatud algoritmidest üks vanimaid on dekonvolutsioon. Interpreteerimiseks pakuti dekonvolutsioon välja 2013. aastal Euroopa masinnägemise konverentsil ECCV 2014 M. D. Zeiler ja R. Fergus poolt [24]. Dekonvolutsiooni puhul on tegemist tavaliste konvolutsiooniliste, filtreerimis- ja ahenduskihtidega, kuid töötades tagurpidi [25]. Võrgu interpreteerimiseks pannakse iga konvolutsioonilise kihi kõrvale dekonvolutsiooni kiht või vastav vastupidine komponent. Nendest koosneb vastupidise arhitektuuriga tehisnärvivõrk, mille sisendiks on interpreteeritava võrgu valitud väljund ning väljundiks on omistusväärtused.

Ahenduskihi (*pooling*) vastandiks on laienduskiht (*unpooling*). Ahenduskihi otsene tagurpidi pööramine pole võimalik. Selleks hoitakse meeles maksimaalsete väärtuste asukohti iga konvolutsioonilise kihi tunnuste maatriksil ja pannakse dekonvolutsioonil need sisendiks [25]. Ülejäänud väärtused seatakse nulliks.

Teadusartiklis analüüsitud konvolutsiooniline tehisnärvivõrk kasutab ReLU aktivatsioonifunktsiooni, mis muudab kõik tunnuste maatriksid positiivseteks. Sellisel juhul ReLU aktivatsioonifunktsiooni tagurpidi ei pöörata vaid kasutatakse uuesti tavalist ReLU aktivatsioonifunktsiooni dekonvolutsioonil [25].

Konvolutsioonilised tehisnärvivõrgud kasutavad õpitud filtreid, mida kasutatakse konvolutsiooni tunnuste maatriksi peal. Selle tagurpidi pööramiseks transponeeritakse need samad filtrid ja kasutatakse seda parandatud tunnuste maatriksi peal.

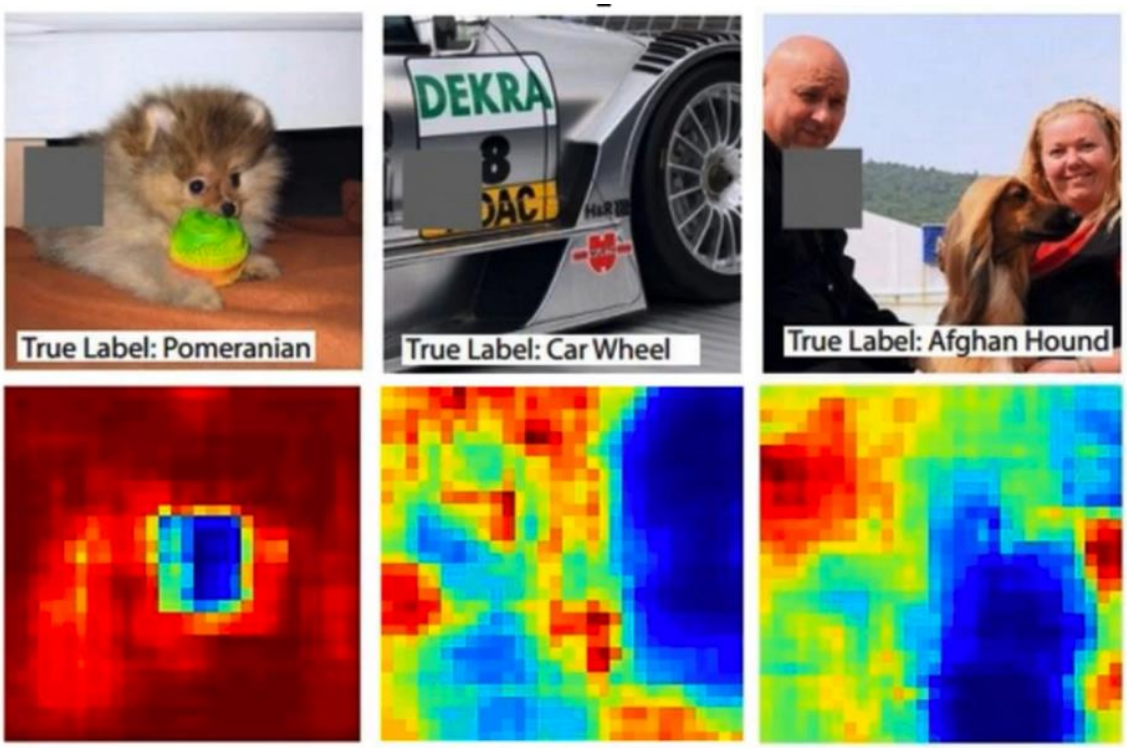
Joonisel 12 on näha, kuidas sügavamad kihid vaatavad klassifitseeriva mudeli sisendit oluliselt detailsemalt kui eelnevad kihid.



Joonis 12. Dekonvolutsioon. Kuidas erinevate kihtide tunnused muutuvad detailsemaks sügavamatesse kihtidesse minnes [25].

3.1.8 Oklusioon

Oklusioon (*Occlusion*) on segaduspõhine (*perturbation-based*) viis omistuse arvutamiseks. Täpsemalt saab oklusiooni abil leida alad sisendandmetest, mis on kriitilised korrektse tulemuse ennustamiseks. Üldiselt kasutatakse oklusiooni piltide interpreteerimiseks, kuid kuna punktipilved sarnanevad oma olemuselt piltidele, siis on proovitud kasutada oklusiooni ka käesoleva töö mudelil. Oklusiooni puhul lohistatakse üle sisendandmete algtaseme väärtustega aken. Oklusiooni algoritm võtab rohkem aega kui eelnevad, kuna igal akna positsioonil tuleb leida ennustus. Tulemuseks on omistusväärtused, kus kõrgemad väärtused tähendavad kriitilisi tunnuseid sisendandmetes ja madalamad ebaolulisi [25] (Joonis 13).



Joonis 13. Oklusioon pilte klassifitseeriva mudeli peal [26],

3.2 Kihtide omistused

Kihtide omistused (*Layer attributions*) mõõdavad etteantud kihis iga neuroni olulisust mudeli väljundile [11]. Kui üldised omistused mõõtsid ainult sisendi tunnuste olulisust ennustatava tulemuse saavutamiseks, siis kihtide omistused võimaldavad analüüsida mudelit sügavamalt ning uurida kiht haaval, millised neuronid vaadeldavas kihis on olulised ja vajadusel muudatusi teha mudeli implementatsioonile.

Kõik Captumi poolt implementeeritud algoritmid arvutavad omistusväärtuseid vaadeldava kihi väljundi kujul.

3.2.1 Kihi aktivatsioon

Kihi aktivatsioon (*Layer Activation*) on lihtne viis kihi omistuste arvutamiseks, tagastades vaid iga antud kihi neuroni aktivatsioon [22]. Neuroni aktivatsioon sõltub temale antud aktivatsioonifunktsioonist. Antud algoritmi on lihtne implementeerida, kuid see ei pruugi anda täpseid tulemusi. Näiteks ReLu aktivatsioonifunktsiooni tulemus on null, kui funktsiooni sisend on null või sisendi väärtus, kui sisend on suurem nullist. Sellisel juhul on võimalik neuroni aktivatsiooni mõõta vaid positiivse poole pealt, kuna neuroni aktivatsioon ei ole kunagi negatiivne.

3.2.2 Kihi kalle X aktivatsioon

Kihi kalle X aktivatsioon (*Layer Gradient X Activation*) korrutab neuronite kaupa kihi kalde ja aktivatsiooni valitud eesmärgi suhtes.

3.2.3 Kihi GradCam

GradCAM on konvolutsioonilistele tehisnärvivõrkudele mõeldud omistusväärtuste leidmise algoritm ning seda kasutatakse tavaliselt viimasel konvolutsioonilisel kihil [22] (Joonis 14). GradCAM arvutab valitud väljundi kalded valitud kihi suhtes, võtab iga kanali kohta keskmise ning korrutab läbi keskmised kalded kihtide aktivatsiooniga [22]. GradCAM on loodud erinevate neuronite olulisuse määramiseks, kuid seda kasutatakse ka üldiste omistuste arvutamiseks, kuna konvolutsioonilised kihid kattuvad ruumiliselt sisendiks oleva pildiga [22].

Joonis 14. GradCAM klassifitseeriva tehisnärvivõrgu peal [27].

3.2.4 Kihi DeepLIFT

Kihi DeepLIFT (*LayerDeepLift*) on analoog eelnevalt kirjeldatud DeepLIFT üldiste omistuste algoritmile, kuid see on arvutatud valitud kihi suhtes [22].

3.2.5 Kihi GradientShap

Kihi GradientShap (*LayerGradientShap*) on samuti analoog eelnevalt kirjeldatud Gradient SHAP üldiste omistuste algoritmile. Sisendi asemel arvutatakse kalde väärtus valitud kihi suhtes. SHAP väärtused on lähendatud kallete ja kihi aktivatsioonide erinevuse korrutisele algtaseme suhtes [22].

3.3 Neuroni omistused

Neuroni omistamised (*Neuron attributions*) mõõdavad iga neuroni sisendi tunnuse olulisust antud neuronis mudeli väljundile [11]. Lihtsamate mudelite jaoks võib see pakkuda rohkem infot, kuid keeruliste mudelit puhul see piisavalt infot ei anna, seega on jäetud antud töös neuroni omistused vahele.

4 Tehniline lahendus

Käesolevas peatükis kirjeldab autor analüüsimiseks vajalikke praktilisi samme, tehnoloogilisi valikuid ning esinenud probleeme.

Analüüsitava tehisnärvivõrgu aluseks sai võetud doktorant Martin Rebase poolt loodud VoxelNeti implementatsioon. Implementatsioon on arendatud nii autode kui ka inimeste tuvastamiseks. Mõlema puhul kasutab see treenimiseks ja valideerimiseks vabalt kasutatavat KITTI etalonandmestikku.

4.1 Keskkond

Antud implementatsioon on implementeeritud Python[1] programmeerimiskeeles, PyTorch[2] masinõppe raamistikule ning on valmis töötama ka CUDA[3] platvormil. CUDA on Nvidia[4] poolt loodud paralleelrehkenduste platvorm, mida kasutatakse tihti masinõppe ülesannete puhul kiiremateks rehkendusteks. CUDA jagab rehkendused laiali erinevatele graafikakaardi (GPU) tuumadele, mida on palju rohkem kui protsessoril (CPU) ning seetõttu võimaldab rehkendusi läbi viia oluliselt kiiremini kui protsessori peal. Masinõppe raamistikud nagu PyTorch on disainitud läbi viima paralleelseid rehkendusi.

Programmeerimisliideseks kasutas autor PyCharm keskkonda, kus saab lihtsalt kasutada kaudset interpretaatorit, antud juhul Google Cloud virtuaalmasinat.

Peamiseks probleemiks on antud implementatsiooni puhul vajalik mälumaht. Käesolevas töös kasutas autor nii 6GB Nvidia GeForce GTX 1060 graafikakaardi ja 16GB muutmäluga sülearvutit kui ka 16GB Nvidia Tesla T4 graafikakaardi ja 13GB muutmäluga Google Cloud virtuaalmasinat. Ka Google Cloud keskkonnas piisas olemasolevatest ressurssidest vaid madalate sammudega integraalide lähendamisteks katsetatud algoritmides.

[1] https://www.python.org/

[2] https://pytorch.org/

[3] https://developer.nvidia.com/cuda-toolkit

[4] https://www.nvidia.com/en-us/

Analüüsiks sobiliku mudeli treenimine kestab päevi, seega aluseks on võetud varasemalt 149 epohhini autode tuvastamiseks treenitud mudel.

Lihtsamaks arenduskeskkonna haldamiseks lõi autor tarkvara sõltuvuste kirjelduse faili (*requirements.txt*), mis fikseerib erinevad tarkvarateekide versioonid ning võimaldab need ühe käsuga igal implementatsiooni arendajal alla laadida.

4.2 3D visualiseerimisvõimekuse loomine

Visualiseerimiseks oli implementatsioonis kasutusel juba Facebooki poolt arendatud Visdom[1] rakendus, mille abil visualiseeriti vaid kaofunktsiooni kahanemist treenimise ajal. Valideerimise tulemuseks saadud punktipilved salvestati faili ning visualiseeriti hiljem eraldiseisva tarkvaraga. Esimeseks probleemiks tekkis tarkvara puudumine antud kujul punktipilve kuvamiseks Windows operatsioonisüsteemil. Linux operatsioonisüsteemidel oli võimalik avada selliseid faile PCLViewer[2] rakendusega.

Punktipilvede visualiseerimiseks lisas autor projekti avatud lähtekoodiga Open3D tarkvarateegi, mida kasutatakse 3D andmete visualiseerimiseks. Open3D sisaldab endas ka erinevaid algoritme 3D andmete manipuleerimiseks, kuid käesolevas töös neid ei kasutatud. Mitme punktipilve visualiseerimiseks lisati projekti abifunktsioonid, millega on võimalik paralleelselt vaadelda mitut punktipilve. Vaikimisi on Open3D visualizeri aknad blokeerivad ning ei võimalda mitut geomeetriat korraga avada.

Esimeseks sammuks visualiseeriti kasutaja poolt valitud stseeni punktipilv. Selleks tuli KITTI andmestiku 3D punktipilvest võtta välja punktide x, y, z koordinaadid, need nihutada LiDAR positsioonilt kaamera positsioonile kasutades andmestikust kalibreerimise väärtuseid ja seejärel punktide värvuseks anda LiDAR laseri peegelduse tugevus. Punktide nihutamiseks oli vajalik funktsioon juba loodud.

Teise sammuna visualiseeriti mudeli sisend. Sisendi töötlemiseks oli varasemalt implementeeritud PyTorchi andmelaadija (DataLoader) klass. Andmelaadija jagas punktid pilves vokseliteks ning andis igale vokselis olevale juhuslikult valitud punktile

[1] https://ai.facebook.com/tools/visdom/

[2] https://www.pclviewer.com/

seitse eelnevalt mainitud tunnust – koordinaadid x, y, z punktipilves, peegeldus r ja relatiivsed koordinaadid x', y', z' vokseli suhtes. Visualiseerimiseks on kõik vokselis olevate punktide peegeldused summeeritud, normaliseeritud (1) skaalale nullist üheni ja visualiseeritud.

$$X = \frac{\mathrm{x}}{\max\{\,|\max\{\,X\,\}|,|\min\{\,X\,\}|\}} \quad (1)$$

Kolmanda sammuna visualiseeriti mudeli väljundit. Mudeli väljundiks on tõenäosuste maatriks ja regressiooni maatriks, mida otse ei ole võimalik visualiseerida kuna need ei ole visualiseeritaval kujul ja sisaldavad nii nõrga tõenäosusega piirkaste kui ka korduvalt kattuvaid piirkaste. Selleks kasutatakse mudeli valideerimisel *non-max suppression* meetodit, mille tulemusena valitakse visuaalselt tugevama tõenäosusega piirkastid ning eemaldatakse kõik liigselt üle kattuvad piirkastid. Ülekattuvus (*intesection over union* ehk *IoU*) on protsentuaalne hüperparameeter, mida saab jooksvalt konfiguratsioonis määrata. Eelnevalt implementeeritud validatsiooni tulemuseks antakse piirkastid, kus asuvad kõige suurema tõenäosusega objektid. Visualiseerimiseks pandi kokku mudeli sisend ja piirkastid validatsiooni tulemusetest.

Edasi leiti tõenäosuste maatriksilt suurima tõenäosuse indeks, mille suhtes hakati erinevate algoritmide puhul omistusväärtuseid leidma.

4.3 Omistused

4.3.1 Üldised omistused

Esimeseks algoritmiks valiti Integrated Gradients. Selle puhul oli vaja captumi jaoks muuta mudeli edasilevi funktsiooni, et sisendiks oleks vaid üks tensor. Tensor on ühe või mitme dimensiooniline andmestruktuur PyTorch raamistikus, mida PyTorch oskab kasutada paralleelrehkenduste läbiviimiseks. Eraldi tuli kaasa anda koordinaatide puhvri tensor, mis hoiab vokseli indeksi alusel meeles tema koordinaate punktipilvel. Algoritmile tuli ette anda eesmärk (*target*), milleks valis autor suurima tõenäosusega vokseli. Hiljem loodi eesmärgi valimiseks eraldi funktsioon, mis võimaldab ühe stseeni raames erinevaid objekte valida sorteeritud tõenäosuse järgi. Kuna antud implementatsioon toetab antud hetkel vaid ühte pakki (*batch*) korraga, siis tuleb ka Integrated Gradients algoritmile see eraldi ette öelda. Viimaseks kasutati integraali

lähendamiseks vaikimisi pakutud 50 sammu, mis algoritmi autorite sõnul sobivas vahemikus (20 - 300).

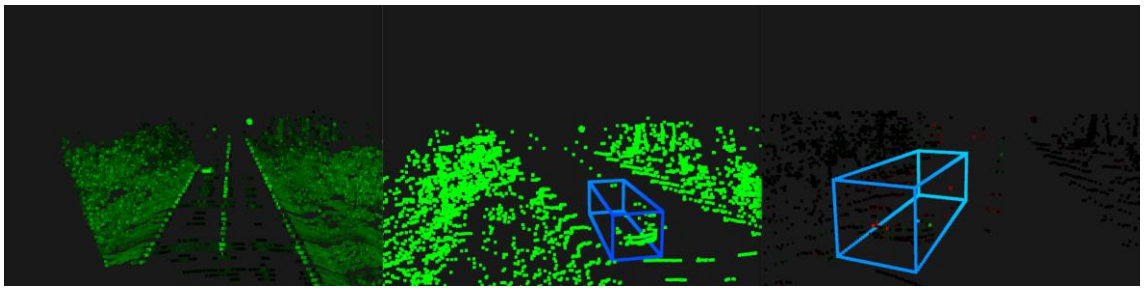
Integrated Gradients tulemuseks on omistusväärtused sisendi kujul. VoxelNet tehisnärvivõrgu sisend peab olema kujul (K, T, 7), kus K tähistab dünaamilist vokselite arvu ruudustikul sõltuvalt punktipilve suurusest, T tähistab maksimaalset punktide arvu ühes vokselis ning 7 tähistab tunnuseid, mis valitakse iga punkti kohta – x, y, z koordinaadid punktipilves, peegeldus r ja x', y', z' suhtelised koordinaadid vokseli suhtes. Visualiseerimiseks otsustas autor näidata omistusväärtuseid sisendiks olevale vokseliteks jagatud punktipilve peale. Kuna omistusväärtused arvutati eelnevalt mainitud seitsmele tunnusele eraldi, siis otsustas autor omistusväärtused iga punkti puhul kokku liita. Prooviti ka maksimaalse ja keskmise omistusväärtuse arvutamist, kuid summeerimine andis üldisema ülevaate punktide olulisusest tulemusele. Lisaks tuli omistusväärtused summeerida vokselite suhtes ja seejärel normaliseerida. Eraldati positiivsed ja negatiivsed omistused, et punktipilvel need välja tuua vastavalt rohelise ja punase värviga.

Omistusväärtuseid on võimalik valida ka individuaalselt, näiteks kui palju mõjutab punkti peegeldus ennustuse tulemust või kui palju mõjutavad eraldi absoluutsed või relatiivsed koordinaadid ennustuse tulemust.

Kuna vokselitest koosnev punktipilv oli suhteliselt madala resolutsiooniga, siis lisaks eelnevale kanti arvutatud omistusväärtused ka tavalisele KITTI stseeni punktipilvele, millelt on näha täpsemalt, millised punktid mõjutavad kõige rohkem ennustuse tulemust ja millised tekitavad probleeme.

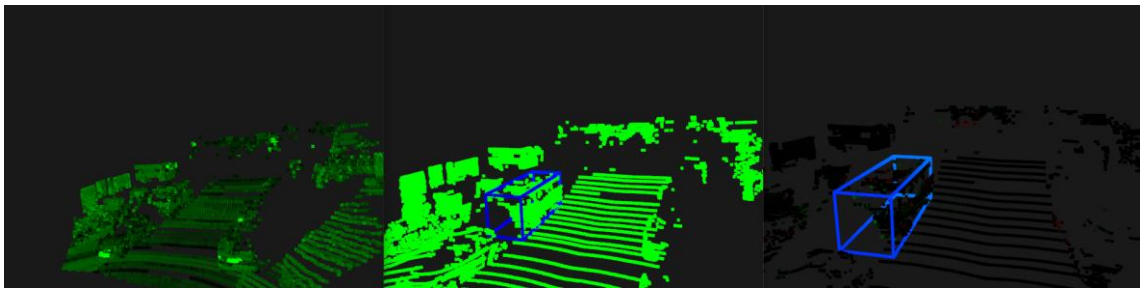
Joonisel 15 on näha stseenil 000013 valitud ennustuse peale arvutatud omistusväärtused Integrated Gradients algoritmi poolt.

Tänu Captumi universaalsele raamistikule, kasutati visualiseerimise loogikat ka teiste Captumi poolt pakutud algoritmide jaoks.



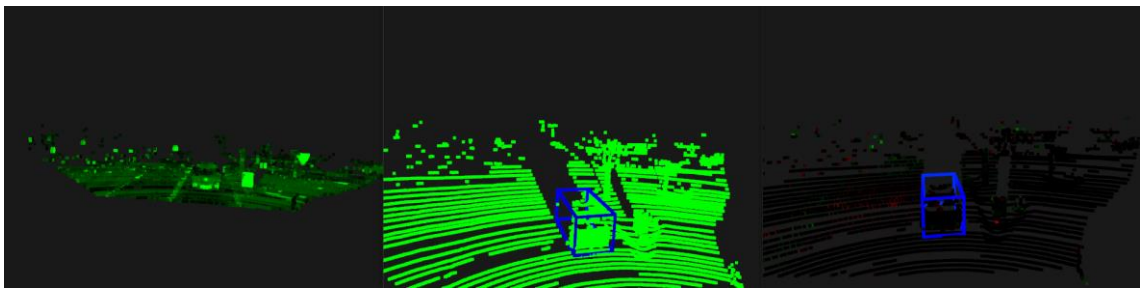
Joonis 15. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) Integrated Gradients eesmärgi suhtes.

Järgmiseks algoritmiks valis autor soovituskaartide algoritmi. Joonisel 16 on näha stseenil 000032 valitud ennustuse peale arvutatud omistusväärtused soovituskaartide algoritmi poolt.



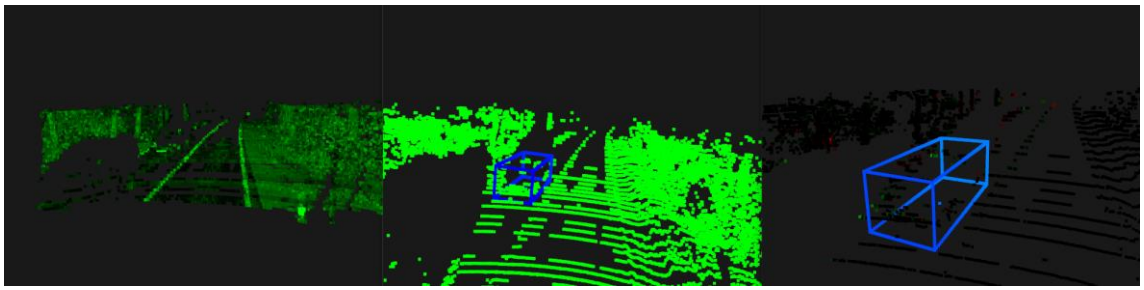
Joonis 16. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) soovituskaardid eesmärgi suhtes.

Joonisel 17 on näha stseenil 000036 98-protsendilise tõenäosusega valitud ennustuse peale arvutatud omistusväärtused GradientSHAP algoritmi poolt. Kahjuks on see üks algoritmidest, mis annab kõige vähem infot punktide olulisuse kohta punktipilves.



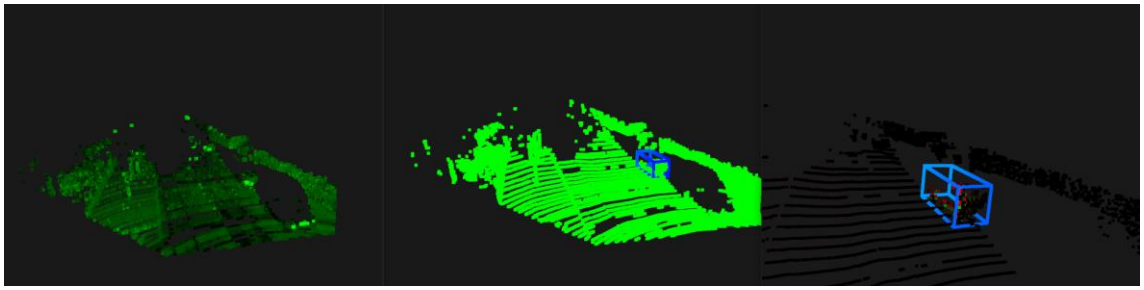
Joonis 17. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) GradientSHAP eesmärgi suhtes.

Joonisel 18 on näha stseenil 000044 99-protsendilise tõenäosusega valitud ennustuse peale arvutatud omistusväärtused InputXGradient algoritmi poolt.



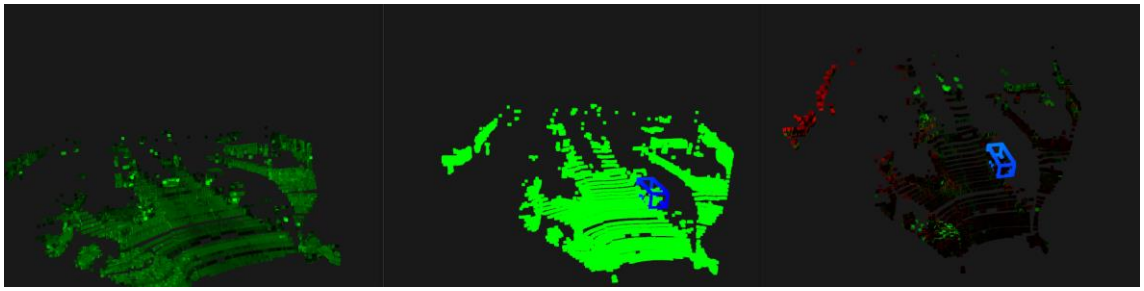
Joonis 18. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) InputXGradient eesmärgi suhtes.

Joonisel 19 on näha stseenil 000046 99-protsendilise tõenäosusega valitud ennustuse peale arvutatud omistusväärtused juhatatud tagasilevi algoritmi poolt. Juhatatud tagasilevi on autori arvates kõige täpsem algoritm, mida kasutada VoxelNeti implementatsiooni puhul.



Joonis 19. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) juhatatud tagasilevieesmärgi suhtes.

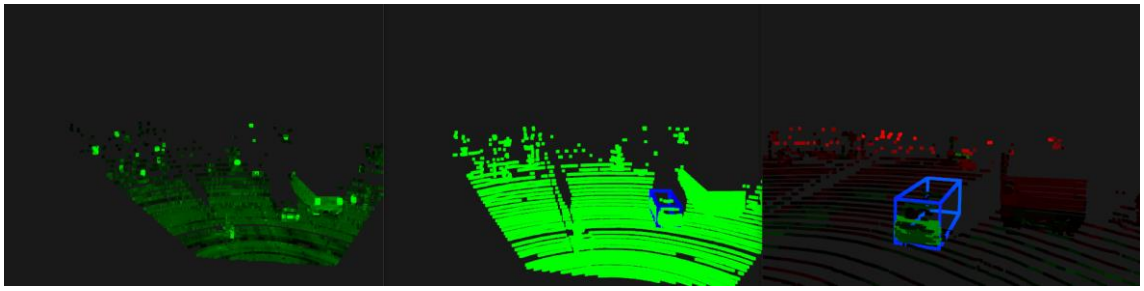
Joonisel 20 on näha stseenil 000049 97-protsendilise tõenäosusega valitud ennustuse peale arvutatud omistusväärtused dekonvolutsiooni poolt. Dekonvolutsiooniga ei ole mõistlik uurida vigaseid ennustusi, kuid on huvitav visualiseerida, milliseid mustreid konvolutsioonilised kihid on õppinud vaatama.



Joonis 20. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) dekonvolutsioon eesmärgi suhtes.

Joonisel 21 on näha stseenil 000064 99-protsendilise tõenäosusega valitud ennustuse peale arvutatud omistusväärtused oklusiooni poolt. Oklusioon määrab väga täpselt ära punktide olulisuse punktipilvel, kuid nende arvutamine võtab aega. Oklusiooni jaoks määras autor lohistatava akna (*sliding window*) suuruseks 20 ning täienduse (*padding*)

suuruseks 10. See tähendab, et omistuste arvutamiseks lohistatakse 20 vokseli suurune aken üle sisendiks oleva punktipilve 10 vokseli suuruste sammudega.

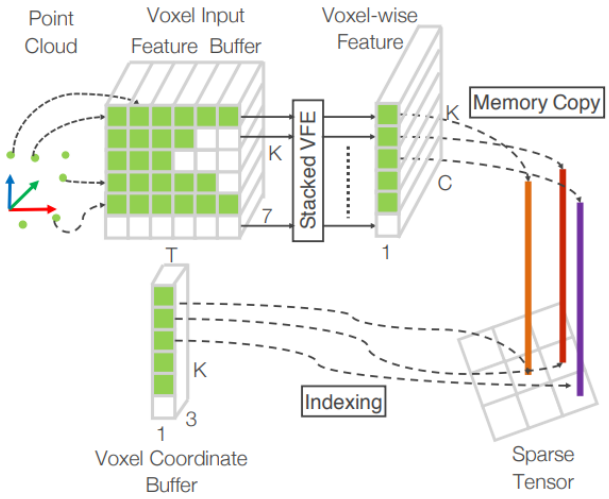


Joonis 21. 1) Sisend, 2) väljund koos valitud eesmärgiga ja 3) oklusioon eesmärgi suhtes.

4.3.2 SVFE kiht

Järgmisena on visualiseeritud *Stacked Voxel Feature Encoding* (SVFE) kihi omistusi.

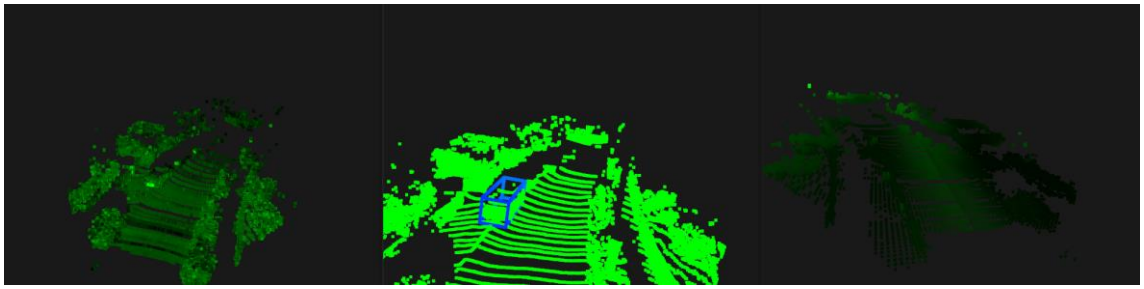
SVFE kihi sisend on kujul (paki suurus, K, T, 7), kus K on vokselite arv, T on maksimaalsete punktide arv vokselis ning 7 on eelnevalt mainitud eraldatud tunnuste arv ühe valitud punkti kohta. SVFE kiht õpib ette antud ühe punkti seitsme tunnuse abil 128 abstraktset tunnust iga vokseli kohta. SVFE väljund on kujul (K, paki suurus, 128). Kihi omistusväärtused arvutatakse valitud kihi väljundi suhtes. Kuna SVFE väljundis on alles vokseli järjekorranumber, siis on võimalik tänu koordinaatide puhvrile uuesti luua tehisnärvivõrgumudelile ette antud sisendi kujul punktipilv (Joonis 22). SVFE omistuste puhul on summeeritud kokku iga abstraktse tunnuse omistusväärtused.



Joonis 22. Efektiivne SVFE kihi implementatsioon [7].

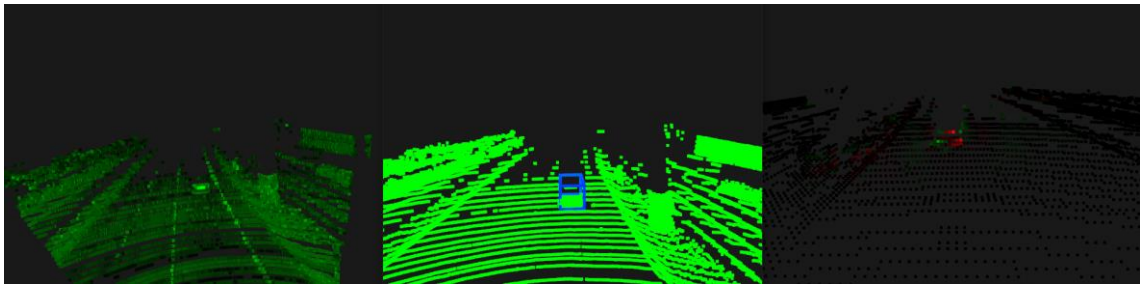
Järgnevatelt joonistel on ka näha, et SVFE omistusväärtuste punktipilved on peegelpildis võrreldes sisendiks oleva punktipilvega.

Joonisel 23 on näha stseenil 000075 98-protsendilise tõenäosusega valitud ennustuse peale arvutatud omistusväärtused kihi aktivatsiooni poolt. Nagu näha, siis kihi aktivatsioon SVFE kihi puhul ei anna mitte mingisugust infot punktide olulisusest. See on nii, sest ainuüksi SVFE kiht ei muuda ennustatava tunnuse tõenäosust, vaid teeb eeltöö järgmiste kihtide jaoks.



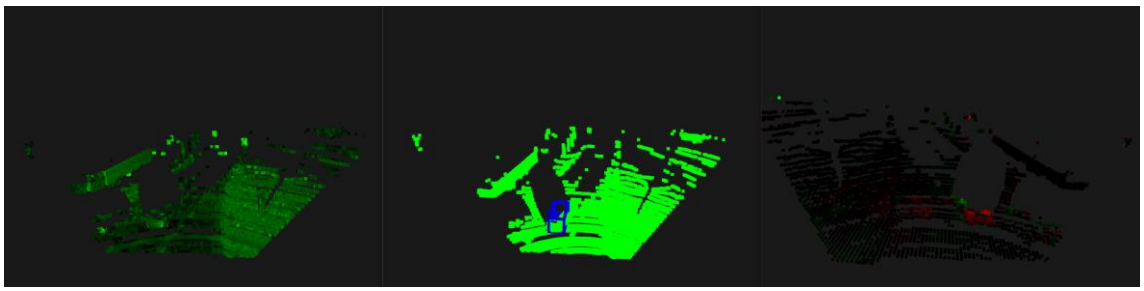
Joonis 23. Kihi aktivatsioon SVFE kihi suhtes.

Joonisel 24 on näha stseenil 000071 99-protsendilise tõenäosusega valitud ennustuse peale arvutatud SVFE kihi omistusväärtused kalle x aktivatsioon algoritmi poolt. Tänu SVFE õpitud abstraktsetele tunnustele võimaldab see visualiseerida paremini punktide olulisust valitud ennustuse jaoks. Nagu nimigi ütleb, korrutatakse kihi kalle x aktivatsiooni algoritmi puhul kihi neuronite aktivatsioon ja kalle läbi, mis võimaldab võrreldes joonisega 23 visualiseerida paremini kalde mõju omistusväärtustele.



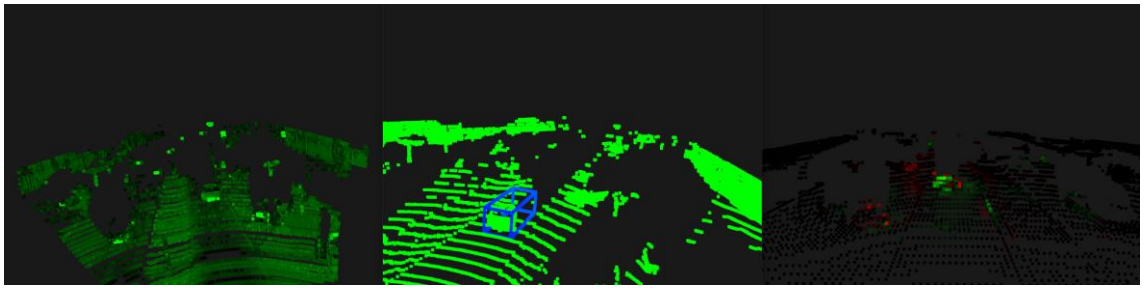
Joonis 24. Kihi kalle x aktivatsioon omistusväärtused SVFE kihi suhtes.

Joonisel 25 on näha stseenil 000079 96-protsendilise tõenäosusega valitud ennustuse peale arvutatud SVFE kihi omistusväärtused GradCAM algoritmi poolt.



Joonis 25. GradCAM omistusväärtused SVFE kihi suhtes.

Joonisel 26 on näha stseenil 000095 98-protsendilise tõenäosusega valitud ennustuse peale arvutatud SVFE kihi omistusväärtused Gradient SHAP algoritmi poolt.



Joonis 26. Gradient SHAP omistusväärtused SVFE kihi suhtes.

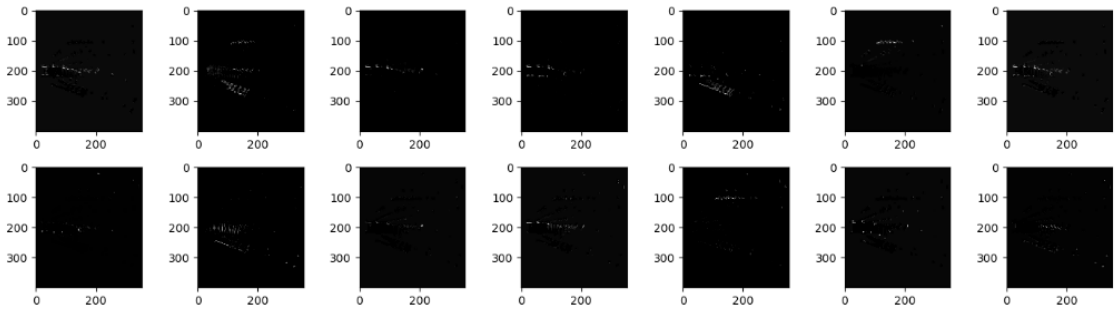
4.3.3 Keskmised konvolutsioonilise kihid

Keskmised konvolutsioonilised kihid (edaspidi KKK) saavad SVFE kihist sisendi kujul (paki suurus, 128, sügavus, kõrgus, laius), kus sügavus, kõrgus ja laius tähistavad vokseliteks jagatud ruudustiku koordinaate. KKK väljund on kujul (paki suurus, kanalite arv, sügavus, kõrgus, laius), kus kanalite arv tuleb 128 abstraktsest tunnusest, mis on konvolutsiooniliste kihtide tõttu ahendatud 64 peale. Samamoodi on ahendatud ka sügavus eelneva 10 pealt (autode puhul) kahe peale. Seetõttu ei ole enam võimalik visualiseerida kihi omistusi punktipilvena.

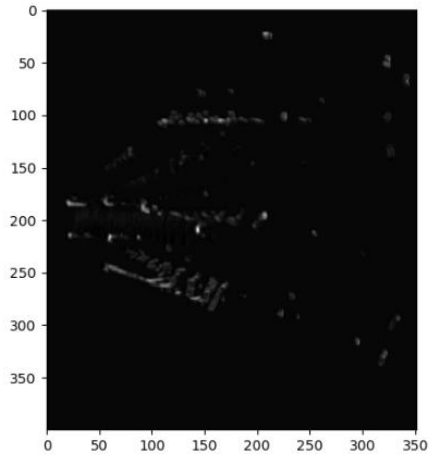
Visualiseerimiseks on kasutatud projektis olemasolevat matplotlib tarkvarateeki, mis võimaldab lihtsalt visualiseerida 2D maatrikseid.

KKK puhul on dokumendis visualiseeritud KKK tervikuna, kuid väikese edasiarendusega on võimalik uurida konvolutsioonilisi kihte ka sügavamalt.

KKK visualiseerimiseks on implementeeritud kaks variant – detailne (joonis 27) või tavaline (joonis 28). Detailne variant võimaldab analüüsida iga kanali omistusväärtuseid KKK kihis eraldi. Tavaline summeerib kanalite ja sügavuste kaupa omistusväärtused kokku ning kuvab neid ühe 2D pildina.



Joonis 27. Osa KKK aktivatsioonidest detailses vaates.



Joonis 28. KKK summeeritud aktivatsioon tavalises vaates.

4.4 Testimine

Eelnevalt joonisel näidatud ennustused olid kõik valitud korrektsed. Testimise peatükis on eesmärk aru saada, milliste tunnuste puhul tuvastab mudel vahel liiga kõrge tõenäosusega objekte, mida ei ole olemas.

Testimise jaoks on kasutatud eelnevalt mainitud 149 epohhini autode tuvastamiseks treenitud VoxelNeti mudelit. Mudel on väga täpne autode tuvastamiseks, kuid vahel tuleb ette ka kõrge tõenäosusega ennustusi puuduvale autole.

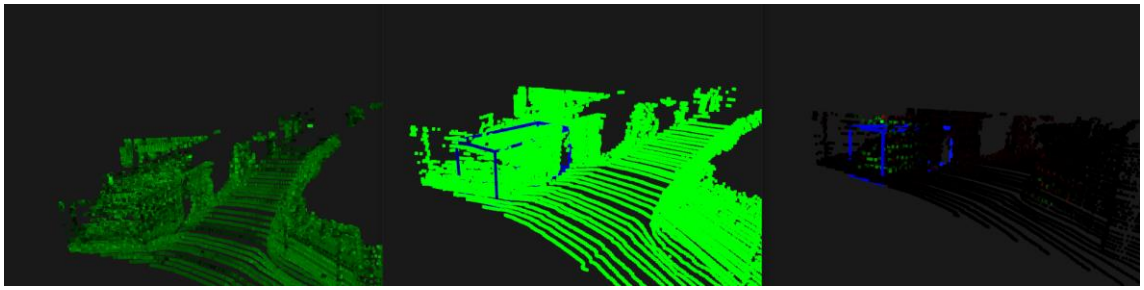
Testimiseks vaadati pisteliselt läbi erinevaid stseene ja vailiti välja ebakorrektsete ennustustega stseenid.

Stseenil 000467 (joonis 29) on ennustuseks kõrval asetsev aed, mille mudel märkis autoks 75-protsendilise tõenäosusega. Joonisel 30 on näha juhatatud tagasilevi vigase ennustuse suhtes. Kuna punktipilvel on aed lõigatud ära, siis see omab sarnaseid tunnuseid auto nurgale, mistõttu ka mudel arvab, et aia peaks märkima autoks. Joonisel 31 on näha

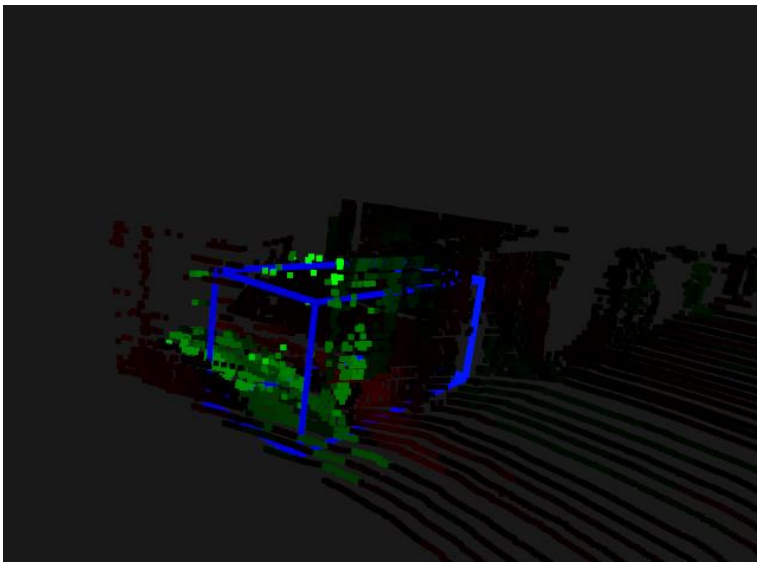
oklusioon sama tulemuse suhtes ning ka sellelt on näha, et aia lõigatud pool ja ülemine nurk tõstavad ennustuse tõenäosust kõige rohkem.

Eelnevalt mainitud *non-max suppression* meetod arvestab kõikide ennustustega, mille tõenäosus on üle 70 protsendi. Antud ennustuse eemaldamiseks oleks vaja tõsta piir üle 75 protsendi või ennustuste arvutamisel välja jätta piiril asuvad objektid.

Joonis 29. Stseen 000467 kaamera pilt.



Joonis 30. Vigane ennustus ja juhatatud tagasilevi ennustuse suhtes.



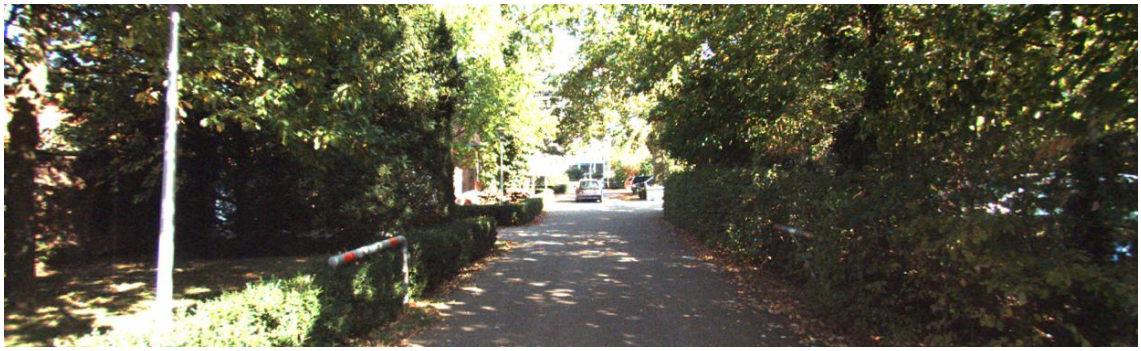
Joonis 31. Stseeni 000467 vigase ennustuse oklusioon.

Teiseks valiti stseenil 002252 (joonis 32) asetsev vigane 96-protsendilise tõenäosusega ennustus. Joonisel 33 on näha juhatatud tagasilevi vigase ennustuse suhtes.

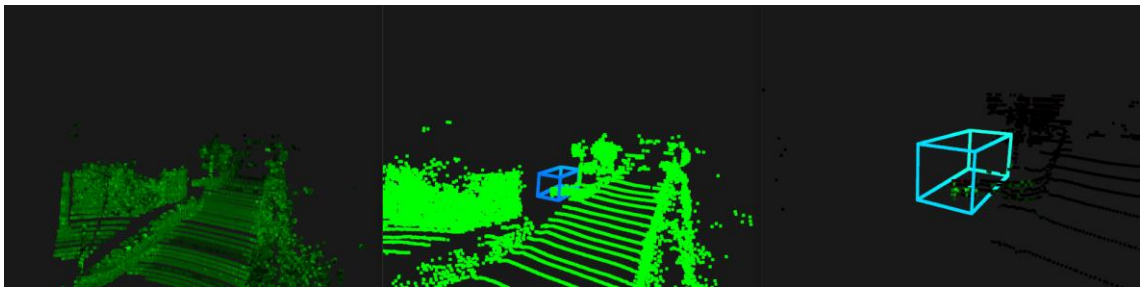
Pildilt on näha, et ennustus on tehtud vaid aia järgi, mille kõrgus on autoga võrreldes tavaliselt alumise tagaklaasi ääreni. Tõenäosust suurendavad oluliselt LiDARiga mõõdetud punktid, mis asetsevad üle aia, kujundades autole sarnase kuju.

Joonisel 34 on näha SVFE kihi suhtes arvutatud kalle x aktivatsioon omistusväärtused, kust näeb, et positiivset mõju avaldavad piirkasti sees asuvad punktid ning negatiivset mõju avaldab ülejäänud aed.

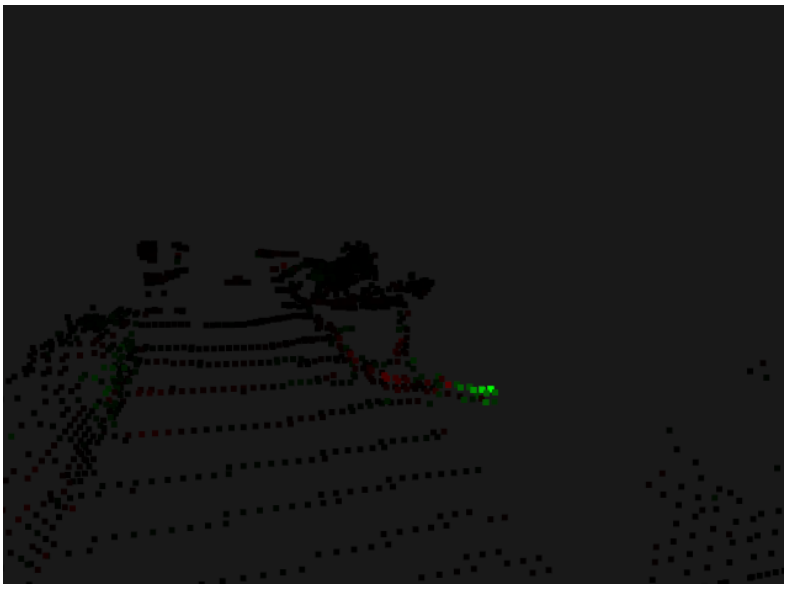
Antud juhul ei ole probleem autori arvates tehisnärvivõrgu arhitektuuris vaid väheses mudeli treenimises. Positiivsed punktid on tuvastanud sarnase mustri autodele, kuid negatiivsed punktid peaksid avaldama mõju rohkem kui positiivsed. Sellisel juhul langeb ka ennustuse tõenäosus.



Joonis 32. Stseen 002252 kaamera pilt.



Joonis 33. Stseeni 002252 vigase ennustuse juhatatud tagasilevi.



Joonis 34. Stseen 002252 SVFE kihi suhtes arvutatud vigase ennustuse kalle x aktivatsioon.

4.5 Järeldused

Kokkuvõtteks on teoreetilisi lahendusi erinevaid. Kasutades treenimiseks vaid lähemal olevaid objekte tõstab objektide tuvastamise täpsust kuna lähemal objektid on punktipilvel detailsemad. Praegu tuvastab mudel ära väga kõrge tõenäosusega kaugel asuvad objektid vaid mõne üksiku punkti järgi. Probleem tekib aga sisendis oleva müra puhul, kui punktid juhuslikult asetsevad kaugel asuvate autodega sarnasel kujul. Seda kutsutakse tehisnärvivõrkude puhul ülesobituseks.

Praktilises osas arendati 3D punktipilvede tugi Captumi 2D algoritmidele. Enamus algoritmidest andsid loetavaid tulemusi, kuid kohati oli müra ja madalate omistusväärtuste tõttu keeruline välja lugeda olulisi tunnuseid.

Lähtudes testimise peatüki katsetest, järeldab autor, et kõige täpsemaks algoritmiks 3D tehisnärvivõrkude puhul on kaldepõhistest algoritmidest juhatatud tagasilevi. Täpsed on ka segaduspõhised algoritmid, nagu testimisel kasutatud oklusioon, kuid nende arvutamine võtab kauem aega. Lisaks, kuna sisendis olevad vokselid ei ole järjestatud sarnaselt 2D piltide pikslitele, siis segaduspõhiste algoritmide puhul tekitas see omistusväärtuste puhul müra. Segaduspõhised algoritmid kasutavad tavaliselt lohisevaid aknaid (*sliding windows*) ning kui olulised vokselid satuvad mitteolulistega samasse aknasse, siis muutub ka mitteoluliste vokselite mõju.

5 Kokkuvõte

Käesolevas töös uuriti, kuidas interpreteerida keerulisi 3D tehisnärvivõrke. Alustati tehisnärvivõrkude tehnilisest ülevaatest, kirjeldati enimtuntuid ja töös kasutatud tehisnärvivõrgu arhitektuure ning kirjeldati 3D objektituvastuseks välja töötatud VoxelNet tehisnärvivõrgu arhitektuuri.

3D tehisnärvivõrgu aluseks võeti doktorant Martin Rebase poolt arendatud VoxelNet'i implementatsioon. Kuna implementatsioonis visualiseeriti treenimise ajal vaid kaofunktsiooni ning tulemused salvestati faili, siis arendati sellele juurde 3D andmete visualiseerimisvõimekus. Lisaks muudeti implementatsiooni nii, et see toetaks Facebook'i poolt arendatud Captum tarkvarateeki, mis pakub erinevaid algoritme tehisnärvivõrkude interpreteerimiseks. Implementatsiooni mudeli interpreteerimiseks kasutati peaaegu kõiki Captumi poolt pakutud algoritme ning visualiseeriti tunnuste olulisust nii tehisnärvivõrgu sisendi kui ka spetsiifiliste kihtide suhtes. Lõpuks katsetati silmajäänud algoritme vigaste ennustuse peal.

Tulemustest oli selgelt näha tunnused, mille alusel mudel otsuseid tegi. Samuti pakuti välja teoreetilised lahendused vigaste ennustuste tõenäosuste langetamiseks.

Lõpetuseks võib öelda, et 3D tehisnärvivõrkude interpreteerimisel on võimalik kasutada 2D tehisnärvivõrkude interpreteerimiseks kasutusel olevaid meetodeid ja algoritme. Töö põhieesmärk, milleks oli Captum tarkvarateegi toetamine ja komponentide olulisuse visualiseerimine VoxelNet'i implementatsioonil, sai täidetud ning see pakub väärtust ka tulevikus ootamatute tulemuste analüüsimiseks.

Kasutatud kirjandus

[1] Y.-H. L. C.-C. K. M.-H. C. I.-H. Y. Yung-Yao Chen, „Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes,“ *Sensors*, kd. 19, nr 9, 2019.

[2] „Data Assimilation by Artificial Neural Networks for an Atmospheric General Circulation Model,“ %1 *Advanced Applications for Artificial Neural Networks*, 2018, pp. 266-286.

[3] J. D. F. Z. a. J. J. Akanksh Basavaraju, „A Machine Learning Approach to Road Surface Anomaly Assessment using Smartphone Sensors,“ *IEEE Sensors Journal*, kd. 99, nr 1-1, 2019.

[4] S. W. Smith, „Continuous Signal Processing,“ %1 *The Scientist and Engineer's Guide to Digital Signal Processing*, San Diego, CA, California Technical Publishing, 2016.

[5] S. Saha, „A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,“ Medium, 15 December 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[6] K. H. R. G. a. J. S. Shaoqing Ren, „Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,“ *IEEE Transactions on Pattern Analysis and Machine Intelligence*, kd. 39, nr 6, 2016.

[7] Y. Z. j. O. Tuzel, „VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection,“ %1 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, Utah, United States, 2018.

[8] „3D Object Detection Evaluation 2017,“ [Võrgumaterjal]. Available: http://www.cvlibs.net/datasets/kitti/eval_object.php?obj_benchmark=3d.

[9] „Captum - Introduction,“ Facebook, [Võrgumaterjal]. Available: https://captum.ai/docs/introduction.

[10] G. Brasó, „Attribution Methods for Deep Convolutional Networks,“ 2018.

[11] „Captum - Algorith Descriptions,“ Facebook, [Võrgumaterjal]. Available: https://captum.ai/docs/algorithms.

[12] A. T. Q. Y. Mukund Sundararajan, „Axiomatic Attribution for Deep Networks,“ *Machine Learning*, kd. 70, pp. 3319-3328, 2017.

[13] E. C. C. Ö. M. G. Marco Ancona, „Towards better understading of gradient-based attribtion methods for deep neural networks,“ %1 *ICLR 2018*, Vancouver, Canada, 2018.

[14] A. V. A. Z. Karen Simonyan, „Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps,“ 2013.

[15] G. Tseng, „Interpretable Neural Networks,“ Towards data science, 16 November 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/interpretable-neural-networks-45ac8aa91411. [Kasutatud 4 May 2020].

[16] P. G. A. K. Avanti Shrikumar, „Learning Important Features Through Propagating Activation Differences,“ 2017.

[17] G. Tseng, „Interpretable Neural Networks,“ Medium, 16 November 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/interpretable-neural-networks-45ac8aa91411.

[18] S.-I. L. Scott M. Lundberg, „A Unified Approach to Interpreting Model Predictions,“ %1 *Advances in Neural Information Processing Systems 30*, Long Beach, California, United States, 2017.

[19] W. Kenton, „Shapley Value,“ 13 November 2019. [Võrgumaterjal]. Available: https://www.investopedia.com/terms/s/shapley-value.asp. [Kasutatud 7 May 2020].

[20] E. Ma, „Interpreting your deep learning model by SHAP,“ 18 August 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/interpreting-your-deep-learning-model-by-shap-e69be2b47893. [Kasutatud 7 May 2020].

[21] G. Tanner, „Interpreting PyTorch models with Captum,“ [Võrgumaterjal]. Available: https://gilberttanner.com/blog/interpreting-pytorch-models-with-captum. [Kasutatud 9 May 2020].

[22] „Captum - Algorithms,“ Facebook, [Võrgumaterjal]. Available: https://captum.ai/docs/algorithms.

[23] A. D. T. B. M. R. Jost Tobias Springenberg, „Striving for Simplicity: The All Convolutional Net,“ %1 *ICLR-2015*, San Diego, CA, USA, 2015.

[24] R. F. Matthew D Zeiler, „Visualizing and Understanding Convolutional Networks,“ %1 *ECCV 2014*, Zurich, Switzerland, 2013.

[25] M. D. Zeiler ja R. Fergus, „Visualizing and Understanding Convolutional Networks,“ *Computer Vision and Pattern Recognition*, 2013.

[26] N. Kumar, „Visualizing Convolution Neural Networks using Pytorch,“ Medium, 12 October 2019. [Võrgumaterjal]. Available: https://towardsdatascience.com/visualizing-convolution-neural-networks-using-pytorch-3dfa8443e74e.

[27] M. C. A. D. R. V. D. P. D. B. Ramprasaath R. Selvaraju, „Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization,“ %1 *ICCV'17*, Venice, Italy, 2016.

[28] W. S. K.-R. M. Grégoire Montavon, „Methods for interpreting and understanding deep neural networks,“ *Digital Signal Processing*, kd. 73, pp. 1-15, 2018.

[29] E. Inzaugarat, „Towards Data Science,“ 31 October 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/understanding-neural-networks-what-how-and-why-18ec703ebd31. [Kasutatud 16 April 2020].

[30] M. S. Q. Y. Kedar Dhamdhere, „How Important Is a Neuron?,“ %1 *ICLR 2019 Conference Blind Submission*, New Orleans, Louisiana, USA, 2018.

Lisa 1 – Juhend VoxelNet implementatsioon interpreteerimiseks

Antud lisas on toodud välja näited VoxelNet implementatsiooni interpreteerimisest Pythoni konsoolis või eraldi skriptina.

Interpreteerimise loogika on koondatud eraldi klassi, mis hõlmab endas automaatselt mudeli ja stseeni laadimist.

```
from interpretation.interpretation import Interpretation

scene_id = '000038'
runner = Interpretation(scene_id)
```

Järgnevalt on näha kõikide ennustuste visualiseerimine eelnevalt defineeritud stseenil. Tulemusena arvutab mudel ennustused, kasutab *non-max suppression* meetodit ning visualiseerib valitud ennustused punktipilvel kasutades open3d teeki.

```
runner.show_all_targets()
```

Üksikute ennustuste visualiseerimiseks on eraldi funktsioon, millel parameeter k tähistab ennustuse numbrit sorteerituna tõenäosuse alusel kahanevalt. Parim tõenäosus on numbriga 1. Tulemuseks visualiseerib funktsioon punktipilve, millel on näha vaid valitud ennustus.

```
runner.show_target(k=1)
```

Kui ennustuse number on teada, on võimalik arvutada ennustuse suhtes omistusväärtuseid. Järgnev funktsioon on üldiste omistusväärtuste visualiseerimiseks, mis võtab sisendis valitud üldise omistuse algoritmi Captumi poolt ning ennustuse numbri. Tulemuseks visualiseeritakse tehisnärvivõrgu sisend, muudetud kujul sisend koos ennustuse piirkastiga ning omistusväärtused koos ennustuse piirkastiga.

```
from captum.attr import GuidedBackprop
runner.show_general_attributions(GuidedBackprop, k=1)
```

Järgmiseks on loodud eraldi funktsioon SVFE kihi omistusväärtuste visualiseerimiseks. Funktsioonil on samad parameetrid, mis eelnevalt, kuid algoritmideks sobivat kihi omistusväärtuste algoritmid.

```
from captum.attr import LayerGradientXActivation
runner.show_svfe_attributions(LayerGradientXActivation, k=1)
```

Kuna iga kihi väljund on erinev, siis tuleb luua erinevate kihtide jaoks eraldi loogika visualiseerimiseks. Järgnevalt on loodud eraldi funktsioon keskmiste konvolutsiooniliste kihtide omistusväärtuste visualiseerimiseks. Parameetritele lisandub juurde tõeväärtustüüp, mis kirjeldab, millist varianti visualiseerida.

```
from captum.attr import LayerGradCam
runner.plot_middle_attributions(LayerGradCam, k=1,
detailed=False)
```