TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Tartu Kolledž

AUTODE TUVASTUSSÜSTEEM TARTU KOLLEDŽI PARKLA PÕHJAL

CAR DETECTION SYSTEM BASED ON TARTU COLLEGE PARKING LOT

RAKENDUSKÕRGHARIDUSTÖÖ

Üliõpilane: Henri Kübe

Üliõpilaskood : 193032

Juhendaja: Taavi Kase

Tartu 2023

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.
Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud.
Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"....." ..... 201…..

Autor: .....
/ allkiri /

Töö vastab bakalaureusetöö/magistritööle esitatud nõuetele

"....." ..... 201…..

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud

"....."..... 201… .

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina Henri Kübe (sünnikuupäev: 23.04.1999)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Autode Tuvastussüsteem Tartu Kolledži parkla põhjal", mille juhendaja on Taavi Kase,

1.1 reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;

1.2 üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

---

[1] *Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil.*

_____ (*allkiri*)

_____ (*kuupäev*)

Tartu kolledž
LÕPUTÖÖ ÜLESANNE

Üliõpilane: Henri Kübe, 193032

Õppekava, peaeriala: EDTR17/18, Küberfüüsikalised Süsteemid

Juhendaja: Taavi Kase taavi.kase@taltech.ee

Lõputöö teema:

Autode tuvastussüsteem Tartu Kolledži parkla põhjal

Car detection system based on the Tartu college parking lot

Lõputöö põhieesmärgid:

1. Luua süsteem autode automaatse tuvastamise Tartu kolledži parklas
2. Luua juhendid lõputöö ajal tehtud süsteemide uuesti kasutamiseks

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.		
2.		
3.		

Töö keel: eesti keel **Lõputöö esitamise tähtaeg:** "....."..........201….a

Üliõpilane: Henri Kübe ..... "....."..... 201….a
/allkiri/

Juhendaja: Taavi Kase ..... "....."..... 201….a
/allkiri/

Konsultant: …….. ..... "....."..... 201….a
/allkiri/

Programmijuht: Aime Ruus ..... "....."..... 201….a
/allkiri/

Kinnise kaitsmise ja/või lõputöö avalikustamise *piirangu tingimused formuleeritakse pöördel*

SISUKORD

EESSÕNA

Lõputöö sõnastati autori algatusel koostöös juhendaja Taavi Kasega. Töö on koostatud Tehnikaülikooli Tartu Kolledžis. Paigaldamistega, kooli poolse seonduva infrastruktuuriga ja konsultatsioonidega abistas Taavi Kase.

Tänan Taavi Kase lõputöö raames kaamerate hankimise ja kooliserveri ettevalmistamise eest.

Objektituvastus, masinõpe, süvaõpe, YOLO, rakenduskõrgharidustöö

SISSEJUHATUS

Linnastumise tõttu muutub parkimiskoha leidmine üha aeganõudvamaks ülesandeks. Autode koguse suurenemisega linnas suureneb vajadus parkimiskohtade jaoks. Kahjuks ei suuda linnad tihtipeale tagada kõikide jaoks parkimiskohta ja tekib puudujääk teatud piirkondades parkimiskohtadest. Lõputöö eesmärgiks on arendada välja objektituvastus süsteemi prototüüp, mille alusel hinnata kui efektiivne on süsteem kooliparklas olevate autode tuvastamisel.

Tartu Kolledžis on tudengite jaoks mõeldud kaks parkimisala. Üks parkimisala on koolimaja ees ja teine koolimaja taga. Selleks, et teada saada, kas parklas on veel vabu kohti, tuleb praegu parklasse sisse sõita. Tipptunnil võib see, aga tekitada ummikuid. Ummikud tekivad, kui üks tudeng üritab tagumisse parklasse autot parkida ja parkimiskohtade puudumisel üritab parklast väljuda. Samal ajal siseneb järgnev tudeng parklasse ja parkla täituvuse tõttu pole võimalik ümber keerata. Parkimist raskendab ka see, et tegu on kruusaparklaga ja parkimismärgistused puuduvad. Lõputöö teema tuleneb soovist kirjeldatud situatsiooni leevendada või täielikult eemaldada.

Lõputöö teoreetilises osas analüüsib autor varasemalt koostatud lahendusi ja selgitab kui hästi iga lahendus toimiks kooli parkla puhul. Seejärel selgitatakse masin- ja süvaõpet ning kirjeldatakse erinevaid algoritme süvaõppe rakendamiseks.

Lõputöö praktilises osas valitakse parim algoritm ettenähtud ülesande jaoks ja täidetakse vajadused algoritmi kasutamiseks. Kooli parkla kohale paigaldatakse kaamerad, mis ühenduvad koostatud süsteemiga. Lõputöö lõpus analüüsitakse, kui hästi koostatud süsteem toimib.

Lõputöö raames paigaldas autor kaamerad parkla vastas oleva A korpuse hoone teise korruse aknalaudadele. Kaamerad on ühendatud Wi-Fi võrguga. Kooli serveris on virtuaalmasin, mis töötleb kaamerapilti ja annab ülevaadet kooli parklast. Virtuaalmasinale on paigaldatud objektituvastustarkvara, mis suudab ära lugeda autode koguse parklas. Autode kogus salvestatakse perioodiliselt faili, millega on hiljem võimalik teha statistikat. Tulevikus oleks võimalik teha sobilik andmebaas andmete salvestamiseks, mille abil oleks võimalik andmed edasi saata näiteks telefoni rakendusele ja andmed kättesaadavaks teha tudengitele.

Autor analüüsib, kui hästi koostatud süsteem toimib ja kas süsteemi oleks mõttekas edasi arendada autode tuvastamise eesmärgil. Lõputöö lõpus annab autor juhised, kuidas süsteemi kasutada ja vajadusel uuesti koostada.

1. VAREM TEHTUD TÖÖD JA SOBIVUS KOOLI PARKLAGA

1.1 Smartparking arukas parkimissüsteem

Väga populaarne lahendus parkimise tuvastamiseks on sensorid iga parkimiskoha jaoks, millega tuvastatakse auto olemasolu. Andurid võivad olla maasisesed või parkimiskoha peal olevad. Sellist lahendust kasutatakse, näiteks Tartu Kaubamaja parklas. Enne parklasse sisenemist on näha vabade parkimiskohtade arvu stendilt. Tänu sellisele süsteemile välistatakse täielikult parkla üleliigset täitumist. Süsteemiga on võimalik näha kaugelt, kas parkimiskoht on kasutusel või mitte. Selline süsteem sobib väga hästi tähistatud parkla jaoks [1].

Autori hinnangul parkimiskoha andur iga parkimiskoha jaoks kooliparkla puhul ei sobi. Niisugune süsteem vajab markeeritud parkimiskohti ja hind andurite ning süsteemi tegemiseks oleks liiga suur. Samuti ei oleks kasulik paigutada indikaatortuld parkimiskoha kohale, kuna indikaatorit poleks kaugelt näha.

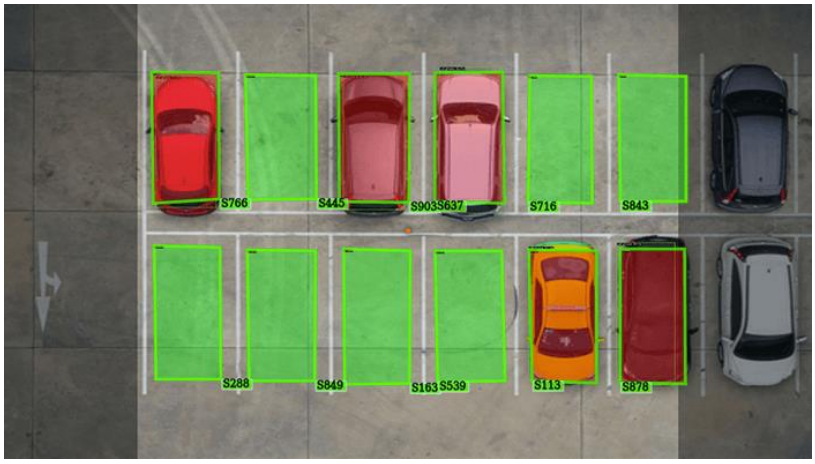
Joonis 1.1 Parkimislahendus Tartu kaubamaja parklas [2]

1.2 DeepParking lahendus

DeepParking on nutikas parkimislahendus, mis kasutab parkimishalduse automatiseerimiseks arvutinägemise tehnoloogiat. Süsteem on loodud selleks, et muuta parkimine juhtide jaoks tõhusamaks ja mugavamaks, vähendades samal ajal parkimise haldamise kulusid ja keerukust. DeepParking töötab kaamerate, andurite ja masinõppe algoritmide kombinatsiooni abil ning lubab jälgida parkimiskohti reaalajas. Kui juht siseneb parklasse, tuvastab süsteem numbrimärgi. Selle abil tehakse kindlaks, kas juhil on parklas parkimisõigus [3].

Kui sõiduk on volitatud, suunab süsteem juhi digimärkide või mobiilirakenduse märguannete abil vabale parkimiskohale. Seejärel jälgivad kaamerad ja andurid parkimiskohta veendumaks, et sõiduk on õigesti pargitud ega ületanud parkimisaega. Omavolilise parkimise või parkimisreeglite rikkumise korral teavitab süsteem parkimisoperaatorit või järelevalvetöötajaid. Süsteemi saab integreerida ka maksesüsteemidega, et võimaldada parkimisarvete automaatset väljastamist. DeepParking pakub autojuhtidele ka mobiilirakendust, mis võimaldab neil leida vabu parkimiskohti, broneerida parkimiskohti ja maksta parkimistasusid [3].

Autori hinnangul oleks kasulik rakendada kirjeldatud süsteemist arvutinägemist. Süsteemist on võimalik välja jätta autode numbrimärkide tuvastamise ja volitamise, juhtide suunamise parkimiskohani, maksesüsteemid ja igasugused süsteemiteavitused rikkumiste korral. Selle asemel, et tuvastada parkimiskohti on kooli süsteemis vaja tuvastada autosid. DeepParking süsteem eeldab, et parkimiskohad on markeeritud. Kooli parkla on markeerimata ala ja spetsiifilisi parkimiskohti pole võimalik märkida.



Joonis 1.2 Märgistatud parkla automatiseerimine DeepParking lahendusega [3]

1.3 Auto tuvastamine kooli värava ees

Üks pakutud lahendustest Tartu kolledži õppejõudude poolt oli paigaldada andurid kooli hoovi värava juurde. Andur või tuvastuskaamera tuvastaks auto sisenemise või väljumise. Lahendusega saaks väga lihtsalt mõõta ära mitu autot siseneb kooli hoovi ja mitu väljub ja korrektselt edastada, kui palju autosid parklas tegelikkuses on.

Autori hinnangul oleks sellise lahenduse puhul on mitmed puudujäägid. Kuna koolihoovi värava kaudu sisenevad autod eesmisse ja tagumisse parklasse pole võimalik eristada, kuhu auto on pargitud. Lisaks sisenevad värava kaudu ka õppejõudude autod. Õppejõududel on tudengitest eraldatud parkla ja õppejõudude autosid ei tohi koosseisu lisada.

2. MASINÕPE JA SÜVAÕPE

2.1 Masinõpe

Masinõpe on tehisintellekti alamvaldkond. Masinõpe hõlmab algoritme, mis suudavad õppida andmetest ja teha nende põhjal ennustusi või otsuseid. Masinõppemudeleid koolitatakse märgistatud andmete põhjal ning need kasutavad mustrite tuvastamiseks ja uute andmete prognoosimiseks statistilisi meetodeid [4].

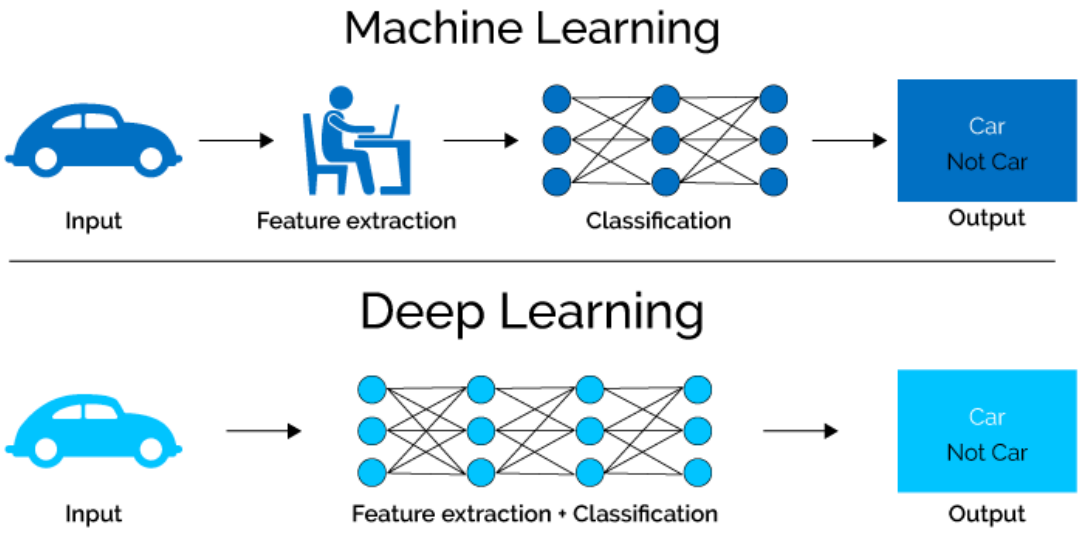
2.2 Süvaõpe

Süvaõpe on masinõppe alamvaldkond, mis kasutab õppimiseks ja suurte andmemahtude põhjal prognooside tegemiseks mitmekihilisi tehisnärvivõrke ehk ANN-e (*Artificial Neural Network*) [5]. Süvaõppes konstrueeritakse ANN-id mitme kihiga omavahel ühendatud sõlmedest või neuronitest, kus iga neuron teeb oma sisenditega lihtsa matemaatilise toimingu ja edastab tulemuse järgmisele kihile. Kihid on virnastatud üksteise peale, moodustades sisendandmete hierarhilise esituse. Võrgu esimene kiht võtab vastu töötlemata sisendandmed, näiteks pildi või heli lainekuju. Järgmised kihid töötlevad andmeid ja teevad sisendandmetega üha keerukamaid arvutusi [6].

Süvaõppemudeli väljaõpetamine toimub mudelile suure hulga märgistatud andmete sisendamisega ja selle kaalude ja eelarvamuste kohandamisega. Seda nimetatakse tagasileviks. Tagasilevi on tehnika, mis arvutab kaofunktsiooni gradiendi mudeli parameetrite suhtes. Kaofunktsioon näitab, kui palju iga parameeter aitab kaasa mudeli veale. Seejärel värskendatakse kaalusid ja kaldeid gradiendi vastassuunas, kasutades kaotusfunktsiooni minimeerimiseks optimeerimisalgoritmi [5, 6].

Süvaõppe eelis on võime õppida toorandmetest automaatselt keerulisi ja abstraktseid omadusi. Süvaõppe on eriti kasulik näiteks pildituvastuses.

Süvaõppe puuduseks on näiteks arvutuslik kulukus. Mudelite treenimine vajab palju arvutusvõimekust ja samuti palju märgistatud andmeid. Mudelitel võib ka tekkida üle sobitus. Üle sobitus toimub, kui mudel jätab koolitusandmed liiga hästi meelde ja ei üldista piisavalt andmeid. Üle sobituse tulemusel ei suuda mudel tuvastada uusi andmeid [7].



Joonis 2.1 Masinõpe ja Süvaõpe näitena [8]

2.2.1 Süvaõppe kihid

Süvaõpe koosneb erinevatest kihtidest ja igal kihil on oma kindel ülesanne. Kihid tihtipeale jaotatakse kolmeks: sisendkiht, peidetud kihid ja väljundkiht. Sisendkiht on kiht, kus andmed vastu võetakse. Vahest lisatakse sisendkihile juurde ka pilditöötlust. Pilditöötlust tehakse, siis kui on vaja teha pildid sobilikumaks süvaõppe jaoks või kui on vaja tuvastada objekte erinevates suurustes, mille korral pilte skaleeritakse väiksemaks või suuremaks. Väljundkiht on närvivõrgu poolt välja töötatud kiht, mis väljastab tulemused. Peidetud kihid on kihid, kus tegelik süvaõpe toimub. Need kihid tavapäraselt on:

Normaliseerimiskiht - kiht skaleerib sisendandmed sobivate intervallidega. Normaliseerimiskihiga muudetakse mudel stabiilsemaks ja parandatakse jõudlust.

Konvolutsioonikiht - kiht rakendab sisendile filtreid. Filtrite abil tõstetakse esile sisendi olulised omadused. Kiht on väga kasulik, näiteks kujutiste klassifitseerimises ja objektide tuvastamises.

Koondamiskiht – kiht vähendab funktsioonide tundlikkust ja säilitades kõige silmatorkavamad funktsioonid ja omadused. Selle tulemusel saavutatakse paremad ja üldisemad andmeid. Samuti vähendatakse tunnuskaartide suurust.

Aktiveerimiskiht – kiht rakendab sisendkihile matemaatilisi funktsioone f(x). Samuti tutvustatakse mudelile mittelineaarsust, võimaldades mudelil õppida keerukamaid seoseid sisendite ja väljundite vahel.

Väljalangemise kiht – kiht tühistab teatud juhuslikud sisendväärtused, et genereerida üldisem andmestik ja vältida üle sobitamise probleemi. Üle sobitamine toimub siis, kui mudel muutub liiga keeruliseks ja hakkab andmetesse sobitama müra, mitte aluseks olevaid mustreid [9].

3. OBJEKTITUVASTUS ALGORITMID

Objektide tuvastamine on tehisintellekti võtmevaldkond, mis võimaldab arvutisüsteemidel oma keskkonda tajuda, tuvastades objekte visuaalsetes piltides või videotes. Objektituvastusel on palju rakendusi erinevates valdkondades, nagu valve, isejuhtivad autod ja robootika.

Süvaõpe on aidanud täpsemat ja tõhusamat objektide tuvastamist, kui traditsioonilised arvuti nägemismeetodid. Süvaõppe põhjal on välja mõeldud mitmed algoritmid, mis lihtsustavad süvaõppe rakendamist objektituvastuse eesmärgil [10].

3.1 R-CNN

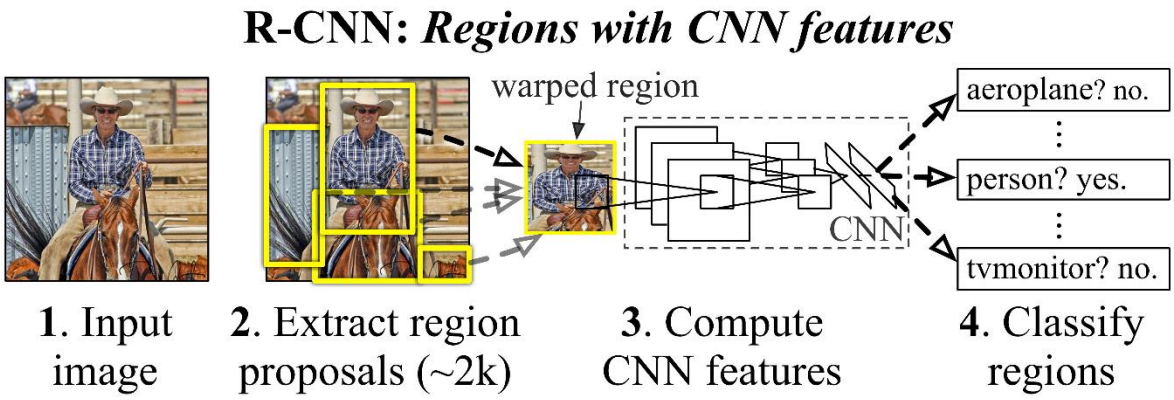
R-CNN (Region-Based Convolutional Neural Network) on väga populaarne objektituvastus algoritm. R-CNN loob esmalt piirkonna ettepanekud ja seejärel klassifitseerib objektid nende ettepanekute sees. R-CNN-i mudelid kasutavad CNN-i (Convolutional Neural Network), et eraldada igast ettepanekust funktsioonid. Funktsioone kasutatakse objektide asukoha klassifitseerimiseks ja täpsustamiseks ettepanekus.

R-CNN-i algoritm koosneb kolmest põhietapist: ettepaneku genereerimine, funktsioonide eraldamine ja objektide klassifitseerimine.

Ettepaneku etapis töödeldakse sisendkujutist objekti ettepanekute komplekti loomiseks. Ettepanekud on pildi piirkonnad, kus objektid asuvad. Need ettepanekud luuakse selektiivse otsingu algoritmi abil. Selektiivse otsingu algoritm kombineerib madala tasemega pildifunktsioone, et rühmitada sarnased pikslid piirkondadesse.

Funktsioonide eraldamise etapis edastatakse iga objekti ettepanek CNN-i kaudu. Iga ettepaneku jaoks eraldatakse funktsioonivektor. CNN on eelnevalt koolitatud suure andmekogumi jaoks, et tuvastada mitmesuguseid madala taseme funktsioone, nagu servad ja nurgad.

Objekti klassifikatsiooni etapis iga ettepaneku tunnusvektor sisestatakse seejärel klassifikaatorite komplekti, et määrata iga objektiklassi olemasolu. Klassifikaatorid on koolitatud kasutades kaofunktsiooni, et õppida iga klassi tõenäosust, arvestades CNN-i eraldatud funktsioone [11].



Joonis 2.2 R-CNN kaheastmelise detektori visuaalne selgitus [10]

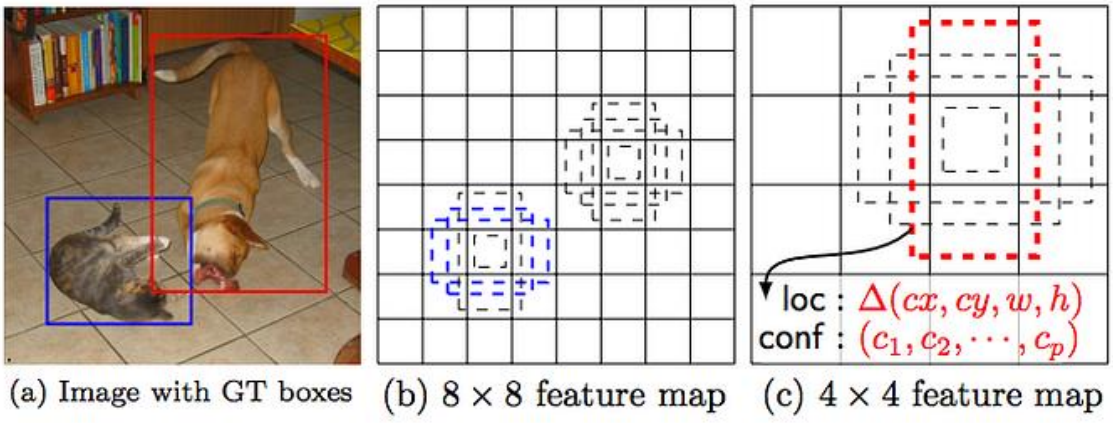
3.2 SSD

SSD (*Single Shot MultiBox Detector*) on populaarne objektituvastussüsteem, mis suudab objekte reaalajas tuvastada. SSD-algoritm jagab sisendkujutise eelnevalt määratletud kuvasuhete komplektiks ning ennustades seejärel objektide olemasolu ja nende asukohti igas piirkonnas. See kasutab sisendpildist funktsioonide eraldamiseks keerdkihtide seeriat, millele järgneb ennustuskihtide komplekt, mis ennustavad objekti klassi ja asukohta igas piirkonnas.

Funktsioonide ekstraheerimine - sisendpilt juhitakse läbi rea konvolutsioonikihte, mis eraldavad eri ulatuse ja abstraktsioonitasemega funktsioone. Neid funktsioone kasutatakse seejärel objektide tuvastamiseks pildi erinevates piirkondades.

Ankrukasti genereerimine - iga objektikaardi jaoks luuakse erineva kuvasuhte ja suurusega ankrukastide komplekt. Neid ankrukaste kasutatakse objekti asukoha ja suuruse tuvastamiseks igas piirkonnas.

Klassifikatsioon ja regressioon - funktsioonide eraldamise ja ankrukasti genereerimise etappide väljund juhitakse läbi ennustuskihtide komplekti. Need kihid ennustavad iga objektiklassi olemasolu ja täpsustavad prognoositava objekti asukohta ja suurust igas ankrukastis [12].



Joonis 2.3 SSD Üheastmelise detektori visuaalne selgitus [12]

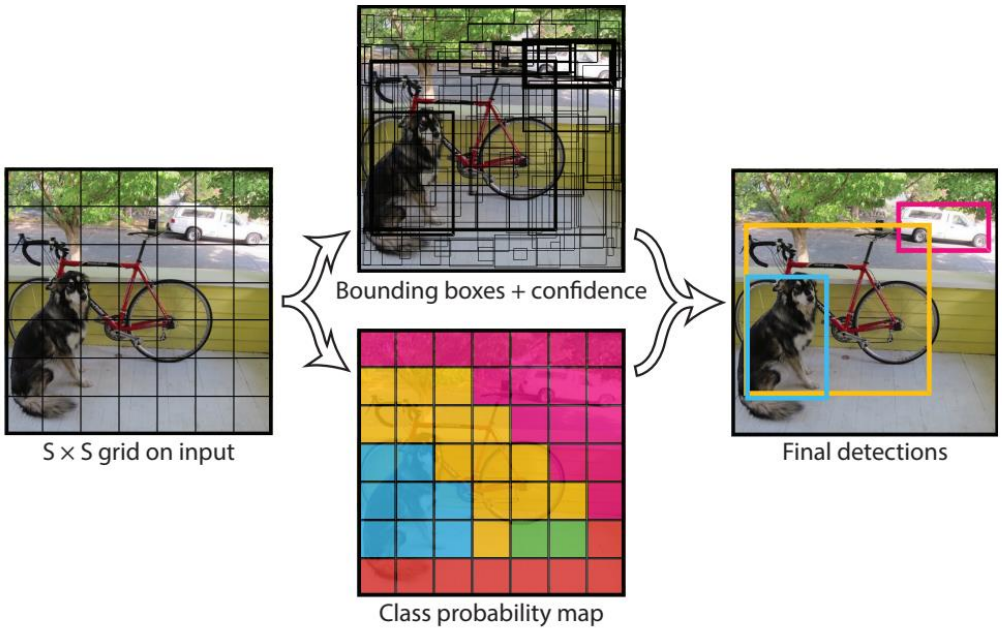
3.3 YOLO

YOLO (*You only live Once*) on tipptasemel objektituvastussüsteem, mida kasutatakse arvutinägemisrakendustes. YOLO on tuntud oma reaalajas objektide tuvastamise võimaluste poolest, saavutades kõrge täpsuse madalate arvutuskuludega.

YOLO jagab sisendpildi lahtrite ruudustikuks ning ennustab iga lahtri jaoks piirdekaste ja klasside tõenäosusi. Iga piirdekast ennustab objekti asukohta ja sellele vastava klassi tõenäosust. Seejärel arvutab võrk iga ennustatud piirdekasti usaldusväärsuse skoori, mis näitab tõenäosust, et objekt asub piirdekastis.

YOLO kasutab objektide tuvastamiseks ja klassifitseerimiseks ühte närvivõrku. See erineb teistest objektituvastussüsteemidest, mis kasutavad objektide tuvastamiseks ja klassifitseerimiseks tavaliselt mitut närvivõrku. See lähenemisviis võimaldab YOLO süsteemil saavutada head täpsust, muutes selle ideaalseks rakenduste jaoks, kus kiirus on kriitiline.

YOLO süsteemil on võrreldes teiste objektituvastussüsteemidega palju eeliseid. Üks olulisemaid eeliseid on selle kiirus. YOLO suudab objekte reaalajas tuvastada, muutes selle ideaalseks rakendustes, kus kiirus on kriitiline, nagu isejuhtivad autod, valvesüsteemid ja robootika. Selle ühtne võrguarhitektuur muudab selle ka teistest objektituvastussüsteemidest tõhusamaks, vähendades arvutusnõudeid ja hõlbustades teistesse süsteemidesse integreerimist [13].



Joonis 2.4 YOLO algoritmi visuaalne selgitus [13]

3.4 Algoritmi valik lõputöö raames

Esimene keeruline ülesanne lõputöö tegemisel oli sobiva algoritmi valimine. Peamine piirang, millega autor pidi arvestama oli arvutuslikud ja mälu piirangud. Lõputöö raames koostatud süsteem oli plaanis paigutada kooli serverile. Kooli server ei ole võimeline täies mahus rakendama objektituvastust, kuna see töötab madala võimelise protsessoriga ja väga piiratud mälukogusega. Objektituvastuse täies mahus rakendamiseks on vaja vähemalt modernset videokaarti. Autori hinnangul sobib näiteks Nvidia 3060TI videokaarti, mille abil oleks võimalik saavutada populaarsete objektituvastusalgoritmidega 70-80 kaadrisagedust. Kooliserveril hakkab objekti tuvastus töötama videokaardi asemel protsessoril.

R-CNN algoritmil on teatud puudused, mis muudab selle sobimatuks lõputöö tegemise raames. Esiteks koosneb R-CNN mitmest etapist, sealhulgas piirkonna ettepaneku genereerimine, funktsioonide eraldamine ja klassifitseerimine, muutes selle arvutuslikult kulukaks. Kõrged arvutusnõuded koormavad madala võimekusega riistvara, mille tulemuseks on aeglasem tuvastamiskiirus ja piiratud reaalajas võimalused. Lisaks on R-CNN algoritmi mälunõuded suured, mistõttu on keeruline kasutada piiratud ressurssidega seadmeid [14].

SSD suudab tagada reaalajas jõudluse ja saavutab väga hea täpsuse. SSD algoritmiga kaasnevad ka mitmed piirangud, kui seda kasutatakse vähese ressursiga süsteemides. Üks SSD peamisi probleeme on selle keerukus, mis tuleneb ennustussüsteemist. Keerukus seab väljakutseid madala ressurssidega riistvarale suure hulga konvolutsioonikihtide ja sellega seotud arvutuslike kulude tõttu. Lisaks SSD nõutav suur mälumaht piirab selle rakendatavust [14].

Autori hinnangul oli parim algoritmi valik lõputöö raames YOLO. YOLO algoritm suudab kõige paremini ja kõige tõhusamalt tagada kiiret tuvastamist madalate arvutuskuludega. Lisaks on YOLO arhitektuur mälutõhus ja seda on võimalik optimeerida erinevate riistvarakonfiguratsioonide jaoks. Samuti saab YOLO algoritmi kohandada vastavalt konkreetsetele ressursipiirangutele. Kirjeldatud eeliste pärast valis autor lõputöö raames koostatud süsteemi jaoks YOLO algoritmi.

4. ANDMETE KOGUMINE

4.1 Kaamerad ja kaamerate paigaldamine

Lõputöö raames autode tuvastamiseks valis autor kaameraks ZSKJ välikaamera. Kaamera valiti kuna sobis objektituvastus ülesande jaoks kõige paremini. Omadused, mis läksid arvestusse kaamera valiku puhul:

- Odav Hind
- Sobib välitingimustes
- Suudab otse ühenduda võrguga
- Lai tuvastusnurk
- Hea Eraldusvõime
- On kiiresti kättesaadav

Lõputöö koostamiseks ostis juhendaja Taavi Kase 2 ZSKJ välikaamerat. Kaamerate täpsed parameetrid on nähtavad lisades [Lisa 1].

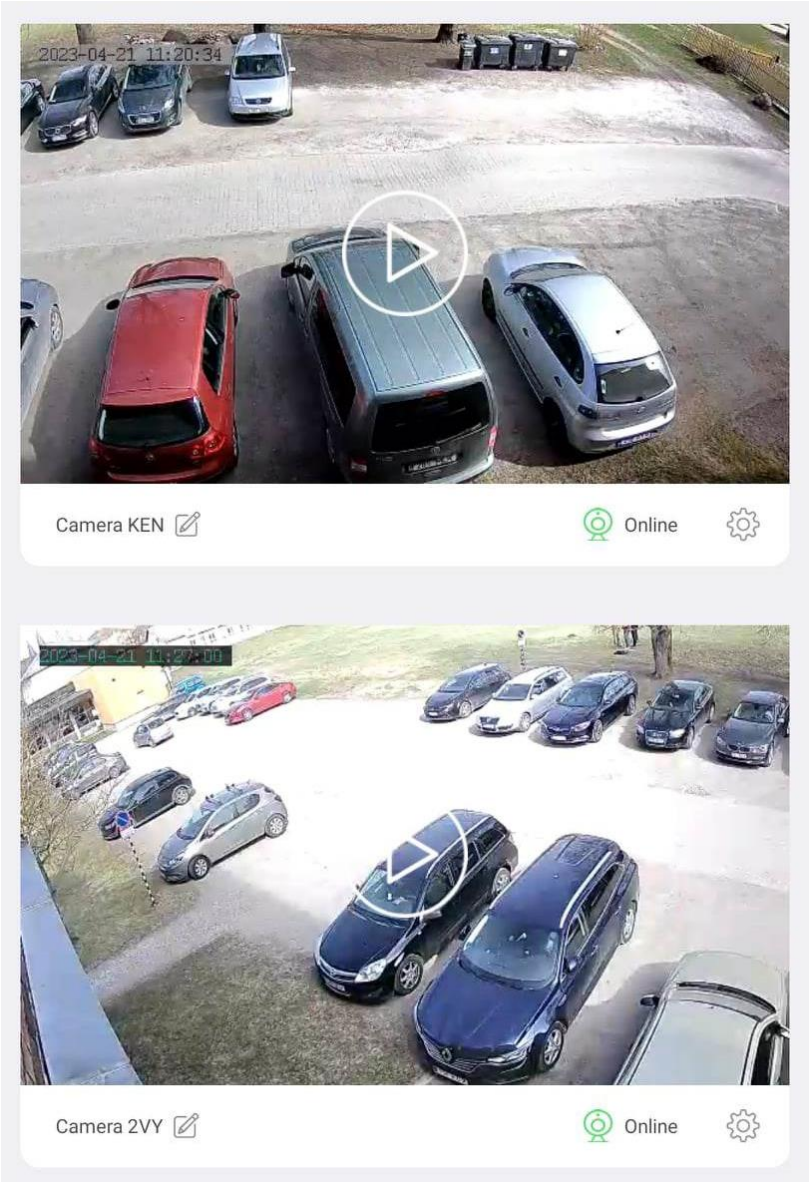
Kaamerad paigaldati koos juhtmekarbiga puidust alusele. Paigaldatud karbi ülesanne on vältida seina sisse puurimist jämedate juhtmete tõttu. Kaamera toitejuhe on liiga lühike ja vajas ühenduse tekitamiseks puurimist, kuid koos karbiga oli võimalik paigaldada kõik juhtmed väljas olevale aknalauale.

Joonis 4.1 Kaamera koos karbi ja alusega enne aknalauale paigaldamist

Kaamerate aknalauale paigaldamisel tekkisid mitmed probleemid. Kõige suurem probleem oli sobimatu kaamera nurk – kaamerat polnud piisavalt hästi võimalik keerata, et maja kõrval olevaid autosid oleks näha. Selle jaoks valmistasid autor ja juhendaja Taavi Kase kaamerate jaoks alused. Alused on puidust ja suunasid kaamera pildi paremini kooli parkla suunas. Kaamerad said lõplikult paigaldatud A205 ruumis kahele aknalauale.

4.2 Kaamerate ühendamine tarkvaraga

Kaamerate ühendamisel tarkvaraga tekkis mitmeid probleeme. Kaamerad olid originaalselt mõeldud juhtimiseks vaid läbi telefoni tarkvara. Otseseid juhendeid arvutis kasutamiseks kaameratel polnud.



Joonis 4.2 Kaamerate juhtimine läbi telefoni tarkvara

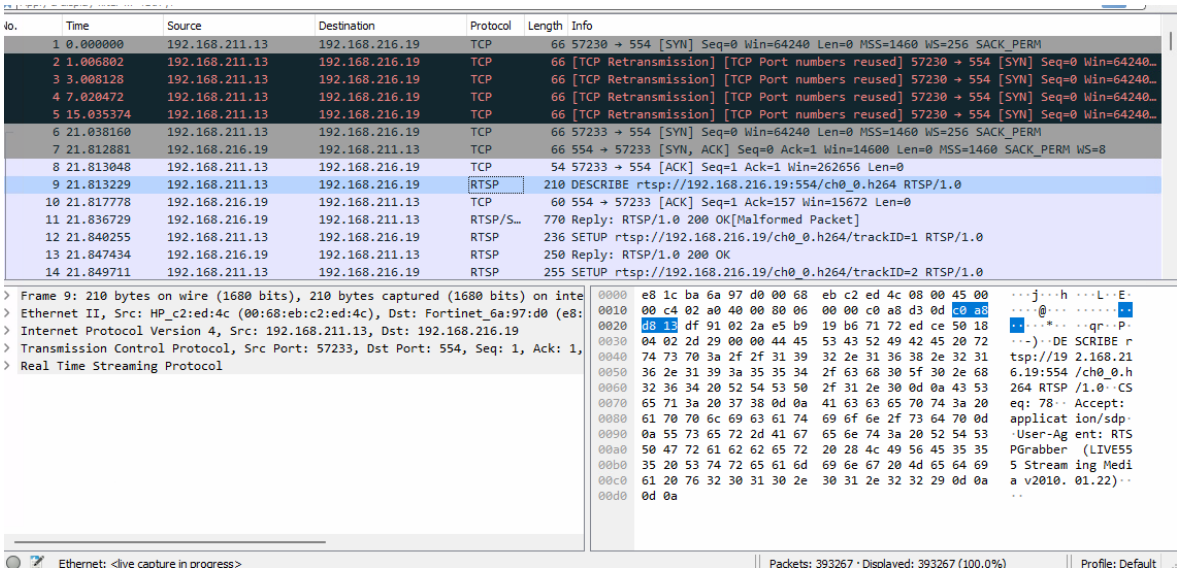
Läbi telefoni juhtimine polnud lõputöö raames piisav, kuna kaamera pilti oli vaja töödelda läbi arvuti ja Python'i skripti. Läbi Python'i skripti töötlemine tähendab seda, et oli vaja leida täpne aadress ja protokoll kaamera jaoks. Läbi täpse aadressi saab rakendada Pythoni tarkvara teeki OpenCV, mis suudab vastu võtta läbi *VideoCapture* klassi videoedastuse [15]. Otse IP aadressi paigutamine *VideoCapture* klassi ei toiminud.

Videoedastusele läbi IP aadressi ei pääsenud. Juhendaja Taavi Kase leidis tarkvara nimega *IP Camera Viewer*, millega oli võimalik videoedastust näha. Ligi pääsemine tähendas seda, et kaamera pilti on võimalik näha arvutiga ning läbi Python'i skripti juhtimiseks on vaja leida õige aadress ja protokoll. *IP Camera Viewer* tarkvara määras protokolli ning aadressi ise ja seda otseselt läbi tarkvara näha polnud.

Autor proovis mitmeid erinevaid viise, kuidas kaameratele läbi Python'i juurde pääseda ja selle jaoks kulus mahukalt aega. Lahendus leidus läbi *IP Camera Viewer* tarkvara. Autor otsustas kasutada tarkvara nimega *Wireshark*, et jälgida, kuidas tarkvara *IP Camera Viewer* suhtleb kaameratega.

Wireshark on võrguprotokolli analüsaator, mida kasutatakse võrguliikluse analüüsimiseks ja jälgimiseks. See jäädvustab andmepakette pakkudes teavet liikluse kohta, mis toimub võrgus. Läbi Wiresharki on näha allika ja sihtkoha aadressid, kasutatud protokollid, paketi suurus ja paketis sisalduvad andmed [16].

Wiresharki abiga suutis autor jälgida, kuidas IP Camera Viewer tarkvara suhtleb kaameratega. *Wiresharki* kasutamiseks otsis autor pakette, mida saadetakse videokaamera IP aadressile. Joonisel 4.3 on näha, kuidas läbi RTSP protokolli toimub suhtlemine. Suhtluse sees on näha järgnevat aadressi: „rtsp://192.168.216.19:554 /ch0_0.h264". Selle aadressiga oli võimalik juurdepääs otse kaamera videoedastusele ja kasutada seda Python'i skriptis.



Joonis 4.3 *Wiresharki* kasutamine videoedastuse protokolli välja uurimiseks

Juhendaja Taavi Kasega sai kaamerate jaoks määratud kaameratele staatilised IP'd. Staatilised IP'd on aadressid, mis määratake DHCP-serveris vastavalt seadme MAC-aadressile. Seadme kadumisel võrgust või restardi tegemisel ei määrata IP'd teistele seadmetele. Staatilise IP'ga ei teki vajadust pidevalt skriptides muuta kaamera aadressi.


Joonis 4.4 valvekaamerate poolt nähtav parkla

Valvekaamerate vaatenurk sai valitud, et kattuvaid autosid ei tekiks ja kogu parkla oleks ära kaetud.

4.3 Andmestiku koostamine

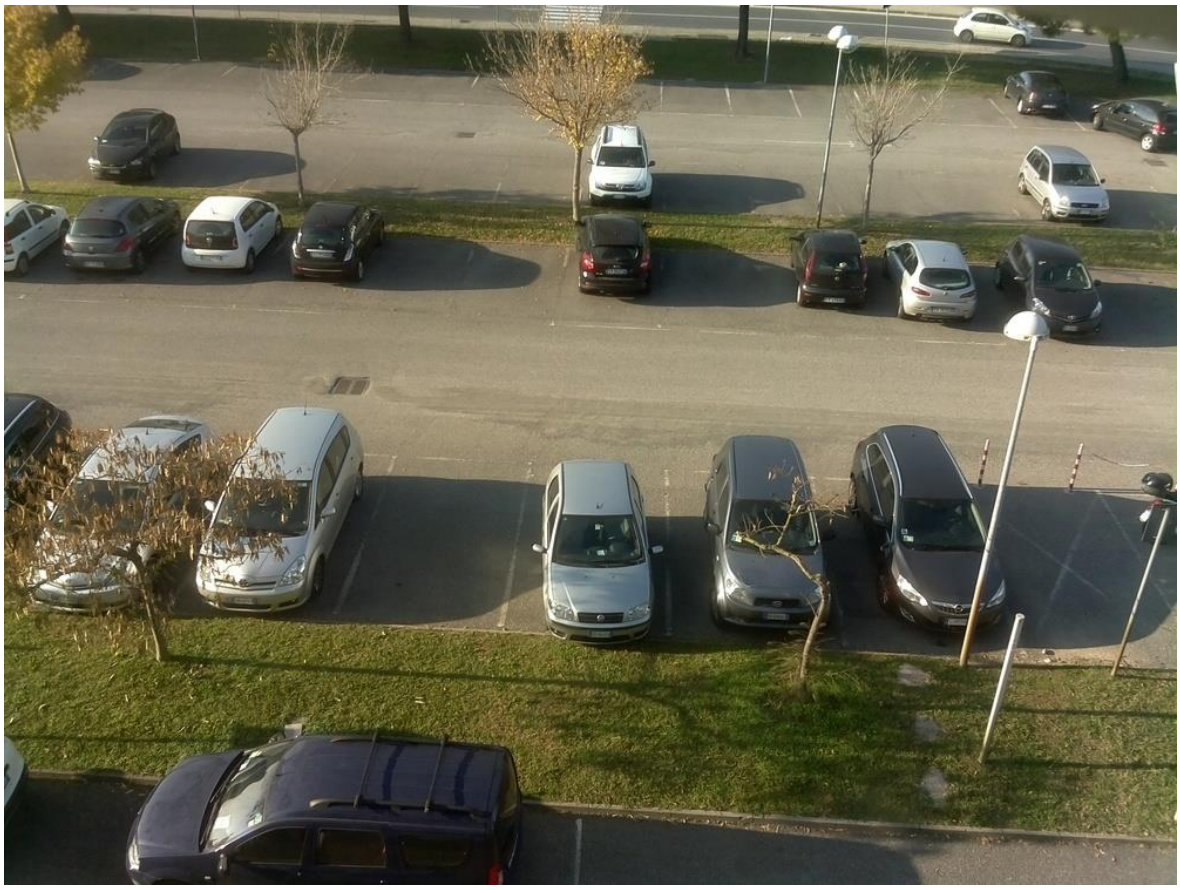
YOLO mudeli treenimiseks on vaja andmestikku. Andmestik sellises ülesandes on pildikogum, mis sisaldab erineva suurusega, valgustusega ja pildinurgaga auto pilte. Kõige tähtsam oli leida sarnase nurga all tehtud pildid. Kooli parkla kaamerad on paigutatud A korpuse teise korruse aknalaudadele. Andmestiku koostamiseks otsis autor varasemalt koostatud andmestike, mis andsid ülevaadet parklast sama nurga alt. Andmestikele lisaks otsis autor autoriõigusteta pildi andmebaasidest ise sobilike pilte

juurde. Autori poolt koostatud andmestik koosneb 84 pildist erinevates tingimustes olevatest autodest. 84 pilti ühe mudeli treenimiseks on tavaliselt liiga väike kogus ja ei piisa hea mudeli tegemiseks. Igal pildil on keskmiselt 18,9 autot ja iga markeeritud auto on eraldi õppepunkt mudeli jaoks. Suures koguses autode olemasolu igal pildil annab piisavalt õppepunkte mudeli treenimiseks. Kokku markeeris autor andmestikus ära 1580 autot.

Hiljem otsis autor mudeli parandamiseks pilte juurde ja samuti tegi autor mõningad pildid kooli parklast samast ruumist, kus kaamerad lõputöö raames paigutati. Selle tulemusel sai andmestikule juurde lisatud 40 pilti, millel oli kokkuvõtlikult 402 autot. Lõputöö raames paigutatud kaameraid polnud võimalik piltide tegemiseks kasutada, kuna mudeli treenimise ajal polnud kaamerad veel kohale jõudnud

26 parklapilti tuli andmestikust nimega CNR-EXT. CNR-EXT koosneb piltidest, mis koguti 2015. kuni 2016. aastatel erinevates ilmastikutingimustes. Andmestiku koostamiseks kasutati 9 erinevat kaamerat [17].

CNR-EXT andmestiku valis autor kaameranurga tõttu. Kaameranurk CNR-EXT andmestikus on peaaegu täpselt sama, mis on Tartu Kolledži parkla kaameratel. Kuna YOLO algoritmi treenimisel on väga tähtis õige nurga all olevad treenimisandmed, oli see andmestik väga sobilik lõputöö ülesande jaoks. Samuti oli andmestiku lisatud pildid koos talviste ja öiste tingimustega. Eesti ilma arvestades on kindlasti vajalik objekti tuvastusalgoritmi treenida ka talvistel ja pimedamatel tingimustel.



Joonis 4.5 Näide CNR-EXT andmestikust [17]

CNR-EXT andmestikuga tulid kaasa ka mitmed probleemid. Üks probleemidest oli, et CNR-EXT kasutab pilte markeeritud parklast. Andmestik koosneb kahte sorti piltidest– autoga parkimiskoht ja autovaba parkimiskoht.

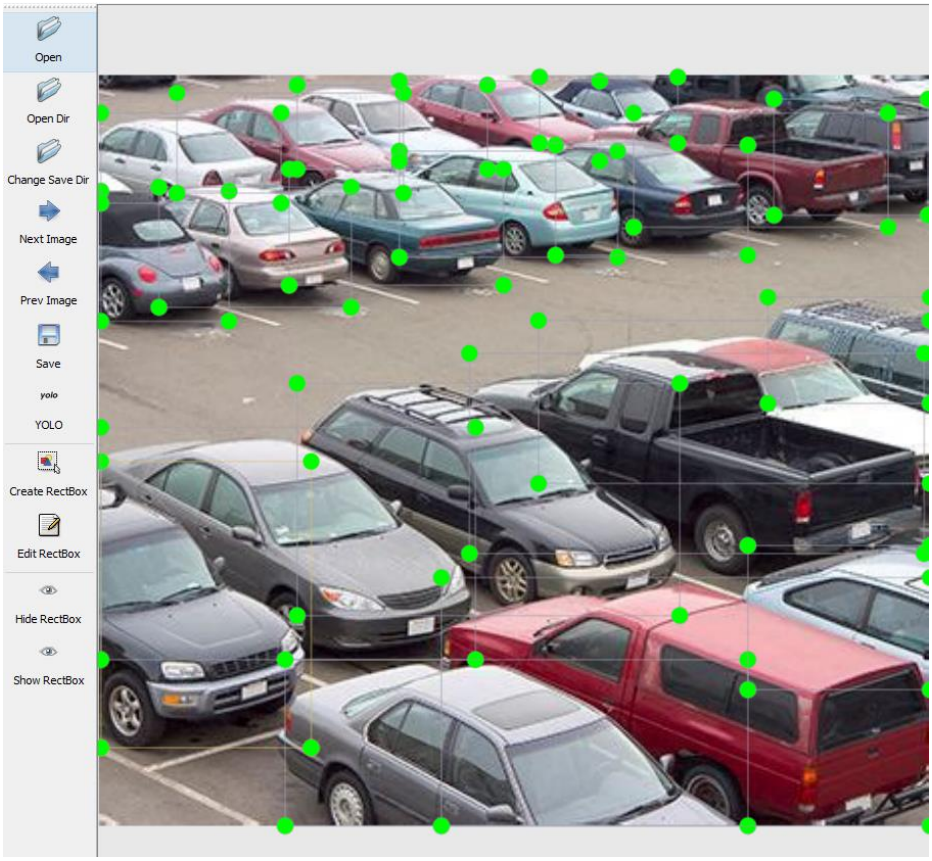
Teine probleem oli märgistatud andmestiku formaat. Tavaliselt on märgistatud andmestikel igal pildil vastav XML või TXT faili. Igas failis on ära märgistatud iga asjakohaliku objekti asukoht formaadis: **<objekti number> <objekti x koordinaat> <objekti y koordinaat> <laius><pikkus>**

Näitena on mõne objekti asukoht pildil: **0 0.193038 0.529536 0.170886 0.054852**

CNR-EXT andmestikus on standard formaadi asemel pildid jaotatud „lappideks" – igalt pildilt on võetud kõik autod ja välja kärbitud ainult auto pilt. Paljude objektituvastus algoritmide jaoks selline formaat sobib, kuid mitte YOLO jaoks. Andmestikuga on ka kaasa antud täismõõdus pildid. Andmestiku sobilikumaks tegemiseks võttis autor igast kaamerast 4 täismõõdus pilti ja kasutas neid enda andmestiku koostamisel.

4.3.1 Andmestiku märgistamine

Peale andmestiku koostamist oli vaja andmed märgistada. Mudeli treenimiseks on vajalik igal pildil ära märgistada auto asukoht. Piltide märgistamiseks kasutas autor tarkvara nimega Labelimg. LabelImg on graafiline pildi märgistamistarkvara. Labelimg on kirjutatud Pythonis ja kasutab graafilise liidese jaoks Qt raamistiku. Märgistused salvestatakse Yolo algoritmile sobivas formaadis TXT failina. [18]



Joonis 4.6 Labelimg tarkvara kasutamine, kus iga ruut märgistab ühte autot.

5. YOLO TREENIMINE JA PAIGALDAMINE

5.1 Vajalikud tarkvarad ja paigaldamine

5.1.1 Vajalikud tarkvarad

Süvaõppe treenimiseks ja kasutamiseks kasutas autor Windows 10 operatsioonisüsteemiga arvutit. Süvaõppe treenimist tegi autor kohalikul masinal ja ei kasutanud *Google Colab* keskkonda. *Google Colab* keskkonnas saab ilma tarkvarade installimata ja eeltööta kasutada ja treenida YOLO mudeleid. Otsus kasutada isikliku masinat lõputöö raames tulenes soovist läbida iseseisvalt kõik astmed. Selle tulemusel soovis autor paremini aru saada, kuidas objektituvastus toimib ja kuidas seda kasutada.

Süvaõppe kasutamiseks ja treenimiseks oli vaja paigutada arvutisse järgnevad tarkvarad:

- Cuda Toolkit
- CuDNN
- OpenCV
- CMake
- Visual Studio
- Darknet

CUDA (*Compute Unified Device Architecture*) ja CuDNN (*CUDA Deep Neural Network library*) on tööriistad, millega saab kiirendada Nvidia GPU-de süvaõppe ülesandeid. Treenimist tegi autor personaalsel arvutil, mis kasutab Geforce RTX3070 videokaarti. Mudeli treenimiseks kasutati CUDA versiooni 12.1 ja CuDNN v8.8.1 CUDA 12.x versiooni jaoks. Mõlemad tarkvarad on kättesaadaval Nvidia veebilehelt.

CuDNN pakub madala taseme ehitusplokkide komplekti, nagu konvolutsioonikihid ja aktiveerimisfunktsioonid. Ehitusplokkide komplekte kasutatakse süvaõppe arhitektuurides. CUDA ja CuDNN tarkvarad on mõeldud kasutamiseks ainult koos Nvidia videokaardiga. AMD kaartide jaoks on tarkvara nimega AMD ROCm. ROCm tarkvaraga on võimalik saavutada sarnaseid kiirusi. Soovi korral on ka võimalik treenida ja kasutada YOLO mudeleid arvuti protsessori peal, mille korral pole vaja paigaldada CUDA ega CuDNN tarkvara. Videokaardi puudumise korral on piiratud treenimise kiirus ja võimekus.

OpenCV on avatud lähtekoodiga arvutinägemise masinõppe tarkvarateek. OpenCV pakub suure valiku funktsioone ja algoritme piltide ja videote töötlemiseks. Samuti pakub see liideseid erinevatele kaameraseadmete ligipääsuks.

CMake on avatud lähtekoodiga ehitustööriistade generaator, mis on mõeldud C-programmeerimiskeele jaoks. Cmake genereerib ehitusfaile erinevate platvormide jaoks, sealhulgas Unix, Linux, Windows ja macOS.

Visual Studio on Microsofti arenduskeskkond. Tavaliselt kasutatakse seda arvutiprogrammide arendamiseks. Lõputöö raames kasutas autor Visual Studio arenduskeskkonda koos CMake tarkvaraga, et ehitada Darknet'i tarkvarateeki.

Darknet on avatud lähtekoodiga närvivõrgu raamistik, mis on kirjutatud C ja CUDA keeles. Darknet'i kasutatakse peamiselt objektide tuvastamiseks ja muudeks arvutinägemisega seotud ülesanneteks. Darknet toetab YOLO objektituvastussüsteemi ja on alus YOLO arendamisele.

5.1.2 Installimine

CUDA

Esmalt oli vajalik installida CUDA tarkvara arvutisse. CUDA installimiseks oli vaja läbida järgnevad sammud:

- Nvidia veebilehelt tuli valida *Cuda Toolkit*. Lõputöö raames kasutati Cuda 12.1 versiooni. Vanemaid versioone kasutades tuleb arvestada CuDNN versiooniga ja valida sobilik versioon.
- Seejärel tuli valida operatsioonisüsteem ja selle versioon. Valikutes on Windows ja Linux.
- Peale alla laadimist oli võimalik tarkvara installida arvutisse. Installimise käigus pidi arvestama CUDA juhendit.

Peale CUDA installimist oli vaja üle kontrollida Windowsi keskkonnas *Environment Variables* väärtused. *System variables* alt peab leiduma:

- CUDA_PATH C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1
- CUDA_PATH_V12_1 C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1

CuDNN

Enne CuDNN tarkvale ligipääsemiseks oli vaja nõustuda eetilise AI seadusega ja registreerida konto Nvidia keskkonnas. CUDA installimiseks oli vaja läbida järgnevad sammud:

- Nvidia veebilehelt tuli valida *CuDNN*. Lõputöö raames kasutati *CuDNN* versiooni 8.8.1 *CUDA* 12.1 jaoks. Installides vanemaid versioone peab arvestama kindlasti CUDA versiooniga ja valida sobilik versioon.
- Peale alla laadimist oli vaja *CuDNN* tarkvara lahti pakkida. *CuDNN* koosneb kolmest kaustast: *bin*, *include* ja *lib*. Need kolm kausta oli vaja üle kopeerida *CUDA* paigaldamise

asukohta. Tavapäraselt on see asukoht: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1.
- Peale CuDNN paigaldamist CUDA keskkonda on vaja teha Windowsi keskkonnas *Environment Variables* väärtused CuDNN jaoks.

System Variables alla pidi tegema uus muutuja nimega „CUDNN". Muutuja väärtusteks:

- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1;
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\bin;
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\include;
- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.1\lib\x64;

Samuti oli vaja eelnevalt lisatud 4 CuDNN muutujat väärtused lisada *User Variables* all leiduvale *Path* muutujale.

Peale arvutile restardi tegemist oli võimalik kontrollida CUDA ja CuDNN korrektset paigaldamist Windows konsoolis käskudega:

- nvidia-smi
- nvcc -V

Korrektse paigaldamise korral annab vastuseks konsoolis CUDA ja CuDNN versiooni

Darknet

Darknet leidub keskkonnast darknet'i *githubi* lehelt [19]. Peale alla laadimist polnud rohkem samme Darknet tarkvaraga vaja algselt teha. Hiljemalt oli vaja Darknet'i asukohta kasutada CMake tarkvaras.

Visual Studio

Visual Studio leidus keskkonnast *https://visualstudio.microsoft.com/*. Visual Studio keskkonnas on võimalik leida *Community 2022* versioon, mida kasutas autor lõputöö raames. Visual Studio *Installer* raames oli vajalik lisada Python ja C++ tarkvarateegid.

CMake

CMake leidus keskkonnast *https://cmake.org/*. CMake paigaldamise keskkonnas pidi valima vastav operatsioonisüsteem.

OpenCV

OpenCV leiab keskkonnast *https://opencv.org*. Lõputöö raames kasutati *OpenCV – 4.7.0* versiooni. OpenCV paigaldamiseks pidi valima *Windows* versiooni, mis installib automaatselt OpenCV tarkvara arvutisse. Peale paigaldamist oli vaja teha Windowsi keskkonnas Environment Variables OpenCV jaoks muudatused.

System Variables alla pidi lisama *PATH* väärtusele juurde OpenCV väärtused:

- C:\opencv\build\include
- C:\opencv\build\bin
- C:\opencv\build\x64\vc16\bin
- C:\opencv\build\x64\vc16\lib
- C:\opencv\build

Tarkvara ehitamine

Peale tarkvarade installimist oli võimalik hakata tarkvara koostama. Tarkvara koostamiseks oli esmalt vaja avada CMake tarkvara. Tarkvara koostamiseks läbiti järgmised sammud:

- *Where is the source code:* Darkneti asukoht.
- *Where to build binaries:* Darkneti asukoht.
- Vajutada *configure* ja täpsustada Visual stuudio versioon, näiteks Visual Studio Visual Studio 2022
- Vajutada Finish ja Generate.
- Avada Visual Studio ja *Debug* asemel peab valida *Release* ja *x64.*
- Vajutada *BUILD* ja *Build Solution.*

5.2 YOLO konfigureerimine

YOLO'l on erinevaid versioone ja lõputöö raames kasutati YOLOv4 versiooni Yolov4-tiny. Yolov4-tiny versioon rakendab teatud muudatusi, mis aitab paremaid kiirusi saavutada. Peamiseks muudatuseks on põhiahela konvolutsioonikihtide arv. Konvolutsioonikihtide arv on tihendatud kokku 29 eeltreenitud konvolutsioonikihiks. Lisaks on YOLO kihtide arvu vähendatud kolme asemel kahele ja ennustamiseks on vähem ankurkaste [19].

Uue mudeli treenimiseks oli lihtsam kasutada mudelipõhja. Mudelipõhi on eelkoolitatud mudel, mis on varasemalt treenitud tuvastama objekte. Mudelipõhjaks kasutati yolov4-tiny.conv.29 mudelit. Mudel on kättesaadaval YOLO Githubi veebilehelt. Mudeliga sai treenitud YOLOv4-tiny versioon, mis on sobilik väheste arvutusresurssidega masinatele.

Enne treenimise alustamist oli vajalik konfigureerida yolov4-tiny-custom.cfg fail, mis leidub cfg kaustast.

Cfg failis oli vaja teha järgmised muudatused:

- batch=64
- subdivisions=16
- width=416 ja height=416
- max_batches = 6000 ja steps väärtusega 4800,5400
- vaja leida 2 [yolo] kihti ja sellele eelnevates [convolutional] kihtides vaja muuta filters=18

- mõlemas [yolo] kihis oli vaja muuta classes=1

Teisi väärtusi on samuti võimalik konfigureerida, kuid soovituslik on teised väärtused algväärtusteks jätta.

Data kausta oli vaja tekitada failid

- *obj.names*. faili oli vaja kirjutada kõik klassid, mida YOLO tuvastama peab. Lõputöö raames oli kirjas ainult üks klass nimega *car*.
- *obj.data.*
- *test*.txt
- *train*.txt

obj.data viitab kõikidele failidele, märkides ära klasside koguse, mis kausta kasutatakse treenimisandmeteks, mis kausta kasutatakse testimisandmeteks ja kuhu uus mudel paigutatakse.

Test ja *train* tekstifailid sisaldasid kõikide treenimispiltide asukohtasid ja on jaotatud „*process*.py" skriptiga nii, et train pilte on 90% ja 10% on testimispilte. „*Process*.py" skript on nähtav lisade all [Lisa 2].

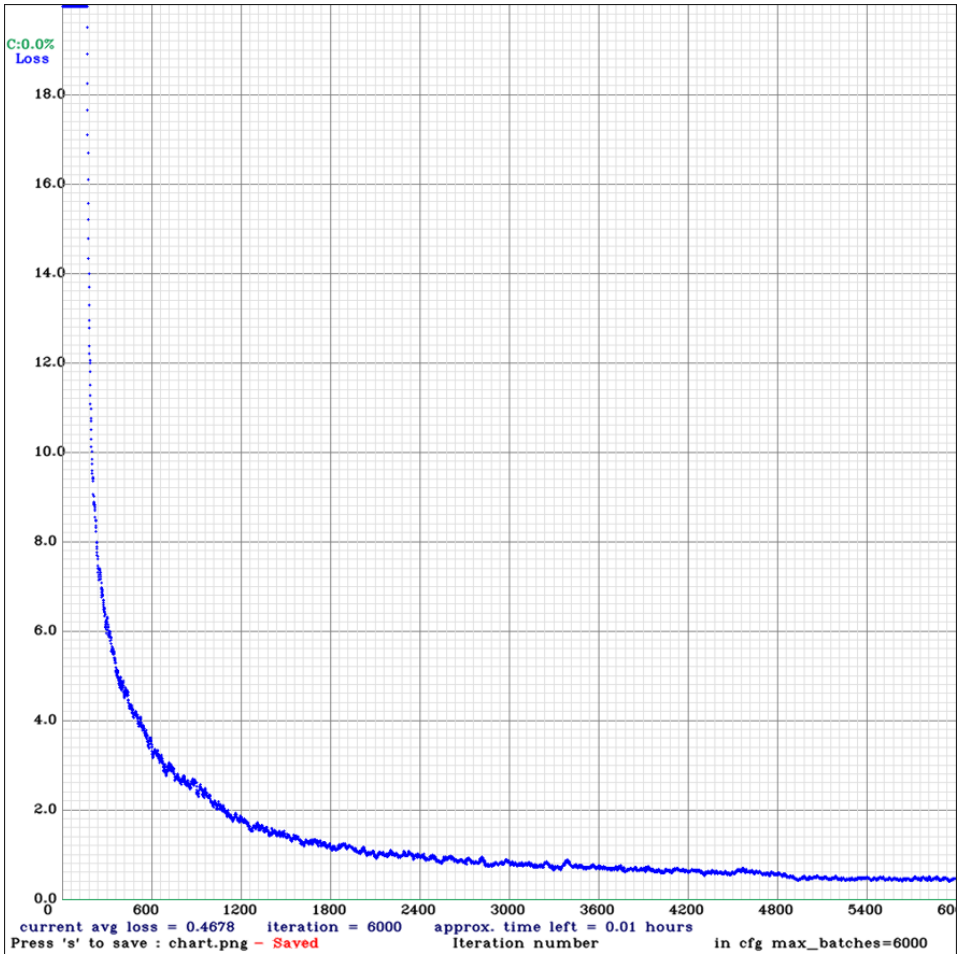
Treenimise alustamiseks oli vaja avada darknet kaustas konsool ja sisestada:

„darknet detector train data/obj.data cfg/yolov4-tiny-custom.cfg yolov4-tiny.conv"

5.3 Tulemused

YOLO mudeli treenimistulemusi mõõdeti *loss* ehk kao parameetriga. Kao parameeter näitab algoritmi töö edukust. Kadu parameetrit kasutatakse ka tagasiside signaalina, et kohandada algoritmi toimimist. Kadu funktsiooni kohandamist nimetataksegi õppimiseks.

Kao parameeter näitab, kui palju erinevad mudeli prognoosid tegelikest väärtustest. Mida väiksem on kao parameeter, seda paremad on tulemused objektide tuvastamisel pilditest [20].



Joonis 5.1 Treenimise tulemuste kadu esimene graafik.

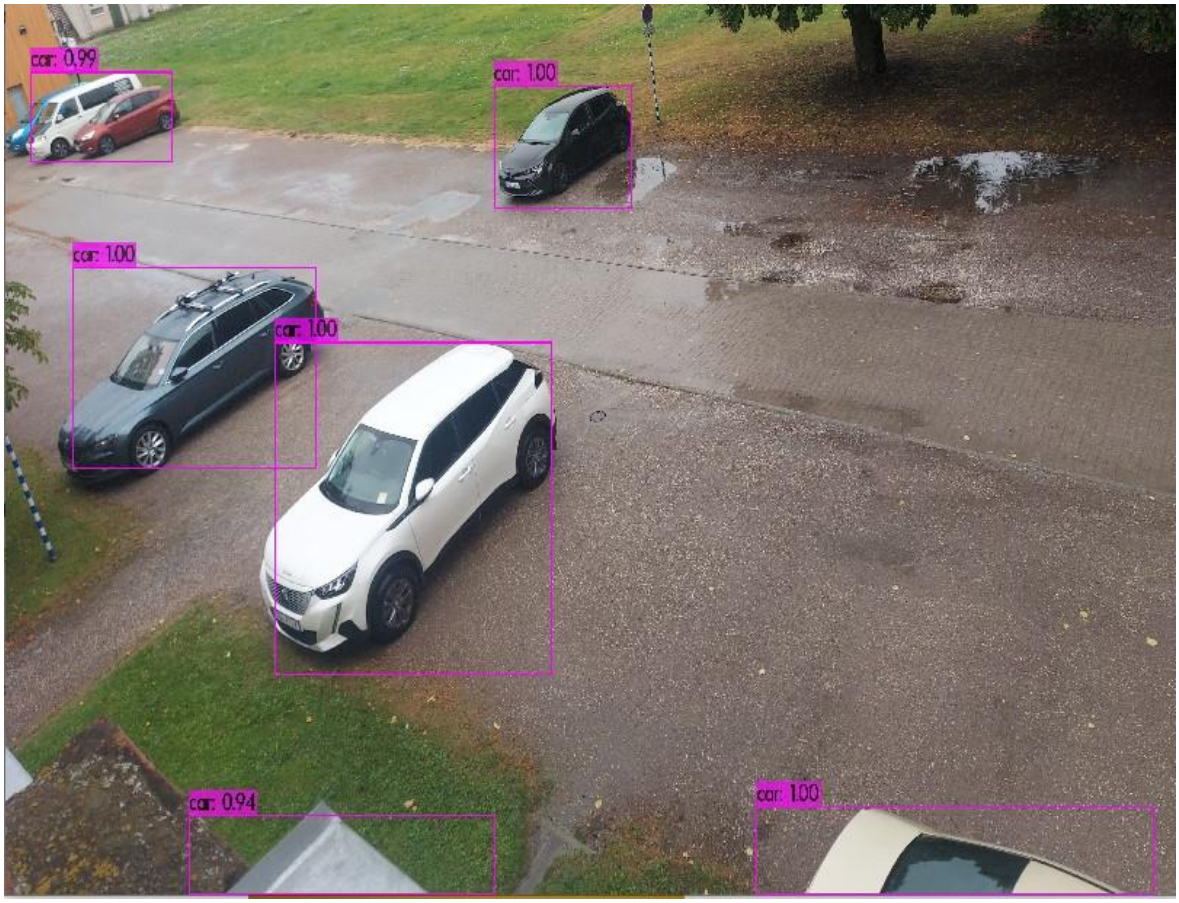
Treenimiseks kulus poolteist tundi ja selle kiirus tuleneb võimekast videokaardist. Graafikul on näha punkte ja iga punkt tähistab ühte iteratsiooni ehk kordust. Igas

korduses võrreldakse kadu funktsiooniga, kui hästi mudel testiandmete suhtes hakkama saab. Kadu vähenes 200 kordusega 20-protsendini. Peale 600 kordust hakkas kao parameeter üha aeglasemalt vähenema ja edaspidised kordused on peamiselt mudeli optimeerimised. Peale 6000 kordust on jõudnud kadu väärtus 0.47% peale.

Peale treeninguid sai mudelit kasutada autode tuvastamise testimiseks.

Joonisel 5.2 on näha, kuidas mudel korrektselt autosid tuvastab. Pildilt leidub üks valepositiivne tulemuse ja üks valenegatiivne tulemus. Pildi vasakul all servas leidub usaldusega 94% aknalaua nurk, mis tuvastatakse kui autot. See probleem tuleneb sellest, et andmestikus olid paljud autod pooleldi peidus või ainult üks nurk näha. Mudel on selle tõttu ära õppinud, et teravad nurgad pooleldi peidus tähendavad autot.

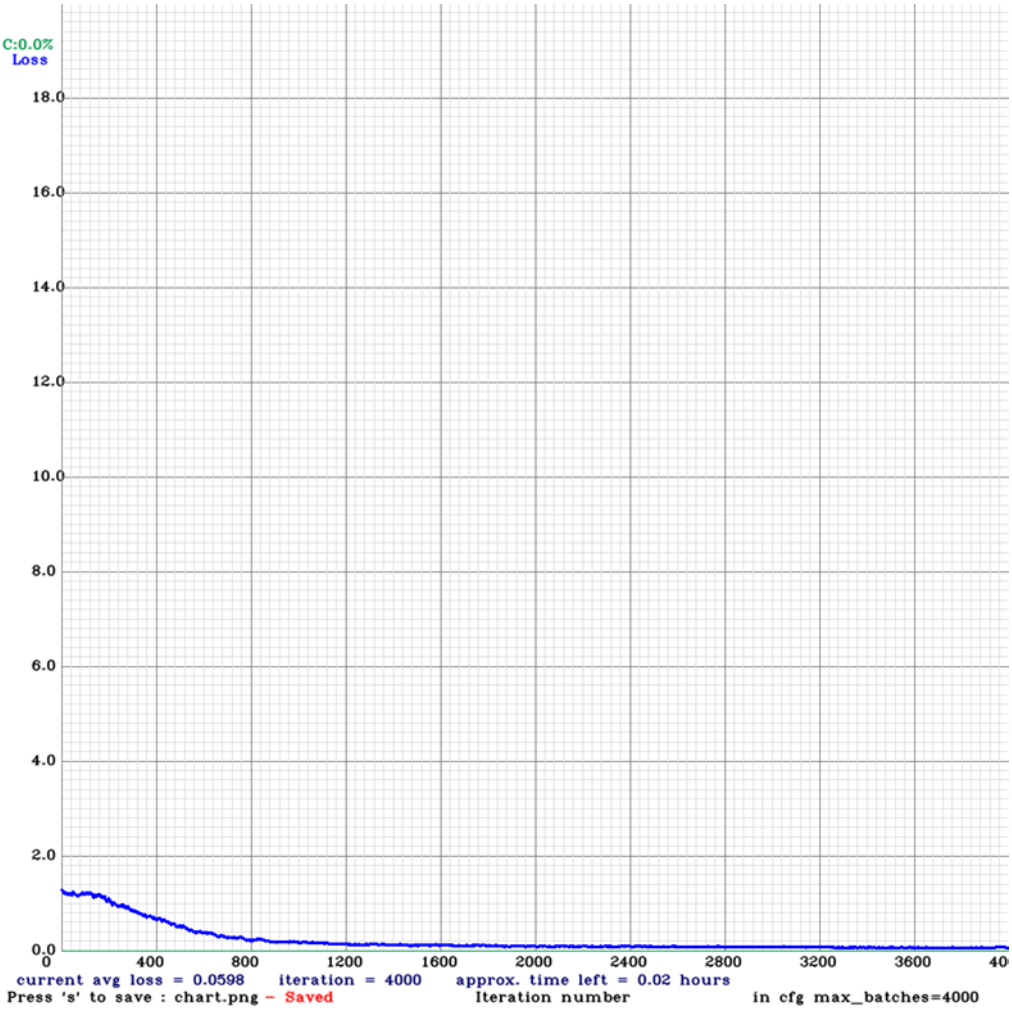
Valenegatiivne tuleneb sellest, et YOLO mudel ei oska väga hästi tuvastada pildilt väikeseid objekte. Pildil tekib situatsioon, kus üks auto on varjatud teise auto poolt ja mudelil tekib valenegatiivne. Joonise 5.2 üleval vasakul servas on näha kuidas kolmest autost on ainult üks tuvastatud.



Joonis 5.2 Mudeli testimine parklas tehtud pildiga

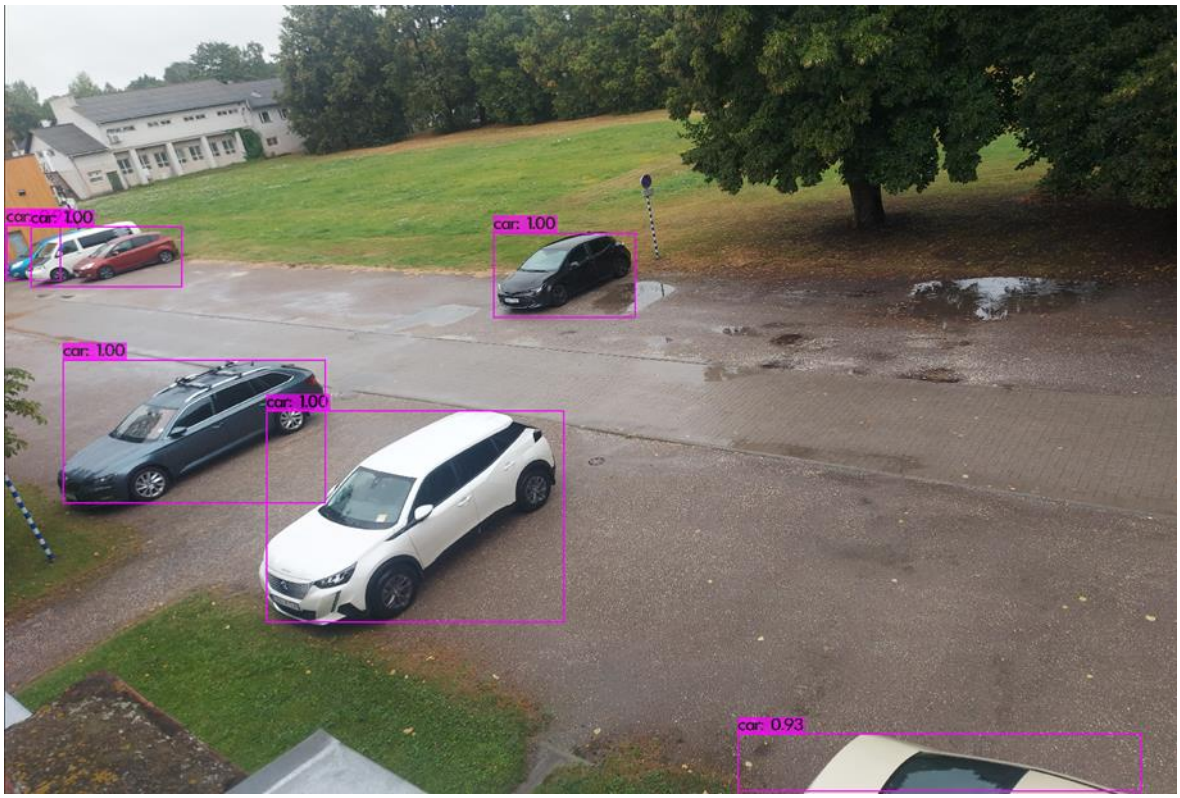
Tulemuste parandamiseks lisas autor testiandmetele juurde 40 pilti ja eemaldas andmestikust autoandmed sellistel juhtudel, kui auto on enamasti mittenähtav. Selle tulemusel soovis autor eemaldada valepositiivseid tulemusi.

Teine treening algas esimese treeningu lõpul saavutatud mudeli põhjal. Teise treeningu vältel suutis autor saavutada 0.06% kadu.



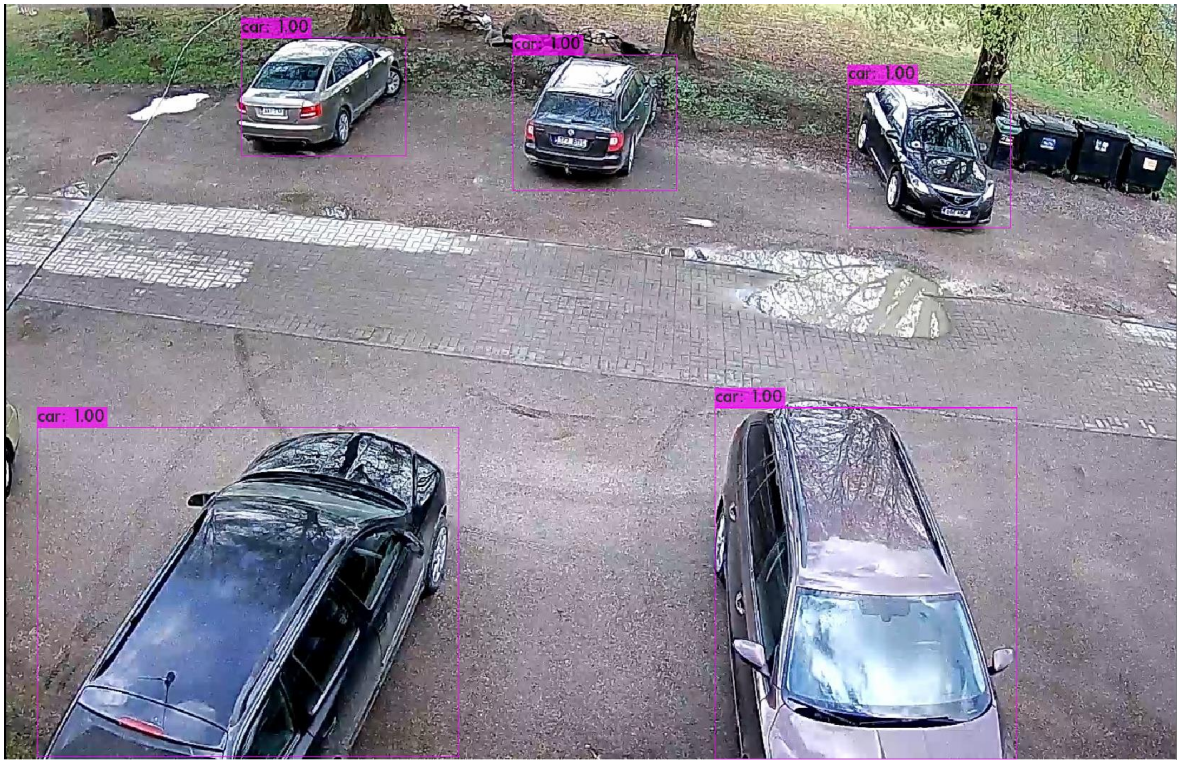
Joonis 5.3 Treenimise tulemuste kadu teine graafik

Peale teist treeningut kadus aknalaua nurga valepositiivne tulemus. Samuti suutis mudel paremini ära tuvastada üleval vasakul servas olevaid autosid – ühe auto asemel on tuvastatud kõik kolm autot.



Joonis 5.4 Mudeli testimine kooli parklas peale teist treeningut

Peale edukat tuvastamist oli võimalik kasutada mudelit autode tuvastamiseks kaamerapildist. Kaamerapildist tuvastamist on näha joonisel 5.5.



Joonis 5.5 Mudeli testimine kooli parklas peale teist treeningut

5.4 Kooliserveri ette valmistamine tuvastamiseks

Kooliserverile ligipääsemiseks oli vaja tekitada VPN (*virtual private network*) ühendus kooli serveriga. VPN ühenduse konfigureerimiseks kasutas autor Tallinna Tehnikaülikooli poolt pakutud juhendit [21]. Virtuaalmasina kooli serveris valmistas juhendaja Taavi Kase. Virtuaalmasin töötab Ubuntu operatsioonisüsteemil. Virtuaalmasinasse oli vajalik paigalda täpselt samad tarkvarad välja arvatud videokaardi rakendamise tarkvarad ja tarkvara ehitamise tarkvarad, ehk CUDA, CuDNN, Visual Studio ja CMake. Videokaardi rakendamise tarkvarasid pole videokaardi puudumise tõttu vaja. Lisaks on Ubuntu operatsioonisüsteemil kõik vahendid tarkvara ehitamiseks juba sisse ehitatud

Tarkvarade installimiseks oli vaja kasutada terminali ja sisestada järgnevad käsud:

- sudo apt update
- sudo apt install python3-pip
- git clone https://github.com/AlexeyAB/darknet.git
- pip install opencv-python

Peale vajalike tarkvarade paigaldamist oli vaja teha muudatused failis nimega *Makefile*, mis leidub darknet kaustas. Kuna serveril puudub videokaart oli vajalik teha *Makefile* failis järgnevad muudatused:

- GPU=0
- CUDNN=0
- CUDNN_HALF=0

Peale *Makefile* failis muudatuste tegemisi oli võimalik darknet kompileerida käskudega:

- cd darknet
- make

5.5 Algoritm autode tuvastamiseks ja loendamiseks

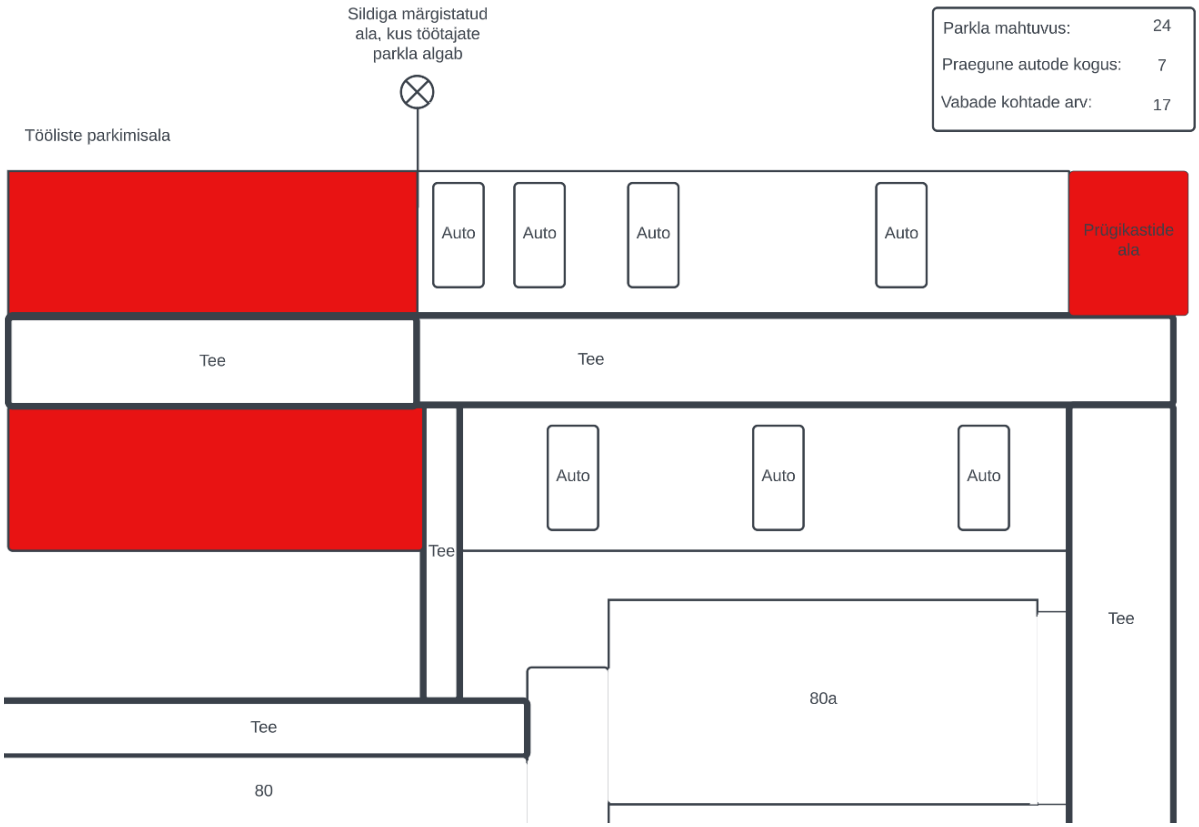
Algselt oli töö autoril plaan luua keerukam algoritm vabade parkimiskohtade leidmiseks parklast. Algoritm oleks arvutanud tuvastatud autode kaugused üksteisest ja samuti autode kaugused parkla äärtest. Selle tulemusel oleks võimalik olnud arvutada, kui palju autosid iga auto vahele mahub ja kui palju autosid auto ning parklaääre vahele mahub.

Eelnevalt kirjeldatud algoritmi loomisest autor loobus, kuna sellega kaasnesid mitmed keerukused. Peamised probleemid olid sobimatu kaamera nurk kauguse hindamiseks ja objektile joonistatud kasti ebatäpsus. Kaamera nurga probleem seisneb selles, et iga auto jääb erineva nurga alla kaamera suhtes ja kaugused on alati erinevad. Probleemi

saaks ära hoida paigutades kaamera täpselt 90-kraadise nurga alla parkla kohale, kuid seda võimalust kooli parklas kahjuks pole.

Objektile joonistatud kastidega tekib probleem, kui kast ei ole piisavalt täpne ja hõlmab rohkem kui tuvastatud objekti. Selle tõttu annaks kirjeldatud algoritm halba ettekujutust parkimiskohtadest. Kirjeldatud probleemide tõttu oli vaja leida teine ja lihtsam lahendus, mis sobiks kooli parkla puhul paremini.

Joonisel 5.6 on näha parkla skeemi. Peale autode tuvastust arvutatakse autode kogus ja lahutatakse saadud arv maha parkla mahtuvusest. Kirjeldatud süsteemil on samuti puudujäägid. Mõne auto valesti parkimise korral võib see auto rohkem kui ühe parkimiskoha kinni parkida ja algoritm edastaks vigase parkimiskohade arvu süsteemile. Parkimisala laius on umbes 64 meetrit, mille autor leidis manuaalselt mõõtes parkla laiust. Keskmine parkimiskoha laius on 2,6 meetrit [22] ja sellest saab arvutada parkla mahtuvuse: 24 parkimiskohta.



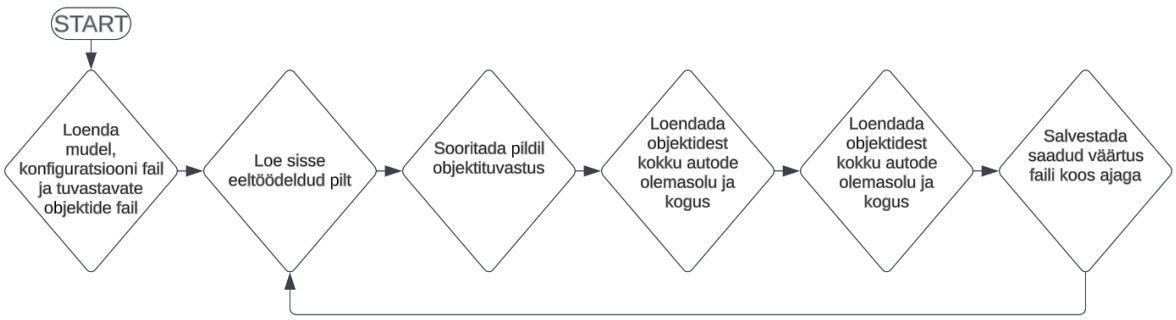
Joonis 5.6 Parkla skeem

5.6 Algoritmi rakendamine

Algoritmi rakendamiseks tegi autor Mitu Python'i programmi. Programmi erinevad osad jaotati mitmeks erinevaks osaks, et lihtsamini aru saada programmi tööst. Lisaks abistas osadeks jaotamine vigade leidmisega- tihtipeale sai programmi tegemise jooksul kooli serveril mälu otsa ja programm lõpetas töötamise. Selle jaoks, et välistada programmi täieliku lõppemise jaotas autor programmi erinevateks osadeks. Ühe programmiosa lõppemisel jätkub kogu programmi töö ja on võimalik eraldi alustada igat programmi osa. Programmide eesmärgid olid kaamerapildi lugemine, selle töötlemine sobilikule kujule, YOLO mudeli rakendamine darknet raamistikus, leitud autode pidev loendamine ja aja ning autode kogus väljastamine faili. Programm on jaotatud neljaks erinevaks osaks:

- Peaprogramm, mis juhib kõiki teisi programmi osasid
- Programm, mis võtab sisse kaamerapildid
- Programm, mis liidab kaamerapildid kokku ning hägustab kattuvad ja mitte tuvastuse alla kuuluvad alad
- Programm, mis teeb objektituvastust hägustatud pildil ja salvestab saadud väärtused faili

Kõige mahukam programm on objektituvastus programm. Programmi kood on nähtaval lisades 2 ja 3. Programmi voodiagrammi on näha joonisel 5.7.



Joonis 5.7 Objektituvastus programmi voodiagramm

Joonisel 5.8 on näha, kuidas tuvastus töötab. Hägustatud on kaks ala. Üks ala on õppejõudude parkimisala ja on leiduv vasakul kaamerapildil. Õppejõudude parkimisala on hägustatud, kuna see ala ei kuulu lõputöö raamesse ja ei vaja tuvastamist. Teine hägustatud ala on kahe kaamera kattuvusalal parempoolse kaamerapildi vasakul ülevas servas. Teine ala on hägustatud ülekatte tõttu, ehk mõlemad kaamerat näevad osaliselt sama ala. Programm on nähtaval lisades (Lisa 3).

Joonis 5.8 Python'i programm, mis tuvastab autode olemasolu parklas ja loendab autosid

5.7 Tulemuste analüüs

Lõputöö raames koostatud objektituvastus süsteem toimib edukalt. Kaamerate vaatepildid katavad kogu tagumise kooli parkla ala ja vajaduse korral saaks samasuguse lahenduse ehitada eesmise parkla alale. Autode tuvastus on väga täpne ja suudab edukalt kõik olemasolevad autod tuvastada. Autode kogus liidetakse korrektselt kokku ja saadetakse edasi faili koos ajaga, millal autod tuvastati. Faili sisestatud informatsiooni alusel on võimalik hiljemalt teha statistikat. Statistika põhjal oleks näha, millal on parkla kõige rohkem täidetud ja kuna on tühi.

Samuti on võimalik saadud arv kättesaadavaks teha tudengite jaoks kaugelt, et oleks võimalik jälgida, kui täidetud on kooli parkla. Selline süsteem oli plaanitud koos teise tudengi lõputööga, mille raames salvestatud autode kogus oleks läbi telefoni rakenduse kättesaadav. Rakendus on veel tegemisel ja selle valmimise korral on võimalik ühendus autode tuvastussüsteemi ja telefoni rakendusega tekitada.

Süsteemil on mitmed puudujäägid. Süsteem hindab ainult autode olemasolu ja ei suuda arvestada, kui palju parkimiskohti on reaalselt parklas. Kuna tegemist on objektituvastusega, on samuti võimalus, et auto ei tuvastata teatud tingimustel. Auto tuvastamata jätmisel ei esita süsteem õiget parkimiskohtade arvu. Samuti ei välista süsteem juhuseid, kui auto on pargitud valesti. Valesti parkimise korral võib üks auto parkida kinni rohkem kui ühe parkimiskoha ja süsteem jällegi ei esita õiget

parkimiskohtade arvu. Lisaks jääb serveril puudu operatiivmälust, kuna kahe videopildi töötlemine ja edastuselt objektide tuvastamine nõuab rohkem mälu kui antud on.

Koostatud süsteem toimiks paremini kui kooli parkla oleks markeeritud. Markeeritud parklas on programmis võimalik määrata igale alale parkimiskoht ning jälgida, kas sellel alal on auto või mitte.

Tänaseks on valminud töötav prototüüp, kuid peab arvestama mitmete puudujääkidega. Süsteemi poolt esitatud autode arvuga saab hinnata kui täitunud parkla on, kuid arv ei pruugi olla kõikidel juhtudel täpne. Olenemata sellest annab saadud autode kogus tavakorral piisavalt täpse arusaama parkla täituvuse kohta. Sarnane lahendus oleks kergelt rakendatav teistes olukordades, näiteks kui on vaja tekitada ajutine parkla muruplatsile. Sellises situatsioonis oleks võimalik paigaldada ajutised kaamerad parkimisplatsile ja kaamerapilt edasi saata kas serverile või SBC-le (*Single-board Computer*).

KOKKUVÕTE

Käesoleva lõputöö eesmärgiks oli valmistada prototüüp parkimise parandamiseks Tallinna Tehnikaülikooli Tartu Kolledži parklas. Antud parklas oli probleeme teatud oludes vabade parkimiskohtade leidmisega ja antud lahendus proovis seda lihtsustada.

Lõputoo teoreetilises osas uuris autor erinevaid meetodeid, mida on kasutatud parkimise parandamiseks. Eelnevalt tehtud lahendustest kasutas autor kõige sobivamat. Tutvustatakse, kuidas toimib masinõpe, süvaõpe ja mis on süvaõppe täpsemad kihid ning selle toimimised. Lisaks tuuakse välja erinevad objekti tuvastus algoritmid ja antakse lihtne ülevaade toimimis põhimõtetest.

Lõputöö empiirilises osas antakse ülevaade algoritmidest objektide tuvastamiseks ja lähtudes piirangutest valis autor parima algoritmi. Autor valmistas ette vajaliku keskkonna objektituvastus mudeli treenimiseks ja annab ülevaate vajalikest tarkvaradest ja nende paigaldamisest. Keskkonnas valmistatakse objekti tuvastussüsteemi YOLO algoritmi baasil. Peale treenimist paigaldas autor kooli serveris olevale virtuaalmasinale vajalikud tarkvarad ja paigaldas lõputöö raames valminud mudeli. Parklas olevate autode tuvastamiseks paigaldasid autor ja juhendaja kooli A korpuse hoonele kaamerad. Kaamerad on ühendatud kooli Wi-Fi'ga ja saadavad videopildi edasi kooliserverile.

Lõputöö tulemusena valmistati autode tuvastussüsteem, mis hõlmab kaameraid, kooliserverit, objekti tuvastus mudelit ja objektituvastus algoritmi. Tuvastussüsteem sai valmistatud Tehnikaülikooli Tartu Kolledži tudengite parkla jaoks. Samuti koostas autor lihtjuhendid mudeli uuesti treenimiseks ja vajadusel süsteemi uuesti loomiseks. Treenitud mudelile tehti ka analüüsid objektide tuvastamise edukusest ja tulemuste parandamiseks tehti lisatreeninguid.

Lõputöö eesmärk saavutati ja koostatud süsteem soovitud raames toimib korrektselt. Süsteemil on mitmed puudujäägid ja erijuhtudel võib süsteemi poolt esitatud parkimiskohtade arv erineda reaalsest parkimiskohtade arvust. Koostatud süsteemi on väga kerge rakendada ka teistes oludes, näiteks eesmise parkla puhul või ajutise parkla autode tuvastamiseks. Olenemata puudujääkidest usub autor, et loodud süsteem on kasulik ja süsteemil on kasulik väljund.

SUMMARY

The aim of this thesis was to prepare a prototype for improving the parking in the Tallinn University of Technology Tartu College parking lot. The need to improve parking resulted from constant problems in the student parking lot which the prototype tries to solve.

In the theoretical part of the thesis, the author studied various methods that have been developed to improve parking. From the previously made solutions, the author researched the most suitable methods and applied them in the framework of the thesis. The thesis introduces how machine learning and deep learning work and what the layers of deep learning are and how they work.

The empirical part of the thesis provides an overview of the best algorithms for object detection and chooses a suitable one considering the circumstances. The author prepared the necessary environment for training the object recognition model and provides an overview of the necessary software and their installation. In the environment, an object recognition system is made based on the YOLO algorithm. After the training, the author installed the necessary software on the virtual machine on the school's server and installed the model that was made during thesis. To detect cars in the parking lot, the author and the instructor installed cameras on the school building. The cameras are connected to the school Wi-Fi and forward the video image to the school server.

As a result of the thesis, a car recognition system was made, which includes cameras, a school server, and an object recognition algorithm. The detection system was made for the student parking lot of Tartu College of the University of Technology. The author also created simple instructions for retraining the model and re-creating the system if necessary. Analyses of object recognition success were also performed on the trained model, and additional training was performed to improve the results.

The goal of the thesis was achieved and the created system functions correctly within the desired framework. The system has several shortcomings, and in special cases the number of parking spaces provided by the system may differ from the real number of parking spaces. The created can be integrated into different situations, for example the front facing car park or another temporary car park, where cars must be detected. Regardless, the author believes that the created system is useful and that there is a useful output from the system.

KASUTATUD KIRJANDUSE LOETELU

[1] Smartparking. arukas parkimissüsteem. [www] https://www.smartparking.com/smartpark-system/smart-sensors (23.03.2023)

[2] Hansab. Parkimislahendus Tartu kaubamaja parklas. [www] https://www.hansab.ee/et/parkimislahendus-tartu-kaubamaja-parkimismajas (23.03.2023)

[3] Deepsolutions . automatiseeritud parkimissüsteem. [www] http://deepsolutions.io/solutions/deep-parking/ (04.04.2023)

[4] IBM. Machine Learning for Dummies. [Online] https://www.ibm.com/downloads/cas/GB8ZMQZ3 (04.04.2023)

[5] Datasci. Masinõppe sõnastik. [www] http://datasci.ee/masinoppe-sonastik/ (04.04.2023)

[6] IBM. Süvaõpe, mis on süvaõpe. [www] https://www.ibm.com/topics/deep-learning (04.04.2023)

[7] Pythonistaplanet. Süvaõppe eelised ja puudujäägid. https://pythonistaplanet.com/pros-and-cons-of-deep-learning/ (04.04.2023)

[8] Researchgate. Workflow of ML and DL [Online] https://www.researchgate.net/figure/Workflow-of-ML-and-DL_fig2_349723503 (04.04.2023)

[9] Opengenus. Süvaõppe kihid. [www] https://iq.opengenus.org/purpose-of-different-layers-in-ml/ (01.04.2023)

[10] Viso AI. Object Detection in 2023: The Definitive Guide. [www] https://viso.ai/deep-learning/object-detection/ (01.04.2023)

[11] Arxiv. Arvutinägemine ja mustrituvastus, R-CNN selgitus. [www] https://arxiv.org/abs/1311.2524# (28.02.2023)

[12] Arxiv. Arvutinägemine ja mustrituvastus, Kuidas SSD algoritm toimib. [www] https://arxiv.org/abs/1512.02325 (28.02.2023)

[13] Arxiv. Arvutinägemine ja mustrituvastus, Kuidas YOLO algoritm toimib. [Online] https://arxiv.org/abs/1506.02640 (28.02.2023)

[14] Arxiv. Object Detection in 20 Years: A Survey [Online] https://arxiv.org/pdf/1905.05055.pdf (12.04.2023)

[15] OpenCV. OpenCV, Videocapture klass. [www] https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html (02.05.2023)

[16] Wireshark. pakettide jälgimise tarkvara. [www] https://www.wireshark.org/ (02.05.2023)

[17] Cnrpark. Andmestik autoparklast. [www] http://cnrpark.it/ (18.12.2022)
[18] Github. Andmestiku märgistamise tarkvara. [Online] https://github.com/heartexlabs/labelImg (02.04.2023)
[19] Github. Yolov4-tiny mudeli täpsustused. [Online] https://github.com/AlexeyAB/darknet (12.04.2023)
[20] Towardsdatascience. Mis on kadufunktsioon. [www] https://towardsdatascience.com/what-is-loss-function-1e2605aeb904 (12.04.2023)
[21] Confluence Taltech. VPN seadistamise õpetus, TalTech. [www] https://confluence.ttu.ee/it-info/arvuti-ja-oppetoeoekoht-workplace-services/kauguehendus-vpn-remote-working-vpn/kauguehendus-forticlient-vpn (01.05.2023)
[22] Parkimisjoon. Parkimiskoha laius. [www] https://parkimisjoon.ee/et/teenused/parklate-joonimine/parkimiskoha-laius-ja-liiklusonnetused

LISAD

Lisa 1 Lõputöö raames kasutatud valvekaamerate parameetrid

- välimus: ilmastikukindel
- mikrofon: jah
- võimalus paigaldada mis tahes tasapinnale: jah
- resolutsioon: 2Mpx, «Full HD"» 1080p (1920x1080) / VGA
- IP66
- horisontaalne resolutsioon: 1200
- toetatud „WiFi" standard: 2,4G
- toetab: „Android" ja IOS
- 3,6 mm objektiiv
- infrapunakiirte kaugus: 20m
- toiteallikas: vooluvõrk / 12V / 1A
- programm: jah, laadige alla QR
- Mõõdud: 6/6 / 10,5 cm
- jala laius 7cm
- kaal: 0,348 kg
- pakendi kaal: 0,543 kg

SUJUV JA TÄPNE PILT - välikaamera paistab silma kõrglahutusega pildiga (eraldusvõime: 1920 x 1080), nii et saate hõlpsasti näha palju üksikasju.

Lisa 2 Python'i skript testi ja treeningu andmete jaotamiseks

```
import os
import random

# Set the directory containing the images
image_dir = "newdataset"

# Set the percentage of images to be used for the test set
test_percent = 10

# Create the train and test text files
with open("train.txt", "w") as train_file, open("test.txt", "w") as test_file:

    # Get a list of all the image file names in the directory
    image_files = [f for f in os.listdir(image_dir) if f.endswith(".jpg")]

    # Randomly shuffle the image file names
    random.shuffle(image_files)

    # Calculate the index to split the list of image file names
    split_index = int(len(image_files) * (1 - test_percent / 100))

    # Write the image file names to the train and test text files
    for i, image_file in enumerate(image_files):
        if i < split_index:
            train_file.write(os.path.join(image_dir, image_file) + "\n")
        else:
            test_file.write(os.path.join(image_dir, image_file) + "\n")
```

Lisa 3 Python'i programm autode tuvastamiseks, loendamiseks ja andmete väljastamiseks faili

```
import cv2
import numpy as np
import datetime

net = cv2.dnn.readNet('yolov4-tiny-custom.weights','cfg/yolov4-tiny-custom.cfg')
classes = []
with open('data/obj.names', 'r') as f:
    classes = f.read().splitlines()

# Read the input image
image_path = 'input_image.jpg'
img = cv2.imread(image_path)
if img is None:
    print('Error loading image data')
    exit()

height, width, _ = img.shape

blob = cv2.dnn.blobFromImage(img, 1/255, (416, 416), (0, 0, 0), swapRB=True, crop=False)
net.setInput(blob)
output_layer_names = net.getUnconnectedOutLayersNames()
layerOutput = net.forward(output_layer_names)

boxes = []
car = 0
confidences = []
class_ids = []

for output in layerOutput:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            x = int(center_x - w/2)
            y = int(center_y - h/2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)

font = cv2.FONT_HERSHEY_SIMPLEX
colors = np.random.uniform(0, 255, size=(len(boxes), 3))

for i in indexes.flatten():
    labels = str(classes[class_ids[i]])
    if labels == 'car':
        car += 1

with open('autod.txt', 'a') as f:
    f.write(f'{datetime.datetime.now()}: {car}\n')

for i in indexes.flatten():
    x, y, w, h = boxes[i]
    label = str(classes[class_ids[i]])
    confidence = str(round(confidences[i], 1))
    color = colors[i]
    cv2.rectangle(img, (x, y), (x+w, y+h), [0, 255, 0], 2)
    cv2.putText(img, 'Autode kogus' + ":" + str(car), (20, 20), font, 0.8, (0, 0, 0), 2)

cv2.imshow('Image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```