TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Siret Jorro
Alexander Frolov

Süvanärvivõrkude abil kaugseire pildiandmetest objektide tuvastamine ja integreerimine statistilise analüüsi kasutajaliidesega

Bakalaureusetöö

Juhendaja: Evelin Halling
PhD
Martin Simon
MSc

Tallinn 2020

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Siret Jorro, Alexander Frolov

25.05.2021

Annotatsioon

Töö eesmärgiks on arendada prototüüp rakendusele, mis süvanärvivõrkude abil kaugseire pildiandmetest objekte tuvastab ning tuvastatud objektide kohta andmeid kogub. Lisaks on eesmärk arendada võimekus rakenduse integreerimiseks statistilise analüüsi kasutajaliidesega. Samuti uuritakse töös, kas on võimalik treenida olemasolevast süvanärvivõrgu mudelist paremaid tulemusi näitav mudel, mida oleks võimalik prototüübis kasutada.

Töö käigus arendatakse API, mis võimaldab rakendusse pildiandmeid saata ning kust on võimalik pildi- ning tuvastusandmeid pärida. Samuti treenitakse töö käigus korduvalt süvanärvivõrgu mudelit, muutes igal treenimisel hüperparameetreid ning uurides muutuste mõju mudeli näitajatele.

Töö tulemusena valmis prototüüp, mis on tulevikus võimalik integreerida statistilise analüüsi kasutajaliidesega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 41 leheküljel, 11 peatükki, 29 joonist, 11 tabelit.

Abstract

Object Detection from Remote Sensing Data with Deep Neural Network and Integration with Statistical Analysis User Interface

The main goal of this thesis is to develop a prototype of an application which with the help of deep neural networks could be used for object detection from remote sensing data and later could provide data for statistical analysis. Additionally, development of the possibility of integrating the application with the statistical analysis user interface is a big part of this thesis. The thesis also involves a research of whether it is possible to train a deep neural network with better performance than that of the existing network, which could also be used in the application.

As part of the thesis an API is developed which allows the application to receive remote sensing data and request images or detections made by the neural network. The deep neural network model is trained with different hyperparameter configurations and an analysis of the impact on overall model metrics is conducted.

As a result of this thesis, a prototype is made which can later be integrated with the statistical analysis user interface.

The thesis is in Estonian and contains 41 pages of text, 11 chapters, 29 figures, 11 tables.

Lühendite ja mõistete sõnastik

AI	*Artificial Intelligence*, tehisintellekt
API	*Application Programming Interface*, rakendusliides
Base64	Kodeerimissüsteem, mis teisendab baidijada ASCII kirjamärkideks [1]
COCO	*Common Objects in Context*, populaarne andmestik, mis koosneb mitmetest tavapärastest objektidest
CPU	*Central Processing Unit,* protsessor
GET	Päringumeetod serverist andmete pärimiseks
GPS	*Global Positioning System*, globaalne positsioneerimise süsteem
GPU	*Graphics Processing Unit,* graafikakaart
JSON	*JavaScript Object Notation*, lihtne andmevahetusvorming, kus andmed esitatakse sarnaselt JavaScripti programmeerimiskeeles kasutatavatele objektidele lihtsate võtme-väärtus paaridena [2]
LR	*Learning Rate*, närvivõrgu hüperparameeter, mis reguleerib selle õppimise kiirust
ORM	*Object-Relational Mapping*, objekt-relatsioonvastendus, tehnika, mis võimaldab andmebaasis andmebaasis andmeid otsida ja käidelda objektipõhistes keeltes [3]
POST	Päringumeetod serverisse andmete saatmiseks
RAM	*Random Access Memory,* arvutimälu
REST	*Representational State Transfer*, tarkvara arhitektuuri stiil veebiteenuste arendamiseks
RGB	*Red, Green, Blue*, värvimudel, kus kõik värvitoonid saadakse kolme põhilise värviga (punane, roheline ja sinine). Enamlevinud viis värviliste piltide näitamiseks ekraani peal [4]
SSD	*Single Shot Detector*, sügavnärvivõrkude arhitektuur objektide tuvastamiseks reaalajas, kus kõik tuvastused tehakse ühe lasuga [5]
TPU	*Tensor Processing Unit*, Google'i poolt arendatud protsessorid masinõppe algoritmide täitmiseks [6]
URL	*Uniform Resource Locator,* üldine infoallika asukohamääraja, veebiaadress

WSGI	*Web Server Gateway Interface*, spetsifikatsioon, mis kirjeldab, kuidas peaks veebiserver suhtlema ülejäänud veebirakendustega ning mis viisil on võimalik ühendada veebirakendused päringute töötlemiseks [7]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Kaugseire pildiandmetelt on võimalik koguda erinevat kasulikku infot. Samuti on võimalik nende andmete põhjal viia läbi statistilist analüüsi, näiteks objektide liikumiste kohta. Pildiandmeid on aga väga palju ning nende käsitsi läbi töötlemine ning neilt objektide otsimine oleks väga ajakulukas. Seega oleks abiks rakendus, mis pildiandmeid töötleb, neilt huvipakkuvaid objekte tuvastab ning infot tuvastuste kohta talletab, et siis nende andmete põhjal analüüsi võimalik läbi viia oleks.

Selle töö eesmärk ongi luua prototüüp sellisele rakendusele, mis näitaks ära, kas sellise süsteemi ehitamine on võimalik ning kas sellega tasub edasi minna.

2 Ülesande püstitus

Töö eesmärgiks on luua prototüüp rakendusele, mis on võimeline vastu võtma kaugseire pildiandmeid ning süvanärvivõrgu abil neilt objekte tuvastama. Tuvastuse käigus on tarvis koguda huvipakkuvate objektide kohta informatsiooni, millega oleks hiljem võimalik läbi viia analüüsi, mis omakorda toimub statistilise analüüsi moodulis. Informatsiooni hulka kuuluvad näiteks objekti keskpunkti GPS (*Global Positioning System*) koordinaadid ning aeg, mil pilt oli tehtud. See, kuidas AI (*Artificial Intelligence*) mooduli poolt kogutud andmeid talletatakse, otsustakse juba statistilise analüüsi moodulis.

Lisaks tuvastuse käigus saadud andmete edastamisele analüüsi moodulise, peavad pildid jõudma ka kasutajaliidesesse, kus on võimalik vaadata kindlal ajahetkel tehtud tuvastust.

Pildiandmete edastamine AI moodulisse peab toimuma API (*Application Programming Interface*) kaudu. API peab olema suuteline vastu võtma pildi enda ning sellega koos käivad metaandmed, mille hulka kuuluvad pildi vasaku ülemise ja parema alumise nurkade GPS koordinaadid ning aeg, mil pilt oli tehtud. Süsteem peab iga hetk olema valmis uusi andmeid vastu võtma ning neid vastavalt töötlema.

Lisaks mainitule on oluliseks osaks olemasolevate teadmiste ja katsetuste põhjal jätkata eksperimenteerimist närvivõrkude treenimisega kaugseire piltide pealt objektide efektiivseks tuvastamiseks. Oodatavaks tulemuseks selles osas on uue ja võrreldes olemasoleva närvivõrguga parema võrgu arendamine.

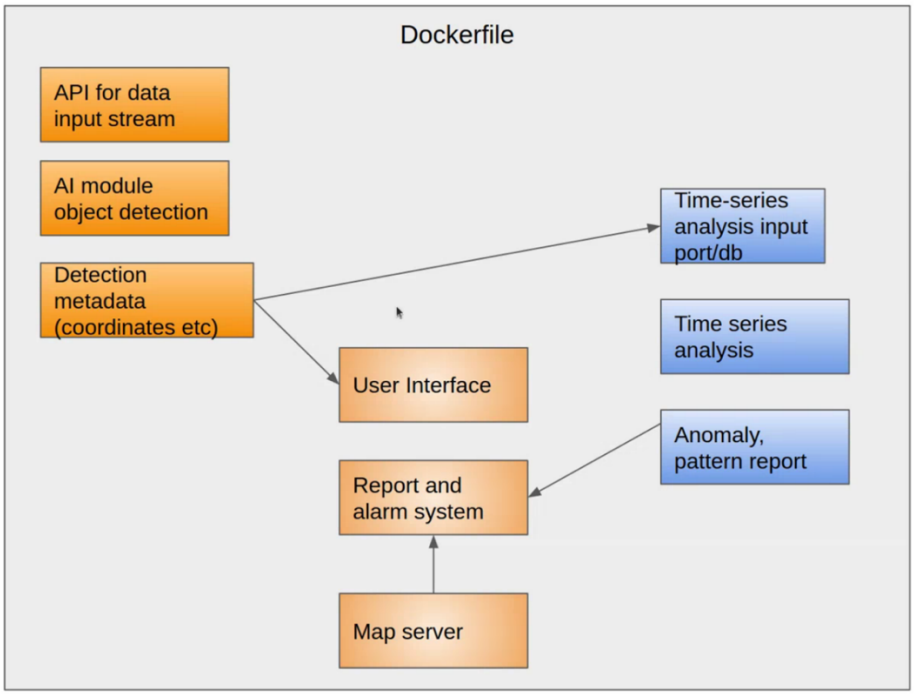
3 Projekti kirjeldus

Projekt kujutab endast suuremale süsteemile prototüübi välja arendamist. Süsteem tervikuna peab võimaldama automatiseeritult vastu võtma kaugseire pildiandmeid, mis teoreetiliselt tulevad erinevatelt satelliitidelt, need masinõppe abil läbi töötlema ja analüüsima ning lõpuks kasutajaliideses välja kuvama analüüsi tulemused, mille põhjal inimene oleks suuteline tegema otsuseid.

Prototüübi peamiseks eesmärgiks on ära näidata, et olemasolevate kaugseire pildiandmete pealt on võimalik teostada efektiivselt tuvastusi ning nende põhjal statistilise analüüsi algoritmide rakendamisel koguda kasulikku informatsiooni, näiteks objektide liikumiste kohta. Juhul kui see õnnestub, on teoreetiliselt võimalik hakata arendama reaalseid autonoomseid süsteeme, mis töötaksid juba satellidi pardal.

Joonis 1 kujutab arendatava prototüübi arhitektuuri, kus

- tumeda oranži värviga on kujutatud AI moodul;
- heleda oranži värviga on kujutatud kasutajaliidese moodul;
- sinise värviga on kujutatud statistilise analüüsi moodul:
- nooltega on ära näidatud, kuidas erinevad moodulid omavahel suhtlevad.



Joonis 1. Arendatava prototüübi arhitektuuri diagramm.

Antud lõputöö koostamisel keskenduti AI mooduli arendamisele ja integreerimisele ülejäänud süsteemi osadega. Statistilise analüüsi moodul ja kasutajaliides arendati paralleelselt teise meeskonna poolt.

3.1 Arendatava süsteemi ülevaade

Nagu eelnevalt mainitud, peab süsteem tervikuna olema suuteline autonoomselt töötlema ja analüüsima huvipakkuvate objektide liikumist ning seda arusaadaval viisil kasutajale näitama. Süsteem koosneb kolmest peamisest osast, milleks on AI moodul, kasutajaliides ja statistilise analüüsi moodul, mis on omavahel ühenduses ning andmed liiguvad järjest ühest moodulist teisse.

3.1.1 AI moodul

AI mooduli peamiseks eesmärgiks on töödelda ja ettevalmistada vajalikud andmed, enne kui need saadetakse statistilisse analüüsi. See on sisenemispunkt, kust andmed hakkavad järjest liikuma kuni need kuvatakse kasutajale.

Andmed saadetakse AI moodulisse läbi API. Andmete hulka kuuluvad tavaline RGB (*Red Green Blue*) pilt ning JSON (*JavaScript Object Notation*) kujul metaandmed, milles sisaldub pildi tegemise aeg ning nurkade GPS koordinaadid.

Kõige olulisem osa on eelnevalt treenitud ja testitud närvivõrk, mis teostab süsteemi sattunud kaugseire pildiandmete pealt tuvastusi. Tuvastatud objektide liikumise analüüsiks peab olema võimalik määrata objektide täpsed koordinaadid, mida samuti teostatakse AI moodulis.

AI moodulit käsitletakse täpsemalt lõputöö järgmistes peatükkides.

3.1.2 Statistilise analüüsi moodul

Pärast seda, kui AI moodul saab tuvastustelt vajalikud andmed kätte, saadetakse need edasi statistilisse analüüsi. Statistilise analüüsi mooduli peamiseks eesmärgiks on erinevate statistiliste algoritmide rakendamisel analüüsida objektide liikumisi ning tuvastada potentsiaalseid anomaaliaid. Anomaaliaks võib pidada näiteks konteinerlaeva liikumist sellise marsruudi järgi, mis ei ole tavaliste laevade puhul normaalne ning potentsiaalselt võib tähendada, et laevaga juhtus midagi või on tegemist millegi illegaalsega. Sellise informatsiooni väärtus on väga suur ning kui süsteem oskab piisavalt

hästi seda talletada ja kasutajatele presenteerida, siis on võimalik selle põhjal teha mitmeid kasulikke otsuseid või järeldusi.

3.1.3 Kasutajaliides

Kui andmed on edukalt analüüsitud, siis tuleb saadud tulemused visualiseerida ning kasutajale mugavas vormis kuvada. Selle jaoks on olemas kasutajaliides, kuhu prototüübi raames ühendati külge interaktiivne kaardivaade, kust on võimalik vaadata objektide liikumist (näiteks laevade analüüsimisel nende liikumise trajektoorid, kõige tihedamini külastatud punktid jne).

Lisaks on võimalik kasutajaliideses vaadata mingil kindlal ajahetkel tehtud tuvastusi.

4 Projekti disain

Selles peatükis loendatakse kõik tehnoloogilised valikud antud lõputöö tegemisel.

Kogu arendus käis Pythoni programmeerimiskeeles. Selle valik oli enamjaolt põhjendatud sellega, et teegid, mida kasutatakse *State of The Art* masinnägemises on kirjutatud Pythonis. Python on märgatavalt kiirem prototüüpimise keel liidestamaks mitmeid alamsüsteeme. Lisaks sellele on suurem osa arenduse käigus kasutatud tehnoloogiatest kõige paremini realiseeritud just Pythonis.

4.1 API

Süsteem peab olema alati valmis kaugseire pilte koos metaandmetega vastu võtma, selleks oli tarvis arendada API, mille arenduseks otsustati kasutada Flaski. See on avatud lähtekoodiga lihtne veebirakenduste arendamiseks mõeldud teek, mis põhineb WSGI (*Web Server Gateway Interface*) spetsifikatsioonil. Selle suurimaks eeliseks on laiendatavus ning kasutamise lihtsus. [8]

Arenduseks kaaluti ka kasutada Javascriptil põhinevat Node.js-i, mille eeliseks on suurem arendajate kogukond ja suurem ökosüsteem. Kuna APIga oli vajalik integreerida objektituvastuse mudel ning Python on masinõppes kõige laialdasemalt kasutatud keel, siis otsutati siiski Flaski kasuks.

API dokumenteerimiseks kasutati Flaski laiendit Flask-RESTX, mis pakub vahendeid API kirjeldamiseks ning Swagger API dokumentatsiooni genereerimiseks [9].

4.2 Närvivõrgu arendamiseks kasutatud tarkvara ja tehnoloogiad

Närvivõrgu treenimiseks otsustati kasutada TensorFlow platvormi ning selle peale ehitatud raamistikku nimega Object Detection API, mis on mõeldud objektituvastuse närvivõrkude loomiseks, treenimiseks ning juurutamiseks [10]. Sellist valikut põhjendas eelkõige platvormi laiendatavus ning väga põhjalik ja läbimõeldud dokumentatsioon. TensorFlow on tänapäeval üks populaarsemaid platvorme *State of The Art* masinnägemise mudelite ehitamiseks, treenimiseks, testimiseks ja juurutamiseks.

Object Detection API pakub suurtel andmestikel eeltreenitud mudeleid koos näidiskonfiguratsioonidega, mida on võimalik oma andmestiku peal edasi treenida. Selle eeliseks on olemasoleva närvivõrgu arhitektuuri ära kasutamine, kuna uue arhitektuuri loomine on väga pikk ja keeruline protsess. Lisaks on Object Detection APIs palju skripte, mis lihtsustavad mudelite treenimise, valideerimise ja testimise protsesside käivitamist.

Treenimise tulemuste jälgimiseks kasutati TensorFlow teegiga kaasa tulevat TensorBoard tööriista, mis on spetsiaalne masinõppe visualiseerimise tööriist [11]. Object Detection API genereerib automaatselt vajalikud failid, mida TensorBoard on võimeline lugema ning vastavalt visualiseerima.

Treeningpiltide augmenteerimiseks kasutati veebilehte Roboflow. Roboflow on keskkond, mis võimaldab kiirelt ning vähese vaevaga läbida terve objektituvastuse või klassifitseerimise mudeli treenimise protsessi, alates piltide markeerimisest, lõpetades mudeli juurutamisega oma rakenduses kasutamiseks. Samuti võimaldab Roboflow rakendada treeningandmete augmenteerimist, mida töös kasutati. [12] Protsessist tuleb lähemalt juttu alampeatükis 5.3.

4.3 Andmebaas

Andmebaasi valikuks osutus PostgreSQL. See on üsna populaarne relatsiooniline andmebaas, mida kasutatakse paljudes süsteemides. Selle peamisteks eelisteks on kiirus ja laiendatavus [13], mis oli arendatava rakenduse juures väga oluline aspekt, kuna tegu on reaalajas toimiva süsteemiga.

Lisaks toetab PostgreSQL väga palju andmetüüpe ning pakub ka võimalust paigutada andmebaasile erinevaid laiendusi nagu näiteks PostGIS[1], mis antud rakenduse korral oleks väga kasulik. Antud prototüübi arendamisel ei kasutatud laiendusi, kuid võimalus need hiljem rakendusele paigutada on suureks plussiks.

Andmebaasi APIga ühendamiseks kasutati Flaski laiendit Flask-SQLAlchemy, mis on vahend objekt-relatsioonvastenduseks ehk ORMiks (*Object-Relational Mapping*).

[1] PostGIS on laiendus tavalisele PostgreSQL andmebaasimootorile, mis võimaldab opereerida ruumiliste objektidega nagu näiteks punkt, joon, polügon jms [14].

Andmebaasi migratsioonide haldamiseks ehk andmebaasi uuendamiseks pärast struktuuri muudatusi kasutati Flaski laiendit Flask-Migrate, mis põhineb Alembicul, mis on tööriist andmebaasi migratsioonide haldamiseks ning mis on mõeldud töötama koos SQLAlchemyga.

4.4 Docker

Rakenduse kiireks jooksutamiseks kasutati Dockerit. Docker võimaldab lokaalselt jooksutada raskeid rakendusi ilma lokaalsesse masinasse lisatarkvara paigaldamata. Lisaks on heaks boonuseks võimalus rakendustest teha pildid[1] ning need üles panna Docker Hubi[2], kust inimesed võivad need endale alla tõmmata ja samamoodi lokaalselt jooksma panna. See aitas meil kiirelt levitada muudatused kliendile ja kliendi esindajale.

4.5 Projektihaldus

Projekti tegemine käis üldiselt ühenädalaste iteratsioonide kaupa.

Projektihaldus toimus peamiselt GitLab keskkonnas. Iga ülesande kohta tehti eraldi GitLabi *Issue*, kuhu pandi ülesande lahendamiseks vajaliku konteksti ning võimalusel ka ideid potentsiaalsete lahenduste kohta. Enamuste ülesannete puhul prooviti ära märkida ka eeldatav aeg, mida peaks kulutama ülesande lahendamisele. Kui ülesanne sai lahendatud, siis tehti eraldi *Merge Request* ning seejärel küsiti koodi ülevaatamist, et rakenduse stabiilsesse harusse jõuaks töötav kood.

Lisaks tehti ka GitLabi *Milestone*'id, kuhu pandi kokku kõik ülesanded, mida sooviti teatud iteratsiooni ajal ära teha. Iga *Milestone* käis üldiselt ühe iteratsiooni kohta.

Samuti kasutati ka ajajälgimise tarkvara nimega Wakatime. See on väike programm, mille abil on võimalik jälgida, kui palju mingil päeval koodi kirjutati. Selle tööle saamiseks piisab laiendi paigutamisest oma lemmikkoodiredaktorisse. [16]

---

[1] Pilt ehk *image* on lihtsalt üks fail, mida on võimalik Docker konteineris jooksutada. Selle sees on tavaliselt instruktsioonid Docker konteineri ehitamiseks [15].

[2] Docker Hub on portaal, kuhu on võimalik luua repositooriumid ning sinna üles laadida konteinerite pildid, mida teised inimesed saavad kasutada.

5 Objektide tuvastamine

5.1 Andmestik

Treenimiseks vajalikud satelliidipildid olid juba olemas ning need saadi Bing Maps[1] veebilehelt, kasutades selleks skripti, mis geograafiliste koordinaatide järgi satelliidipildi tagastab [17]. Kuna tarvis oli väga palju kaubalaevade pilte, siis piltide leidmise lihtsustamiseks kasutati veebilehte Marine Traffic[2] kust on võimalik leida sadamad, kus on kõige rohkem laevu.

Kõige ajakulukam protsess oli piltide markeerimine. Selleks kasutati labelImg tarkvara[3]. Töö tegemise ajal leiti ka veebileht Roboflow, kus on väga lihtne otse brauseris eraldi tarkvara alla laadimata pilte markeerida. Roboflow eelis on ka see, et see toetab 26-t erinevat markeeringu formaati, labelImg aga ainult kolme. Roboflow miinuseks on aga see, et tasuta versioon võimaldab hoiustada kuni 1000 pilti. Lõpuks koosnes treenimise andmestik 3275 positiivsest (kaubalaeva või -laevadega) ning 1400 negatiivsest (kaubalaevata) pildist.

5.2 Eeltöö

Eelnevalt tehtud katseid erinevate mudelite treenimisel nii kaubalaevade piltide peal kui ka lennukite ja lennujaamade piltide peal. Esimese katsetuse tulemusi on võimalik näha selle kohta kirjutatud bakalaureusetöös [18].

Teise katsetuse ajal kasutati treenimiseks kahte eeltreenitud mudelit TensorFlow 2 Model Zoo-st[4]: MobileNet V2 FPNLite 320x320 ja SSD MobileNet v2 320x320. Esimese mudeliga tekkis probleem *overfitting*-uga. See tähendab, et närvivõrk näitab treeningandmetel suurepäraseid tulemusi, aga testimisandmetel on tulemused kehvemad. Probleemi lahendamiseks prooviti lisada mudelile *dropout* parameeter, mis tähendab, et treenimise ajal eemaldatakse juhuslikult närvivõrgust peidetud või nähtavaid sõlmi, mille

[1] https://www.bing.com/maps

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

[3] https://github.com/tzutalin/labelImg

[4] https://bit.ly/2RJORZ6

tulemusena peaks närvivõrk testandmetel paremaid tulemusi näitama [19]. Kahjuks see mudeli tulemust ei parandanud ning otsustati jätkata treenimist teise mudeliga. Treening koosnes 100 000-st sammust.

Pärast treenimise lõppu konverteeriti treenitud mudel TensorFlow Lite (TFLite) formaati, kuna siis oli tarvis mudelit jooksutada TPU-l. TFLite formaat on ka praeguses projektis eeliseks, kuna mudeli jooksutamiseks ja objektide tuvastamiseks piisab TensorFlow Lite *runtime* paki kasutamisest, selle asemel, et terve TensorFlow teek alla laadida. Jooksutades mudelit Docker konteineris ei võta see nii palju ruumi ja ressurssi, sest on spetsiaalselt optimeeritud keskkondade jaoks, kus ressursid on väga limiteeritud. Tänu sellele vähenes näiteks rakenduse Docker pildi suurus ligi kaks gigabaiti.

Mudelit testiti 600 pildiga, millest 420 olid positiivsed ja 180 negatiivsed. Testi tulemuste kohta koostati eksimismaatriks (*confusion matrix*) (Tabel 1), mis näitab, kui täpne mudel on. Tabeli esimeses reas on näha, kuidas tuvastas mudel positiivseid testpilte ning teises reas, kuidas negatiivseid pilte [20].

Tabel 1. Treenitud mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	250	169
	Negatiivne	112	67

Tulemuste põhjal võib öelda, et treeningpiltide hulgas oleks võinud rohkem variatsiooni olla, kuna tulemuste seas oli palju valepositiivseid tulemusi.

5.3 Andmete augmenteerimine

Kuna treenitud mudel ei näidanud testandmetel kõige paremaid tulemusi, siis sooviti mudelit parandada. Kahjuks ei olnud võimalik algset mudelit edasi treenida, kuna olid

kadunud selle treenimisel tekkinud *checkpoint*[1] failid, mis võimaldavad treenimist jätkata pooleli jäänud kohast. Seega otsustati SSD MobileNet v2 320x320 mudel algusest uuesti treenida.

Selleks, et uuesti treenimisest kasu oleks, oli vaja treeningpiltidesse rohkem variatsiooni. Selleks, et aega säästa, otsustati uute treeningpiltide tegemise asemel rakendada olemasolevate treeningandmete augmenteerimist. Andmete augmenteerimiseks nimetatakse protsessi, kus ühest treeningpildist tehakse mitu versiooni, millest igaühte muudetakse mingil viisil, näiteks peegeldatakse, keeratakse, hägustatakse, lisatakse müra. Joonis 2 on üks näide pildi augmenteerimisest, kus esialgset pilti on pööratud väikese nurga all ning rakendatud vertikaalset ja horisontaalset peegeldamist. Inimese silma jaoks ei pruugi need pildid olla väga erinevad, kuid närvivõrgu jaoks on need kaks täiesti erinevat pilti erinevate omadustega.

Joonis 2. Esialgne pilt (paremal) ja selle augmenteeritud versioon (vasakul)

Nii saadakse algsest treeningandmete kogusest mitu korda suurem treeningandmete hulk, ilma, et oleks tarvis uusi andmeid leida [21]. Andmete augmenteerimiseks kasutati veebilehte Roboflow, kus on võimalik piltidele rakendada erinevaid augmentatsioone. Iga algse treeningpildi kohta tekkis kolm pilti, millele rakendati suvaliselt üks järgmistest augmentatsioonidest: horisontaalne peegeldus, vertikaalne peegeldus, kärbe (0% kuni

[1] *Checkpoint* fail sisaldab mudeli parameetrite väärtuseid kindlal treeningsammul, millest on võimalik kogu mudel taastada

30% suurendus), hägustamine kuni üks piksel, pööre -15° kuni 15°. Nii saadi kokku 19 910 pilti, millest 10 386 positiivset ja 4524 negatiivset (Tabel 2).

Tabel 2. Augmenteeritud piltide jagunemine.

Andmehulga tüüp	**Positiivsed pildid**	**Negatiivsed pildid**
Treenimine	7271	0
Valideerimine	2078	905
Testimine	1038	452

5.4 Närvivõrgu treenimist mõjutavate hüperparameetrite ülevaade

Sügavnärvivõrk koosneb mitmetest parameetritest, mis määravad ära närvivõrgu struktuuri ja ka selle, kuidas mudel treenib. Neid parameetreid nimetatakse hüperparameetiteks. Need on olulised näitajad, mis otseselt mõjutavad närvivõrgu treenimist, seega tuleb treenimise protsessis mängida erinevate väärtustega, et leida parimad, mis aitaksid viia närvivõrgu täpsuse soovitud tasemele. [22]

Antud lõputöös keskenduti järgmiste hüperparameetrite mõju uurimisele mudeli täpsusele:

- optimeerija (ingl. *optimizer*);
- õppimiskiirus ehk LR (ingl. *learning_rate*);
- *batch_size* ehk piltide arv, mida närvivõrgust läbi lastakse enne kaalude uuendamist;
- treenimise sammude arv.

Lisaks katsetati ka mudeli sisemiste parameetrite muutmisega, kuid suure võimalike väärtuste hulga tõttu otsustati keskendud eelnimetatud hüperparameetrite uurimisele.

5.4.1 Optimeerija

Optimeerijaks nimetatakse sellist funktsiooni, mis uuendab mudeli sisemiseid parameetreid (kaalusid) vastavalt kao funktsioonile. Kao funktsioon näitab ära, kui hea või halb oli mudeli ennustus kindlal sammul ning ideaalis peaks see tagastama 0, mis tähendab, et mudeli ennustus oli 100% õige. Optimeerija aga võtab selle väärtuse arvesse

ning uuendab mudeli parameetrid vastavalt nii, et üldine mudeli täpsus liiguks suurema väärtuse poole. [23]

5.4.1.1 Momentum

Momentum optimeerimisalgoritm põhineb sellel mõttel, et kui lasta pall veerema mäest alla, kogudes hoogu omab see suuremat tõenäosust jõuda madalama nõlvani. Selle optimeerimisalgoritmi korral lisatakse gradientidele väärtuse, mis on reguleeritud ajaga. See aitab kiiremini jõuda optimaalse lahenduseni. [24]

5.4.1.2 RMSProp

Veel üks populaarne optimeerimisalgoritm on RMSprop. See algoritm oli loodud eelkõige selleks, et lahendada probleemi, kus närvivõrgu treenimisel gradiendid kasvavad liiga suureks või vastupidi jäävad nii väikesteks, et praktiliselt kaovad ära. RMSProp kasutab nimetatud probleemi lahendamiseks ruutgradientide libisevat keskmist, et normaliseerida gradiendid ning sellega hoida õppimise samm stabiilsena. Lihtsamalt öeldes õppimiskiirust ei hoita kogu aeg ühe väärtusena, vaid muudetakse vastavalt treenimise progresseerumisele. [25]

5.4.1.3 Adam

Adam optimeerimisalgoritm on tänapäeval üks kõige populaarsemaid, tänu selle efektiivsusele. See loodi lahendamaks kõige suuremad probleemid, millega ülejäänud optimeerimisalgoritmid nagu RMSprop [26] ja AdaGrad [27] kokku puutusid. [28]

5.4.2 LR

Õppimiskiirus on selline hüperparameeter, mis võimaldab reguleerida, kui suure sammu teeb optimeerija närvivõrgu kaalude muutmisel. Kui igal sammul muuta närvivõrgu kaalud liiga suure väärtuse võrra, siis võib tekkida olukord, kus närvivõrk hüppab üle globaalse kao funktsiooni miinimumi. Kui aga muuta kaalud liiga väikese väärtuse võrra, siis tekib vastupidine olukord, kus närvivõrk ei pruugi kunagi jõuda globaalse miinimumini või jõuab selleni väga aeglaselt. Õppimiskiirus aitab selliseid olukordi vältida. See on tavaliselt mingi väike arv (reeglina 0.01 kuni 0.001), millega korrutatakse närvivõrgu kaalude gradiendid, et kaalude muutused ei oleks liiga suured. [23]

Siinkohal tasub ära mainida, mis on gradient. Matemaatiliselt tähendab see osatuletist, mis näitab ära mingi suuruse muudu. Gradient ühendab kao funktsiooni koos kaaludega,

näidates, mida tuleks teha kaaluga (näiteks korrutada arvuga, vähendada mingi arvu võrra jne), et kao funktsiooni väärtust vähendada. [23]

Antud lõputöös katsetati *cosine_decay*[1] ning *exponential_decay*[2] õppimiskiiruse parameetritega.

5.4.3 *Batch_size*

Batch_size on hüperparameeter, mis mõjutab närvivõrgu treenimise kiiruse ning stabiilsuse. Sellega määratakse ära, kui palju pilte läheb närvivõrgust läbi ning mille pealt hinnatakse kadu, enne kui uuendatakse närvivõrgu kaalude väärtused. Näiteks *batch_size* 4 tähendab seda, et kasutatakse 4 pilti treeningandmestikust närvivõrgu kao selgitamiseks, enne kui kaalud saavad uuendatud. [29]

Batch_size võib osutuda ka pudelikaelaks kui närvivõrku treenitakse GPU (*Graphics Processing Unit*) peal, kuna andmed hoiustatakse GPU mälus. Kui pildiandmed on piisavalt suure resolutsiooniga, siis ei pruugi kogu piltide partii mällu ära mahtuda, seega tüüpiliselt valitakse *batch size* väärtuseks arvud 1st kuni paari sajani, sõltuvalt andmestikust ning GPU parameetritest. [29]

5.4.4 Treenimise sammude arv

Treenimise sammuks nimetatakse olukorda, kus kogu treenimisandmestik, mis jagatakse gruppideks, mille suurust määrab ära *batch_size*, saab närvivõrgust ühe korra läbi lastud. Tavaliselt valitakse sammude arvuks mingi suur arv, reeglina mõni tuhat. Kui valida liiga väike sammude arv, siis ei pruugi mudel selle ajaga jõuda optimaalse miinimumini, seega suuremad arvud on eelistatud valik. [30]

---

[1] *cosine_decay* LR kasvatab õppimiskiirust X sammu jooksul maksimaalse valitud väärtuseni ning pärast hakkab seda alandama nii, et treenimise lõpuks on õppimiskiirus võrdne nulliga.

[2] *exponential_decay* LR hoiab õppimiskiirust kindla väärtuse juures X sammu jooksul ning pärast hakkab seda kindla väärtuse võrra alandama. Selle graafik kujutab ette treppi.

5.5 Uue närvivõrgu treenimine

Object Detection API abil treenides on vaja mudelitele ette anda andmestik spetsiaalses TFRecord formaadis, mis on mõeldud binaarsete järjestikuste andmete hoidmiseks [31]. Selleks, et konverteerida pildid TFRecord formaati, kasutati Object Detection API dokumentatsioonis näitena toodud skripti[1], millele omalt poolt lisati funktsionaalsust mitmete TFRecord failide loomiseks ning andmestiku jagamiseks treening-, valideerimis- ja testimisandmestikeks. Skripti käivitamisel (python tf_record_generator.py
    --input_path='path_to_input_files'
    --output_path='output_tfrecord_path'
Joonis 3) antakse sellele ette teekond piltidele ning teekond, kuhu genereeritud failid salvestada. Selle järel genereeritakse kolm kausta, milles sisalduvad vastavalt treenimise, valideerimise ja testimise TFRecord failid.

```
python tf_record_generator.py
    --input_path='path_to_input_files'
    --output_path='output_tfrecord_path'
```

Joonis 3. TFRecord faile genereeriva skripti käivitamine käsurealt.

Vaikimisi jagatakse andmestik nii, et 70% positiivsetest piltidest lähevad treenimiseks, 20% valideerimiseks ja 10% testimiseks. Negatiivseid pilte kasutati ainult valideerimisel ja testimisel ning osakaal oli vastavalt 20% ja 10%. Kõiki negatiivseid pilte otsustati mitte kasutada, sest vastasel juhul tekkis olukord, kus valideerimine ja testimine toimus pigem negatiivsete piltide peal, mis ei andnud vastust sellele, kui hästi suutis mudel tuvastada objekte, mille peal seda treeniti.

Treenimist käivitati Object Detection APIs sisalduva skripti[2] abiga, mida käivitati samuti käsurealt (Joonis 4).

```
python object_detection/model_main_tf2.py
    --model_dir='output_model_path'
    --pipeline_config_path='path_to_model_config'
```

Joonis 4. Treenimise jooksutamine käsurealt.

Treenimise jooksvat aruannet jälgiti TensorBoard visualiseerimise tööriista abil genereeritud graafikute pealt (Lisa 2).

[1] https://bit.ly/3umvK4q

[2] https://bit.ly/3fMS4in

Tabel 3 on välja toodud arvuti parameetrid, mille peal treenimist läbi viidi.

Tabel 3. Arvuti parameetrid, mille peal toimus treenimine ja testimine.

GPU	NVIDIA GeForce GTX 1650 Super, 4 GB
CPU (*Central Processing Unit*)	AMD Ryzen 5 1600AF, 3.6 GHz
RAM (*Random Access Memory*)	16 GB

5.5.1 Treenimine algsete parameetritega

Esimese katsena prooviti mudelit treenida algse konfiguratsiooniga (Joonis 5), mis tuli mudeliga kaasa, ning hinnata selle tulemusi.

```
batch_size: 512
num_steps: 50000
optimizer {
  momentum_optimizer {
    learning_rate {
      cosine_decay_learning_rate {
        learning_rate_base: 0.800000011920929
        total_steps: 50000
        warmup_learning_rate: 0.13333000242710114
        warmup_steps: 2000
      }
    }
    momentum_optimizer_value: 0.8999999761581421
  }
  use_moving_average: false
}
```

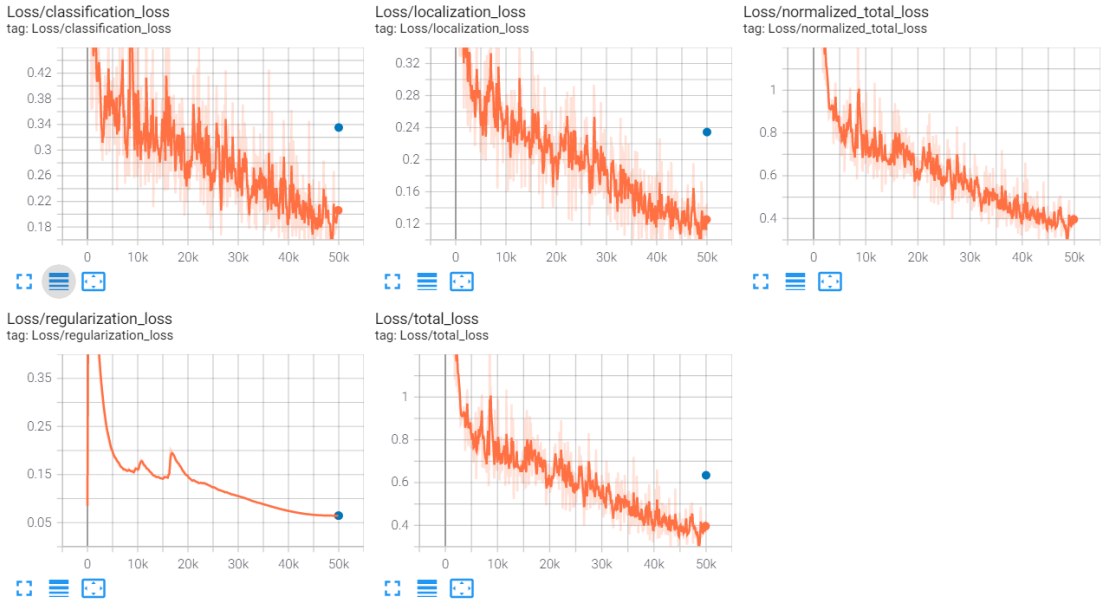
Joonis 5. Algne mudeli konfiguratsioon.

Antud konfiguratsiooniga treenimine kahjuks suure *batch_size* arvu tõttu ei õnnestunud. Nii palju pilte lihtsalt ei mahtunud graafikakaardi mälusse, seega otsustati katsetada erinevate *batch_size* väärtustega, et leida väärtus, mille juures oleks võimalik treenimist jätkata. Esimesel katsel osutus valituks väärtus 16.

Joonis 6 kujutab graafikuid kao funktsiooni muutusest treenimise ajal. Y telg näitab kao funktsiooni väärtust ning X on kindel treenimise samm. Nagu näha, läks kao funktsiooni väärtus minimaalselt veidi alla 0.4 ning graafikutelt on näha, et suuremate sammude arvuga oleks see langenud veel rohkemgi.

Graafikutel on näha ka sinist täppi, mis on valideerimise ajal saadud tulemused. Treenimisaegsest valideerimisest on võimalik täpsemalt lugeda peatükis 8.1. Kahjuks

sellel katsel unustati paralleelselt treenimisega panna käima ka valideerimise skript, seega on graafikul kujutatud ainult viimase mudeli *checkpoint* faili valideerimise tulemus.



Joonis 6. Treenimise tulemused algse konfiguratsiooniga.

5.5.2 Treenimine algse optimeerija ja *exponential_decay* LR parameetriga

Järgenvalt otsustati katsetada algses konfiguratsioonis *learning_rate* parameetri muutmist. Object Detection API koodist leiti[1], et võimalikest väärtustest on võimalik lisaks *cosine_decay* õppimiskiirusele valida ka *exponential_decay* õppimiskiirus, mida ka tehti (Joonis 7). Väärtused valiti järgnevalt:

- õppimiskiiruse headeks väärtusteks peetakse 0.01-0.001 [32], seega otsustati valida *initial_learning_rate* väärtuseks vahepealne väärtus ehk 0.005;
- *decay_steps* on parameeter, mis ütleb ära mitme sammu järel muutub õppimiskiiruse väärtus. See valiti suvaliselt;
- *decay_factor* näitab, kui palju muutub õppimiskiirus. See vaikimisi väärtus leiti Object Detection API koodist ning jäeti muutmata;
- *burnin_steps* näitab, kui mitu sammu tehakse enne, kui õppimiskiirust esimest korda muudetakse. See tähendab, et enne treenitakse mudelit esialgse

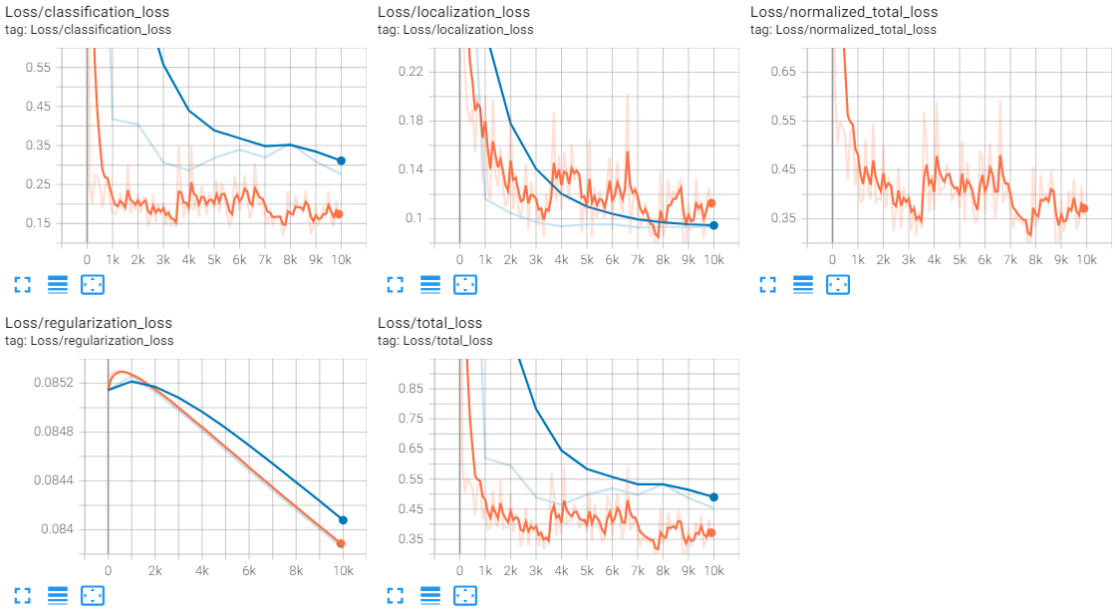
[1] https://bit.ly/3bTA30W

õppimiskiirusega ja kui valitud sammude arv on läbitud, hakatakse õppimiskiirust muutma. See väärtus valiti suvaliselt.

Seekord oli kadu natuke väiksem, kuid ikkagi kaugel ideaalist (Joonis 8). Samuti õnnestus seekord käivitada ka valideerimine, mille tulemusi kujutab graafikul sinine joon.

```
momentum_optimizer {
  learning_rate {
    exponential_decay_learning_rate {
      initial_learning_rate: 0.005
      decay_steps: 750
      decay_factor: 0.969999988079
      burnin_steps: 2000
    }
  }
  momentum_optimizer_value: 0.8999999761581421
}
```

Joonis 7. Algne konfiguratsioon koos muudetud LR parameetriga.



Joonis 8. Treenimise tulemused algse optimeerija ja *exponential_decay* LR parameetriga.

5.5.3 Treenimine Adam optimeerija ja *cosine_decay* LR parameetriga

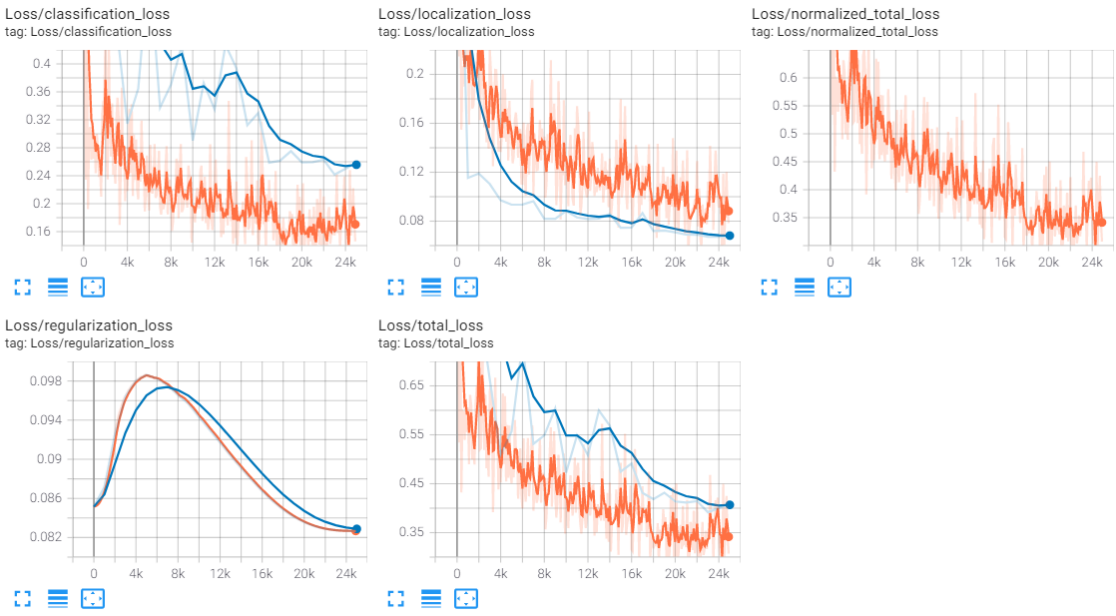
Antud katse ajal kasutati sama õppimiskiiruse konfiguratsiooni, kuid Momentum optimeerija asemel prooviti Adam optimeerijat (Joonis 9). Samuti muudeti ka treenimise sammude arv 25 000 peale. Antud optimeerija konfiguratsioonis olev parameeter *epsilon*

on lihtsalt üks konstant, mis valiti TensorFlow ametliku dokumentatsiooni soovituste järgi[1].

Sellel katsel oli kadu väiksem kui Momentum optimeerijaga, kuid on näha, et kao funktsiooni graafik hüpleb palju rohkem (Joonis 10). Hüplemine viitab suurele sammule optimeerimisel, mis tähendab seda, et närvivõrgu kaalud uuendatakse harvemini.

```
adam_optimizer {
  learning_rate {
    cosine_decay_learning_rate {
      learning_rate_base: 0.800000011920929
      total_steps: 25000
      warmup_learning_rate: 0.13333000242710114
      warmup_steps: 2000
    }
  }
  epsilon: 1.0
}
```

Joonis 9. Konfiguratsioon Adam optimeerija ja algse LR parameetritega.



Joonis 10. Treenimise tulemused Adam optimeerija ja *cosine_decay* LR parameetriga.

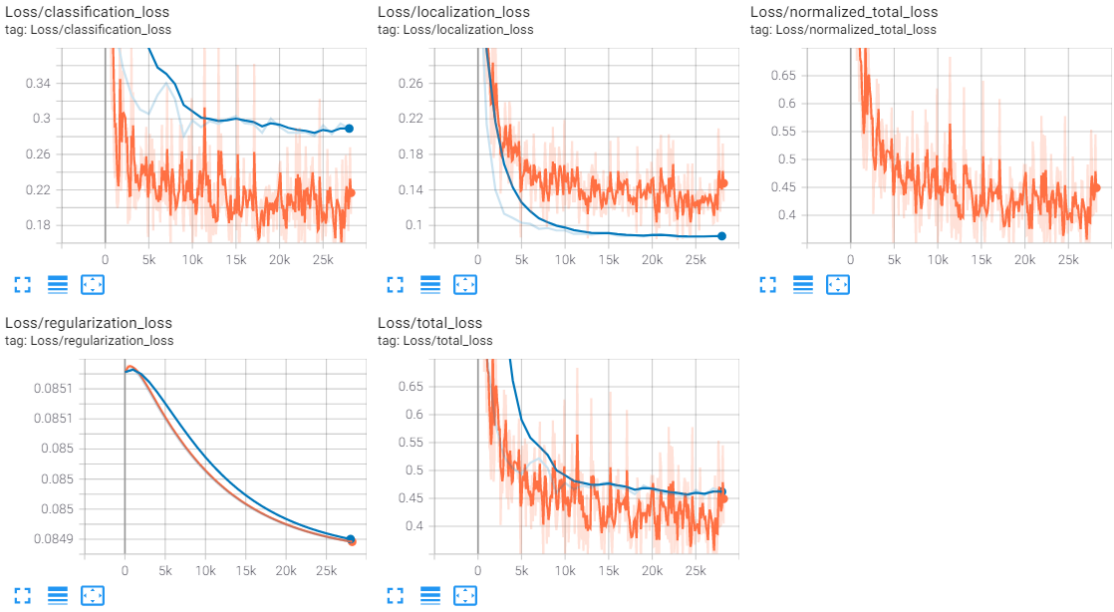
[1] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam#notes

5.5.4 Treenimine Adam optimeerija ja *exponential_decay* LR parameetriga

Kuna esialgse optimeerimisalgoritmi ja *exponential_decay* kombinatsioon andis parema tulemuse kui esialgne konfiguratsioon, siis otsustati proovida ka Adam optimeerimisalgoritmi kasutada koos antud õppimiskiiruse parameetriga (Joonis 11). Seekord otsustati aga katsetada veidi teistsuguste väärtustega ning panna algseks õppimiskiiruseks (*initial_learning_rate*) 0.004, õppimiskiiruse uuendamise sammuks (*decay_steps*) 500 ja õppimiskiiruse muut (*decay_factor*) ümardada 0.95-ni. Nagu näha graafikult (Joonis 12), siis põhjustasid sellised muutused suurema kao funktsiooni väärtuste hüplemise, küll aga valideerimise kadu läks väiksemaks ning selle juures hüplemine vähenes, mis on hea märk.

```
adam_optimizer {
  learning_rate {
    exponential_decay_learning_rate {
      initial_learning_rate: 0.004
      decay_steps: 500
      decay_factor: 0.95
      burnin_steps: 2000
    }
  }
  epsilon: 1.0
}
```

Joonis 11. Konfiguratsioon Adam optimeerija ja *exponential_decay* LR parameetriga.



Joonis 12. Treenimise tulemused Adam optimeerija ja *exponential_decay* LR parameetriga.

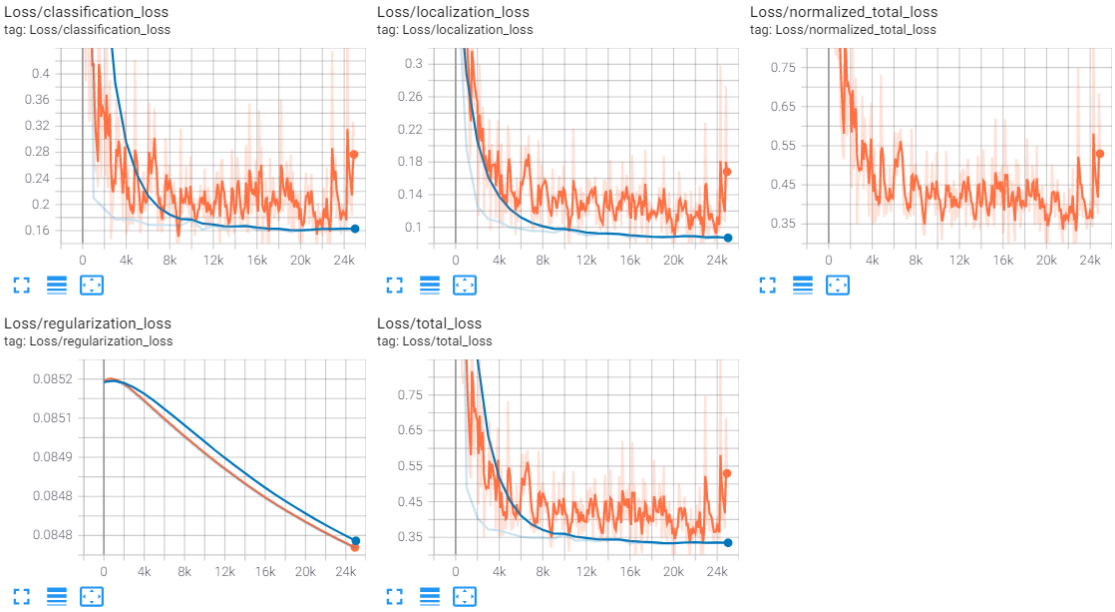
Hiljem tehti veel paar katset ning selgus, et *exponential_decay* õppimiskiirusega *decay_steps* parameetri vähendamisel hakkab kao funktsioon käituma ebastabiilselt.

Seega otsustati treenida ka esialgse optimeerijaga kasutatud *exponential_decay* õppimiskiiruse konfiguratsiooniga (Joonis 13). Antud katsel vähenes kao funktsiooni hüplemine võrreldes eelneva katsega ning kadu langes veidi (Joonis 14). Küll aga antud katse lõpus hakkas kadu veidi kasvama.

Valideerimise kadu stabiliseerus veelgi rohkem võrreldes eelnevate katsetega.

```
adam_optimizer {
  learning_rate {
    exponential_decay_learning_rate {
      initial_learning_rate: 0.005
      decay_steps: 750
      decay_factor: 0.96999997
      burnin_steps: 2000
    }
  }
  epsilon: 1.0
}
```

Joonis 13. Teise katse konfiguratsioon Adam optimeerija ja *exponential_decay* LR parameetriga.



Joonis 14. Teise treenimise katse tulemused Adam optimeerija ja *exponential_decay* LR parameetriga.

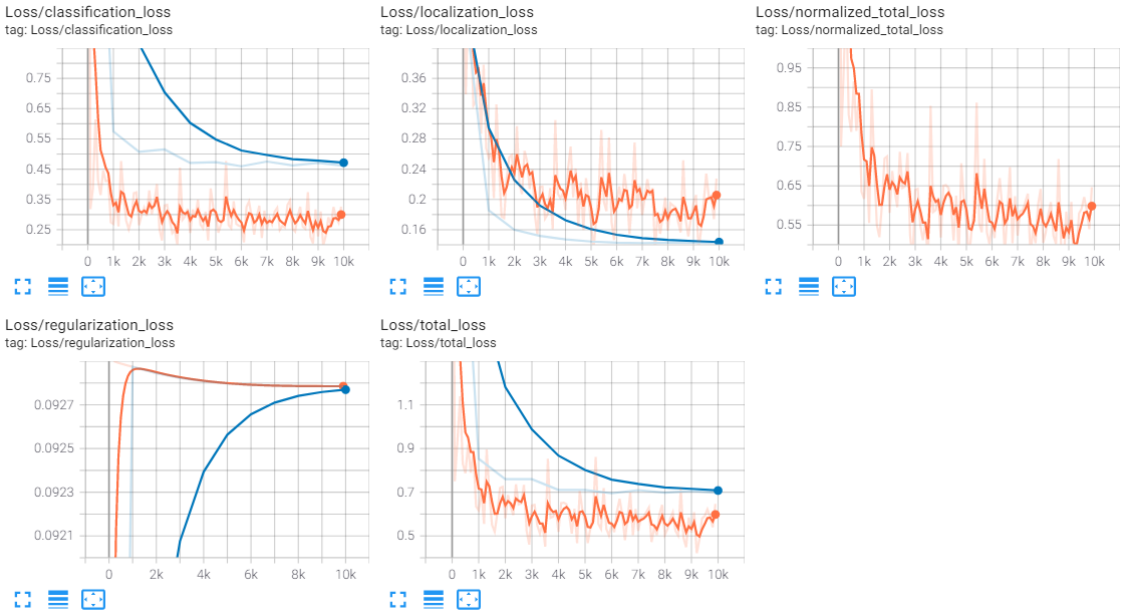
5.5.5 Treenimine RMSprop optimeerija ja *cosine_decay* LR parameetriga

Viimane optimeerimisalgoritm, mida prooviti rakendada, on RMSprop. Esialgu prooviti jällegi muuta ainult optimeerijat ning jätta alles esialgne LR konfiguratsioon (Joonis 15). Samuti prooviti muuta sammude arv 10 000 peale, et vaadata, kuidas see tulemusi

mõjuta.. Antud katse tulemused olid ülejäänutest kehvemad ning vaadates graafikuid võiks järeldada, et kohe algusest ei ole kadu eriti palju muutunud. (Joonis 16).

```
rms_prop_optimizer {
  learning_rate {
    cosine_decay_learning_rate {
      learning_rate_base: 0.800000011920929
      total_steps: 10000
      warmup_learning_rate: 0.13333000242710114
      warmup_steps: 2000
    }
  }
  momentum_optimizer_value: 0.899999976158
  decay: 0.899999976158
  epsilon: 1.0
}
```

Joonis 15. Konfiguratsioon RMSprop optimeerija ja algse LR parameetriga.



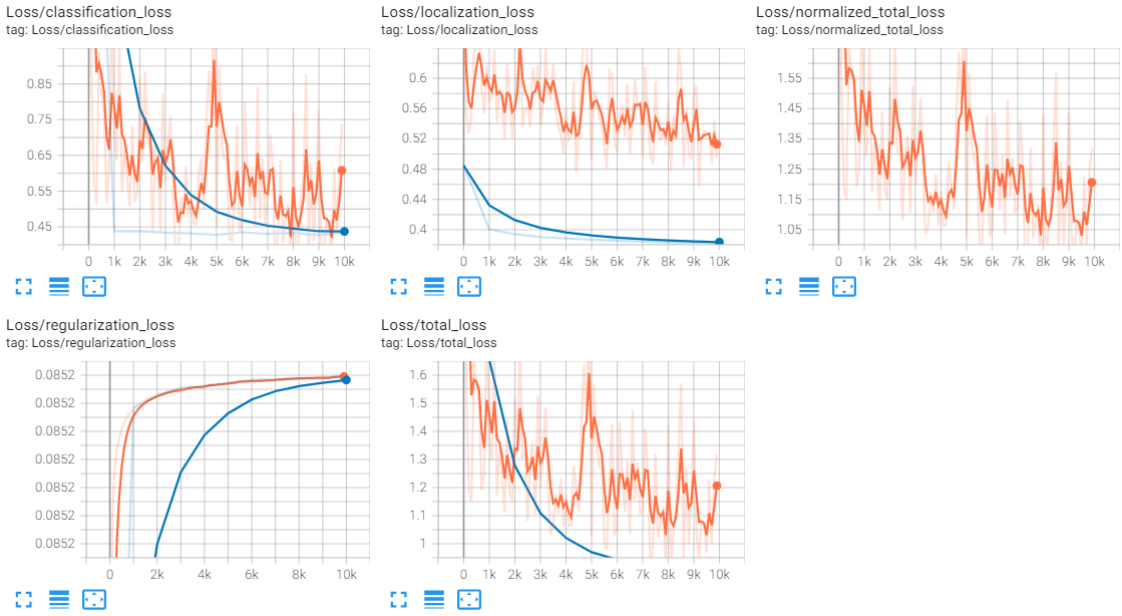
Joonis 16. Treenimise tulemused RMSprop optimeerija ja algse LR parameetriga.

5.5.6 Treenimine RMSprop optimeerija ja *exponential_decay* LR parameetriga

Viimase katsena prooviti rakendada RMSprop optimeerijat koos *exponential_decay* õppimiskiiruse parameetriga (Joonis 17). Antud katse tulemused olid kogu katsete hulgas kõige kehvemad nii treenimise kui ka valideerimise kao poolest (Joonis 18). Treenimise keskel toimus suurem hüpe ning üldiselt oli kadu treenimise ajal pigem kaootiline.

```
rms_prop_optimizer {
  learning_rate {
    exponential_decay_learning_rate {
      initial_learning_rate: 0.005
      decay_steps: 750
      decay_factor: 0.969999988079
      burnin_steps: 2000
    }
  }
  momentum_optimizer_value: 0.899999976158
  decay: 0.899999976158
  epsilon: 1.0
}
```

Joonis 17. Konfiguratsioon RMSprop optimeerija ja *exponential_decay* LR parameetriga.



Joonis 18. Treenimise tulemused RMSprop optimeerija ja *exponential_decay* LR parameetriga.

5.6 Uue närvivõrgu testimine

Pärast seda, kui katsetused erinevate hüperparameetritega olid tehtud, tuli aeg läbi viia testid. Selle jaoks oli eelnevalt vaja saadud mudelid eksportida õigesse formaati, et oleks võimalik nendega tuvastamist jooksutada.

5.6.1 Mudelite eksportimine testimiseks

Eksportimiseks kasutati Object Detection API skripti [1], mis paneb mudeli kokku selle viimasest *checkpoint* failist ning valmistab selle ette testimiseks. Samuti on võimalik eksporditud mudel konverteerida TFLite formaati, mida on hiljem võimalik jooksutada näiteks TPU peal. Kuna antud töö tegemisel ei olnud TPUd võimalik kasutuada ning see ei olnud ka vajalik, siis otsustati TFLite formaati konverteerimine vahele jätta. Eksportimiseks oli vajalik Object Detection API repositooriumist käivitada skript, millele anti ette treenitud mudeli *checkpoint* faili asukoht, mudeli konfiguratsiooni asukoht ning soovitud koht, kuhu genereerida uued mudeli failid (Joonis 19). Lisa 3 kujutab genereeritud faile ning kausta struktuuri.

```
python object_detection/export_tflite_graph_tf2.py \
    --pipeline_config_path path/to/ssd_model/pipeline.config \
    --trained_checkpoint_dir path/to/ssd_model/checkpoint \
    --output_directory path/to/exported_model_directory
```

Joonis 19. Mudeli eksportimiseks vajaliku käsu jooksutamine käskäsurealt.

5.6.2 Testimise protsess

Testimiseks valmistati ette skript, mille abil on võimalik vajalikku mudelit jooksutada koos ette antud piltidega ning koguda statistikat tuvastuste kohta. Samuti salvestab skript pildid, mille peal on kujutatud tuvastatud objektid, objektide klassid ning kui kindel on mudel selles, et antud klass tõesti kuulub sellele objektile (Lisa 4). Skripti põhjaks võeti Object Detection APIs leiduvad näidet[2].

Skriptile tuleb ette anda teekond eelnevalt genereeritud TFRecord failile koos testandmestikuga, teekond eelmisel sammul eksporditud mudelile ning teekond, kuhu salvestada kõik mudeli poolt tehtud tuvastused (Joonis 20).

```
python model_test.py
    --tf_record_path='path_to_tfrecord'
    --model_dir='path_to_exported_model'
    --detection_save_path='path_for_saved_images'
```

Joonis 20. Mudeli testimist käivitatava skripti jooksutamine käsurealt.

[1] https://github.com/tensorflow/models/blob/master/research/object_detection/export_tflite_graph_tf2.py

[2] https://bit.ly/3oPAazQ

5.6.3 Testimise tulemuste analüüs

Eelnevas peatükis mainitud skript genereerib lisaks kõigele muule ka statistikat mudeli tuvastuste kohta. Selle jaoks kasutatakse eksimismaatriksit. Siinkohal tuleb täpsustada, et sellel sammul saadud eksimismaatriks sisaldab tulemusi tuvastatud objektide kohta, mitte piltide kohta. Kuna piltidel võib olla rohkem kui üks objekt, siis ei ole eksimismaatriksi arvud alati võrdsed andmestikus kasutatud piltide arvuga. Eksimismaatriksite koostamise loogika kohta on võimalik lugeda peatükis 8.2.

Testimise käigus filtreeriti välja tuvastused, mille puhul mudelid olid vähem kui 60% kindlad, et tegu on õige objektiga.

Tabel 4 - Tabel 10 kujutavad erinevate mudelite testimisest saadud eksimismaatrikseid.

Tabel 4. Esialgse konfiguratsiooniga mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1167	170
	Negatiivne	117	343

Tabel 5. Esialgse optimeerija ja *exponential_decay* LR konfiguratsioonig mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1239	98
	Negatiivne	106	356

Tabel 6. Adam optimeerija ja esialgse LR konfiguratsiooniga mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1147	190
	Negatiivne	134	326

Tabel 7. Adam optimeerija ja *exponential_decay* LR konfiguratsiooniga mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1165	172
	Negatiivne	87	386

Tabel 8. Adam optimeerija ja *exponential_decay* LR konfiguratsiooniga teise mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1174	163
	Negatiivne	7	452

Tabel 9. RMSProp optimeerija ja esialgse LR konfiguratsiooniga mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1158	179
	Negatiivne	177	287

Tabel 10. RMSprop optimeerija ja *exponential_decay* LR konfiguratsiooniga mudeli eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	1218	119
	Negatiivne	1834	406

Eksimismaatriksite põhjal võib öelda, et Adam optimeerija ja *exponential_decay* LR parameetriga mudelid saavad kõige paremini hakkama õigete objektide tuvastamisega. Samuti on nende mudelite korral valepositiivsete ning valenegatiivsete piltide osakaal oluliselt väiksem kui teiste mudelite korral.

Momentum optimeerijaga mudelitest andis paremaid tulemusi samuti *exponential_decay* LR parameetriga mudel, kuna valepositiivsete ja valenegatiivsete piltide osakaal oli väiksem esialgse konfiguratsiooniga mudelist.

RMSprop optimeerijaga mudelitest osutus paremaks esialgse LR parameetri konfiguratsiooniga mudel, kuid see oli ikkagi veidi halvem kui eelmised. *Exponential_decay* LR parameetriga mudal osutus aga kõige halvemaks, kuna valepositiivsete tuvastuste arv oli märkimisväärselt suur. Hilisemal piltide üle vaatamisel selgus aga, et see oli osaliselt põhjustatud ka sellest, et mudel tegi küll mitu tuvastust, kuid need käisid ühe objekti kohta (Lisa 6), seega õige arv peaks olema ca kolmandik sellest, mida on näha eksimismaatriksis. Sellest võib järeldada, et kui treenida mudelit suurema sammude arvuga, siis oleks arvatavasti võimalik tulemusi parandada.

5.6.4 Võrdlus olemasoleva mudeliga

Selleks, et aru saada, kui head või halvad saadud tulemused võrreldes olemasoleva mudeliga on, otsustati jooksutada sama testimise protsessi ka olemasoleva mudeliga. Kuna aga olemasolev mudel oli juba TFLite formaadis, siis ei olnud võimalik varasemalt kirjutatud skriptiga seda käivitada, seega otsustati skripti muuta. Selleks lisati võimalus testimise skriptile ette anda TFLite formaadis salvestatud mudeli teekond ja lisaks klasside ja klassi ID-de faili teekond. Vajalikud muudatused võeti varasemate aastate jooksul tehtud repositooriumist[1]. Lõpuks saadi üks skript, millest saab jooksutada nii eksporditud mudeleid kui ka juba TFLite formaadis olevaid mudeleid (Joonis 21).

```
python model_test.py
    --tf_record_path='path_to_test_tfrecord'
    --tflite_model_path='path_to_tflite_model'
    --label_map_path='path_to_label_map'
    --detection_save_path='path_to_output_images'
```

Joonis 21. TFLite mudeli testimist käivitatava skripti jooksutamine.

Tabel 11 näitab olemasoleva mudeli testimise tulemusena saadud eksimismaatriksit.

Tabel 11. Olemasoleva mudeli testimisel saadud eksimismaatriks.

		Tuvastatud	
		Positiivne	Negatiivne
Tegelik	Positiivne	334	820
	Negatiivne	273	266

Tulemuste põhjal selgub, et olemasolev mudel ei suutnud paljude piltide korral õigeid objekte tuvastada, kuna valenegatiivsete tulemuste arv on üsna suur. Üks mõjuv faktor selle juures võib olla see, et mudeli jaoks oli tegu uute andmetega, kuna eelnevalt oli see treenitud ideaalsete andmete peal. Kuna aga testimise eesmärgiks ongi anda mudelile pildid, mida see pole varem näinud, siis võib järeldada, et olemasoleva mudeli täpsus ei ole võrreldes uute mudelitega nii hea.

[1] https://bit.ly/3fgFuZU

6 API pildiandmete vastu võtmiseks

API näol on tegu lihtsa REST APIga, mis koosneb kahest *endpoint*-ist[1]: /api/ai/image ning /api/ai/detected_object. API eesmärgiks on vastu võtta kaugseire pildiandmed ning suunata need närvivõrku tuvastamiseks, mille järel suunata närvivõrgu poolt tehtud tuvastused statistilisse analüüsi.

Prototüübi tarbeks oli tarvis süvanärvivõrgu mudelit, mis oleks treenitud objekte tuvastama. Kuna küllaltki edukalt oli treeninud mudel kaubalaevade tuvastamiseks satelliidipiltidelt, siis sobis see mudel ka siia rakendusse. Kuna API arendus toimus enne mudeli uuesti treenimist, siis kasutati APIs eelnevalt treenitud mudelit, kuid lihtsa vaevaga on võimalik mudel selle töö raames treenitud mudeli või mõne muu endale sobiliku mudeliga asendada.

6.1 *Endpoint* /api/ai/image

Endpoint-i /api/ai/image eesmärk on olenevalt päringumeetodist vastu võtta kaugseire pildiandmeid ning metadatat või tagastada info andmebaasi salvestatud piltide kohta. Piltide salvestamisest tuleb juttu alampeatükis 6.3. *Endpoint*-ist saadavaid andmeid on võimalik kasutada kasutajaliideses, kus saab näiteks kaardile kuvada tuvastatud objekti pildi õiges asukohas.

POST päringu korral võtab *endpoint* JSON formaadis vastu *base64* kodeeringus pildi, pildi ülemise vasaku nurga GPS koordinaadid, pildi alumise parema nurga GPS koordinaadid, pildi tegemise aja ja kuupäeva. Näidis päringu kehast (kodeeritud pildi pikkust on vähendatud) (Joonis 22).

[1] *Endpoint* on kindel API teekond, millele on võimalik esitada päring ning saada vastus

```
{
  "image":
"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAgGBgcGBQgHBwcJCQgKDBQNDAsLDBkSEw8UH
RofHh0aHBwgJC4nICIsIxwcKDcpLDAxNDQ0Hyc5PTgyPC4zNDL/2wBDAQkJCQwLDBgNDRg
yIRwhMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyMjIyM
jL/wAARCAqiBdgDASIAAhEBAxEB/8QAHwAAAQUBAQEBAQEAAAAAAAAAAAECAwQFBgcICQo
L/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhByJxFDKBkaEII0KxwRVS0
fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZnaGlqc3R
1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1
NXW19jZ2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQ
FBgcICQoL/",
  "lat_l": -34.888113,
  "lon_l": -56.220761,
  "lat_r": -34.890044,
  "lon_r": -56.218128,
  "timestamp": "2021-05-10 10:05:12.873679"
}
```


Joonis 22. Näide /api/ai/image *POST* päringu kehast.

Pärast päringu saamist liiguvad andmed objektituvastusse, kus käivitatakse TFLite Interpreter, mis viib läbi *inference*-i[1]. Selleks kasutatake skripti *inference.py*[2]. Skripti on lisatud ka funktsioon tuvastatud objekti geograafiliste koordinaatide tagastamiseks (protsessi on pikemalt kirjeldatud alampeatükis 6.4) ning selles skriptis toimub ka tuvastusandmete andmebaasi salvestamine.

GET päring tagastab info kõikide andmebaasi salvestatud piltide kohta. Joonis 23 kujutab selle päringu vastust (kodeeritud pildi pikkust on taas vähendatud).

[1] *Inference* on protsess, mille jooksul antakse mudelile ette andmed, et arvutada vajalikud väljundid ning seostada need mingi kindla skooriga. Tavaliselt kasutatakse seda väljendit juba juurutatud mudelite korral [33].

[2] https://bit.ly/3vqGnEQ

```
[
    {
        "date_created": "Mon, 17 May 2021 09:59:33 GMT",
        "date_modified": "Mon, 17 May 2021 09:59:33 GMT",
        "encoded_image":
"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAIBAQEBAQIBAQECAgICAgQDAgICAgUEBAMEB
gUGBgYFBgYGBwkIBgcJBwYGCAsICQoKCgoKBggLDAsKDAkKCgr/",
        "id": 28,
        "lat_l": 29.168967,
        "lat_r": 29.168382,
        "lon_l": 121.871615,
        "lon_r": 121.872259,
        "timestamp": "Mon, 10 May 2021 10:05:12 GMT"
    },
    {
        "date_created": "Tue, 18 May 2021 17:58:42 GMT",
        "date_modified": "Tue, 18 May 2021 17:58:42 GMT",
        "encoded_image":
"/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAIBAQEBAQIBAQECAgICAgQDAgICAgUEBAMEB
gUGBgYFBgYGBwkIBgcJBwYGCAsICQoKCgoKBggLDAsKDAkKCgr/",
        "id": 34,
        "lat_l": 29.169251,
        "lat_r": 29.168876,
        "lon_l": 121.867053,
        "lon_r": 121.867522,
        "timestamp": "Sun, 16 May 2021 02:47:46 GMT"
    }
]
```


Joonis 23. Näide /api/ai/image GET päringu vastusest.

6.2 Endpoint /api/ai/detected_object

Endpoint-i /api/ai/detected_object eesmärk on tagastada andmebaasi salvestatud info tuvastatud objektide kohta. *Endpoint*-ist saadavaid andmeid on võimalik kasutada kasutajaliideses, kus saab näiteks kaardile kuvada objekti liikumise trajektoori.

GET päring tagastab info kõikide tuvastatud objektide kohta. Joonis 24 kujutab selle päringu vastust.

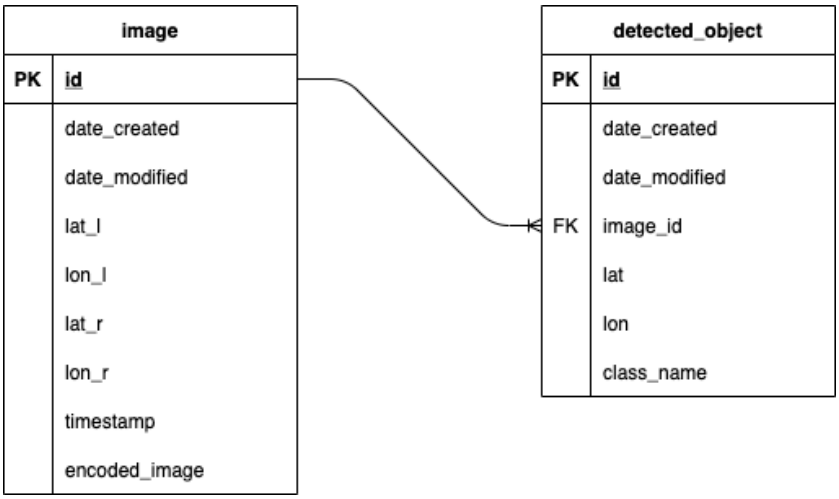
```
[
    {
        "class_name": "cargo ship",
        "date_created": "Wed, 19 May 2021 21:43:13 GMT",
        "date_modified": "Wed, 19 May 2021 21:43:13 GMT",
        "id": 23,
        "image_id": 39,
        "lat": 29.16877161,
        "lon": 121.87177734166667
    },
    {
        "class_name": "cargo ship",
        "date_created": "Fri, 21 May 2021 21:04:41 GMT",
        "date_modified": "Fri, 21 May 2021 21:04:41 GMT",
        "id": 24,
        "image_id": 40,
        "lat": 31.330825672299778,
        "lon": 121.6922261183155
    }
]
```


Joonis 24. Näide /api/ai/detected_object GET päringu vastusest.

Lisades GET päringule URLi parameetri *image_id* (/api/ai/detected_object/<image_id>), tagastab API kindla pildiga seotud tuvastatud objektid.

6.3 Andmemudel ja andmete salvestamine

Selleks, et statistilise analüüsi kasutajaliideses oleks võimalik tuvastatud objektide kohta andmeid kasutada, on need vaja andmebaasis talletada. Süsteemi andmemudel (Joonis 25).



Joonis 25. Süsteemi andmemudel.

Pärast tuvastuse lõppu tehakse tabelisse *image* kirje, kus väli *encoded_image* hoiab tuvastusse jõudnud pilti, millele on lisatud tuvastatud objektide piiriboksid. Pilt salvestatakse taas *base64* kodeeringus. Väljad *lat_l, lon_l, lat_r, lon_r* ja *timestamp* on päringust saadud andmed. Tabelisse *detected_object* tehakse iga tuvastatud objekti kohta kirjed, kus *image_id* on viide *image* kirjele, *lat* ja *lon* on tuvastatud objekti GPS koordinaadid ning *class_name* on tuvastatud objekti klass. Väli *class_name* on prototüübis alati *cargo ship*, kuid väli sai lisatud, kuna edasises arenduses vahetatakse objektituvastuse mudel välja ning see võidakse asendada ka mudeliga, mis mitmeid erinevaid objekte tuvastada suudab.

6.4 Tuvastatud objekti geograafiliste koordinaatide arvutamine

Kuna tuvastusse jõuab pilt koos pildi ülemise vasaku ning alumise parema nurga GPS koordinaatidega, on võimalik leida tuvastatud objektide keskpunktide GPS koordinaadid. Kõigepealt leitakse tuvastusse jõudva pildi pikkus ja laius pikslites, ning seejärel jagatakse pildi parema alumise nurga ja vasaku ülemise nurga laiuskraadi vahe läbi pildi pikkusega pikslites ning pildi parema alumise nurga ja vasaku ülemise nurga pikkuskraadi vahe läbi pildi laiusega pikslites, et saada ühele pikslile vastav GPS koordinaatide arv nii pikkuse kui laiuse kohta.

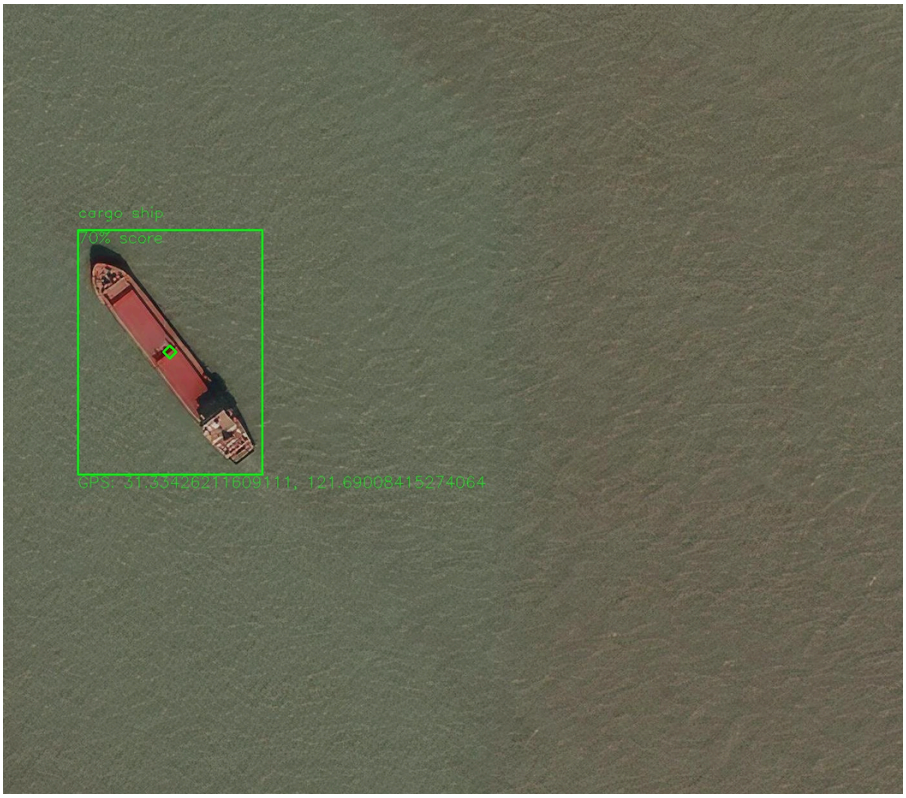
Kuna on teada tuvastatud objekti piiriboksi (ingl. *bounding box*)[1] nurkade koordinaadid pikslites, siis on võimalik määrata piiriboksi keskpunkti koordinaadid pikslites. Kuna mudel tuvastab kaubalaevu, mis on tavaliselt pikliku kujuga, siis laeva keskpunktiks ongi piiriboksi keskpunkt (eeldusel, et piiriboks on küllaltki täpne).

Lahutades pildi vasaku nurga laiuskraadist keskpunkti *y*-koordinaadi ning ühele pikslile vastava GPS koordinaatide arvu korrutise saadakse tuvastatud laeva keskpunkti geograafiline laius. Liites pildi vasaku nurga pikkuskraadile keskpunkti *x*-koordinaadi ning ühele pikslile vastava GPS koordinaatide arvu korrutise saadakse tuvastatud laeva keskpunkti geograafiline pikkus. Küll aga arvestab see lahendus sellega, et pildid on

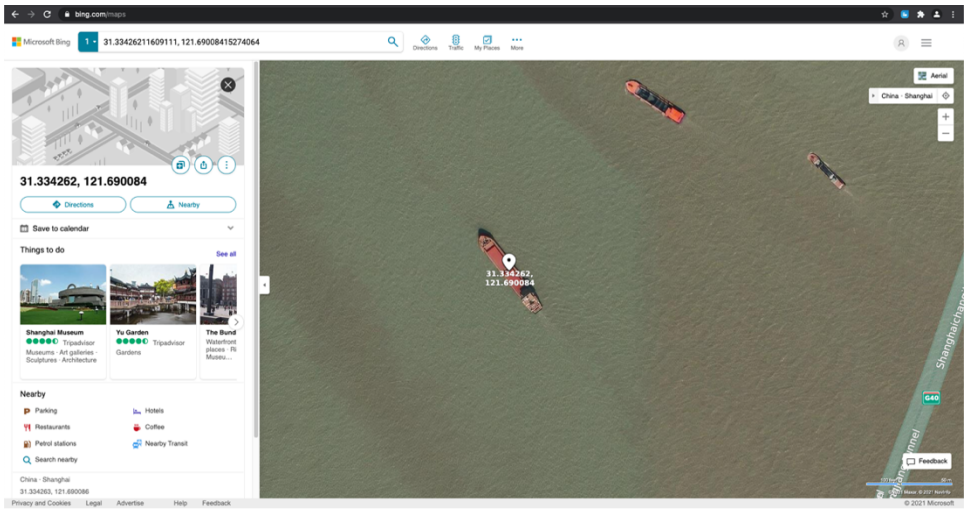
[1] Piiriboksiks nimetatakse kasti, mis tekkib mudeli tuvastuse käigus. Selle abil näitabki mudel, et leitud objekt asub selle piiriboksi sees.

küllaltki väikesemõõdulised, kuna suuremate piltide puhul tuleks arvestada ka Maa kumerusega.

Iga pildil tuvastatud objekti kohta tehakse eelnevalt kirjeldatud arvutus ning tagastatakse objekti keskpunkti geograafilised koordinaadid (Joonis 26). On näha, et kaubalaev asub tagastatud koordinaatidel (Joonis 27).



Joonis 26. Tuvastatud kaubalaev piiriboksi ja geograafiliste koordinaatidega.



Joonis 27. Kuvatõmmis tuvastatud laevast Bing Maps veebilehel.

7 Objektituvastuse mudeli integreerimine statistilise analüüsi kasutajaliidesega

Liikumiste statistilise analüüsi läbi viimiseks on vaja andmeid. Analüüs annab kasulikke tulemusi alates sajast andmepunktist. Analüüsiks on võimalik andmeid küsida andmebaasist, kuid lihtsuse ja kiiruse huvides salvestatakse viimased sada tuvastust ka API sessiooni objekti, mis on sõnastiku (ingl. *dictionary*)[1] andmetüüpi objekt. Sessioon hoiab endas võtmetena sessioonimuutujaid ning väärtustena võtmetele vastavaid andmeid. Selle rakenduse puhul on sessiooni võtmeks *coordinates* ning väärtuseks on omakorda sõnastik, kus võtmeteks on tuvastatud objektide klassi nimed ning väärtusteks järjend (ingl. *list*)[2], mis sisaldab tuvastatud objektide geograafilisi koordinaate ning pildi tegemise aega ja kuupäeva. Joonis 28 kujutab näidet sessiooni objektist.

```
session['coordinates'] = {
    "cargo ship": [
        {
            "latitude": 29.168552820000002,
            "longitude": 121.87205775,
            "timestamp": "2021-05-16 02:47:46.275065"
        },
        {
            "latitude": 29.16877161,
            "longitude": 121.87177734166667,
            "timestamp": "2021-05-16 02:47:46.275065"
        }
    ]
}
```

Joonis 28. Näide API sessiooni objektist.

Pärast igat objektituvastust lisatakse tuvastuse andmed sessiooni ning seejärel kontrollitakse, kas sessioonis on sada objektituvastust. Kui jah, siis kontrollitakse omakorda, kas eelmisest andmete analüüsi saatmisest on möödas tund aega. Kui ei, siis kogutakse andmeid nii kaua, kuni möödunud on tund, sest mida rohkem on andmeid, seda tulemuslikum on analüüs. Kui tingimused andmete analüüsi saatmiseks on täidetud siis

[1] Sõnastikuks nimetatakse andmetüüpi, kus andmeid hoitakse lihtsate võtme-väärtus paaridena.

[2] Järjendiks nimetatakse andmetüüpi, kus andmed hoitakse kollektsioonina ühes kohas. Pythoni programmeerimiskeeles on võimalik järjendisse panna erinevid andmetüüpe, kuid tavapärasemates keeltes peavad järjendi andmed olema ühte tüüpi.

on võimalik mugavalt sessiooni võtme *coordinates* väärtuseks olev sõnastik saata edasi analüüsi APIsse, mis andmetega edasi tegeleb. Seejärel tühjendatakse sessioon ning hakatakse uuesti andmeid koguma.

8 Valideerimine

Valideerimist tehti üldiselt käsitsi või närvivõrkude treenimise/testimise korral ka väikeste skriptide abil.

8.1 Närvivõrkude treenimisaegne valideerimine

Treenimise ajal jooksutati paralleelselt Object Detection APIs sisalduvat valideerimise skripti (Joonis 29), mis uue *checkpoint* faili ilmumisel lasi mudelitest läbi valideerimiseandmestikus olevad pildid ning kuvas TensorBoard kasutajaliideses jooksvad tulemused (Lisa 5). Tegelikult on see sama skript, millega jooksutati treenimist, kuid ühe lisaargumendiga.

```
python object_detection/model_main_tf2.py
    --model_dir='output_model_path'
    --pipeline_config_path='path_to_model_config'
    --checkpoint_dir='path_to_model_checkpoints'
```

Joonis 29. Valideerimise jooksutamine käsurealt.

Valideerimise skripti jooksutamise tulemusena genereeriti kasutajaliidesesse graafikud, mis vastavad COCO (*Common Objects in Context*) tuvastuste meetrikate standardile, mida kasutatakse populaarses COCO objektituvastuse närvivõrkude võistluses [34]. Meetrikate põhjal oli võimalik jooksvalt hinnata mudeli efektiivsust kindlal treenimise sammul (näiteks kui täpne mudel on).

Samuti jälgiti valideerimise ajal ka teist TensorBoardis olevat liidest, milles oli võimalik näha reaalsed mudelite tuvastused valideerimisandmestiku piltide peal (Lisa 7).

Lisaks kasutati palju ära ka vaikimisi treenimisel genereeritud kao funktsiooni tulemusi, mille järgi hinnati, mis suunas (väiksema või suurema kaofunktsiooni väärtuste poole) treenimine parasjagu toimus.

8.2 Närvivõrkude testimisaegne valideerimine

Iseenesest on testimise näol tegemist valideerimisega, kuna selle jooksul hinnatakse mudelite efektiivsust ning täpsust testandmestiku peal, mis peab võimalikult palju sarnanema päris andmetele. Testimisjärgsete tulemuste analüüsi on kajastatud peatükis 5.6.3. Siin aga kirjeldatakse, kuidas tulemused saadi.

Testimise tulemuste analüüsiks otsustati teha mudelitele eksimismaatriksid, mis näitavad tuvastuste õigsust vastavalt sellele, kas kindlal pildid oli või ei olnud soovitud objekti. Oluline on see, et vaadeldi just erinevaid objekte, mitte pilte. Kuna piltide peal võib olla rohkem, kui üks objekt, siis peab arvestama olukordadega, kus mudel tuvastab ainult ühe objekti või vastupidi näeb rohkem objekte kui vaja.

Järgnevalt on kirjeldatud eksimismaatriksi koostamise loogikat:

1) kui pildi peal peab olema vähemalt üks objekt, siis
    a) kui mudel tuvastas täpse objektide arvu, siis pannakse see kirja kui positiivne tulemus;
    b) kui mudel tuvastas rohkem objekte kui vaja, siis
        i) positiivsete tuvastuste alla pannakse kirja õigete objektide arv ja
        ii) valepositiivsete tulemuste alla pannakse kirja tuvastatud ja õigete objektide arvude vahe;
    c) kui mudel tuvastas vähem objekte kui vaja, siis
        i) positiivsete tulemuste alla pannakse kirja tuvastatud objektide arv ja
        ii) valenegatiivsete tulemuste alla pannakse kirja õigete ja tuvastatud objektide arvude vahe;
    d) kui mudel ei tuvastanud ühtegi objekti, siis pannakse see kirja kui valenegatiivne tulemus
2) kui pildi peal ei tohiks olla ühtegi objekti, siis
    a) kui mudel ei tuvastanud ühtegi objekti, siis pannakse see kirja kui negatiivne tulemus;
    b) kui mudel tuvastas kasvõi ühe objekti, siis pannakse valepositiivsete tulemuste alla tuvastatud objektide arv.

8.3 API valideerimine

API valideerimine toimus käsitsi, päringute tegemiseks kasutati Postman[1] rakendust. Kuna rakendus oli piisavalt väikesemahuline, siis piisas valideerimiseks käsitsi

[1] https://www.postman.com/

testimisest. Kindlasti peaks aga API edasi arendamisel kirjutama valmis ka testid API testimiseks.

9 Tulemused

Töö tulemusena sai edukalt arendatud prototüüp rakendusele, mis tuvastab süvanärvivõrkude abil kaugseire pildiandmetest objekte, talletab tuvastuste kohta infot ning pakub võimekuse süsteem integreerida statistilise analüüsi mooduli ning kasutajaliidesega.

Samuti valmis töö tulemusena API, mis võtab vastu pilte koos metadataga ning saadab need edasi objektituvastusse. Lisaks võimaldab API pärida andmeid piltide ning tuvastuste kohta.

Lisaks sai edasi arendatud idee süvanärvivõrkude treenimisest kaugseire piltide peal. Töö käigus treeniti ning testiti olemasolevat objektituvastuse närvivõrku erinevate hüperparameetrite kombinatsioonidega. Katsetati nii erinevate optimeerimisalgoritmide kui ka õppimiskiiruse konfiguratsioonidega. Optimeerimisalgoritmidest osutus kõige efektiivsemaks Adam. Kombinatsioonis *exponential_decay* LR parameetriga oli see teistest mudelitest palju parem.

Arendatud närvivõrgud ning kasutatud andmestik paigutati pilve[1], et tulevased projektiga tegelejad saaksid kasutatud materjalidega edasi tööd teha. Prototüübi arendamisel valminud kood ja ka muud kasutatud materjalid asuvad GitLab repositooriumis[2].

[1] https://drive.google.com/drive/folders/1fHOqrvKJ0kXnHvjCwnV9hRDzJHO-mYvz?usp=sharing.

[2] https://gitlab.cs.ttu.ee/anloba/satellite_ai_image_processing_statistical_analysis

10 Kommentaarid

Tehnoloogiate valik töö tegemiseks enamasti õigustas ennast. Kuigi Flaskiga oli autoritel minimaalne kogemus, toimus arendus lihtsalt ja kiirelt, kuna õppimiskurv ei olnud järsk. Samuti oli API arendusel väga mugav API objektituvastuse mooduliga ühendada tänu sellele, et võimalik oli kasutada Pythonit. Docker tegi arenduse ning kliendile rakenduse testimise väga lihtsaks.

Küll aga tekkis TensorFlow kasutamisel mõningaid probleeme testimisega, kus tekkinud probleemidele oli lahenduste leidmine raske ja aeganõudev. Object Detection API on küll üsna populaarne, kuid mõni osa seal asuvatest skriptidest ei toeta TensorFlow 2 versiooni. Näiteks ei ole hetkel võimalik lihtsalt konverteerida olemasolevat TensorFlow 2 peal tehtud mudelit TFLite formaati, kuna selle käigus tekkivad konfliktid, mille parandamine võtab tohutult aega.

11 Järeldused ja edasised sammud

Edasiste sammudena oleks kasulik APIle testid kirjutada. Samuti oleks hea järgmise sammuna reaalselt objektituvastuse moodul statistilise analüüsi kasutajaliidesega ära ühendada, kuid selleks oleks tarvis kindlaid testandmeid, mille põhjal oleks võimalik analüüsi teha. Samuti peaks tulevikus kindlasti APIt uute *endpoint*idega täiendama, näiteks oleks kasulik tagastada kindlal kuupäeval või kuupäevade vahemikus tehtud tuvastused, kuna analüüsi kasutajaliideses oleks hea sellekohast infot näidata. API täiendamine nõuaks aga kindlasti koostööd kasutajaliidese arendaja(te) ning kliendiga, et edasised nõuded kasutajaliidesele välja selgitada.

Lisaks oleks väga huvitav jätkata erinevate närvivõrkude treenimisega kaugseire piltide peal. Kindlasti saaks lisada rohkem objetke ning katsetada näiteks suurema objektide klasside arvuga. Antud töö tulemusena saadud närvivõrke on võimalik veelgi parandada, kuna tulemused ei ole ikkagi veel ideaalseld. Küll aga tõestab see seda, et kaugseire piltide pealt on võimalik närvivõrkude abil kasulikku informatsiooni kätte saada. Kui arendada seda ideed edasi, siis on kindlasti võimalik saavutada paremaid tulemusi ning ehitada komplekssemaid süsteeme.

Kasutatud kirjandus

[1] "AKIT - Andmekaitse ja infoturbe leksikon." https://akit.cyber.ee/term/8826-base64 (Kasutatud 25 mai 2021).
[2] "JSON." https://www.json.org/json-en.html (Kasutatud 17 mai 2021).
[3] "AKIT - Andmekaitse ja infoturbe leksikon." https://akit.cyber.ee/term/8533 (Kasutatud 25 mai 2021).
[4] "RGB (Red Green Blue) Definition." https://techterms.com/definition/rgb (Kasutatud 17 mai 2021).
[5] J. Hui, "SSD object detection: Single Shot MultiBox Detector for real-time processing," *Medium*, 15 detsember 2020. https://jonathan-hui.medium.com/ssd-object-detection-single-shot-multibox-detector-for-real-time-processing-9bd8deac0e06 (Kasutatud 16 mai 2021).
[6] "Cloud Tensor Processing Units (TPUs)," *Google Cloud*. https://cloud.google.com/tpu/docs/tpus (Kasutatud 16 mai 2021).
[7] "What is WSGI? — WSGI.org." https://wsgi.readthedocs.io/en/latest/what.html (Kasutatud 17 mai 2021).
[8] "Welcome to Flask — Flask Documentation (2.0.x)." https://flask.palletsprojects.com/en/2.0.x/ (Kasutatud 16 mai 2021).
[9] "Welcome to Flask-RESTX's documentation! — Flask-RESTX 0.4.1.dev documentation." https://flask-restx.readthedocs.io/en/latest/index.html (Kasutatud 21 mai 2021).
[10] "tensorflow/models," *GitHub*. https://github.com/tensorflow/models (Kasutatud 16 mai 2021).
[11] "TensorBoard," *TensorFlow*. https://www.tensorflow.org/tensorboard (Kasutatud 24 mai 2021).
[12] "Roboflow: Everything you need to start building computer vision into your applications." https://roboflow.ai (Kasutatud 16 mai 2021).
[13] "Why use PostgreSQL as a Database for my Next Project in 2021," *Fulcrum Blog*, Jan. 21, 2021. https://fulcrum.rocks/blog/why-use-postgresql-database/ (Kasutatud 25 mai 2021).
[14] "PostGIS — Spatial and Geographic Objects for PostgreSQL." https://postgis.net/ (Kasutatud 25 mai 2021).
[15] "What is a Docker Image? Introduction and use cases," *SearchITOperations*. https://searchitoperations.techtarget.com/definition/Docker-image (Kasutatud 25 mai 2021).
[16] WakaTime, "WakaTime · Dashboards for developers," *WakaTime*. https://wakatime.com/about (Kasutatud 17 mai 2021).
[17] L. Chen, *llgeek/Satellite-Aerial-Image-Retrieval*. 2021. Kasutatud: 22 mai 2021. [Võrgumaterjal]. Saadaval: https://github.com/llgeek/Satellite-Aerial-Image-Retrieval
[18] L. Tünts, K. Piven, M. Voskanjan "AI ja piltide töötlemine satelliitide pardal (TPU)," juuni 2020, Kasutatud: 17 mai 2021. [Võrgumaterjal]. Saadaval: https://digikogu.taltech.ee/et/Item/a9e91318-5cdd-4867-bec5-85731dbdea3b
[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.
[20] S. Narkhede, "Understanding Confusion Matrix," *Medium*, 14 jaanuar 2021. https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (Kasutatud 14 mai 2021).
[21] Open Data Science, "Image Augmentation for Convolutional Neural Networks," *Medium*, 29 juuli 2019. https://medium.com/@ODSC/image-augmentation-for-convolutional-neural-networks-18319e1291c (Kasutatud 16 mai 2021).
[22] J. Leonel, "Hyperparameters in Machine/Deep Learning," *Medium*, 7 aprill 2019. https://medium.com/@jorgesleonel/hyperparameters-in-machine-deep-learning-ca69ad10b981 (Kasutatud 20 mai 2021).
[23] "Introduction to Optimizers," *Algorithmia Blog*, 7 mai 2018. https://algorithmia.com/blog/introduction-to-optimizers (Kasutatud 20 mai 2021).

[24] “Optimizers Explained - Adam, Momentum and Stochastic Gradient Descent,” *Machine Learning From Scratch*, 16 oktoober 2019. https://mlfromscratch.com/optimizers-explained/ (Kasutatud 24 mai 2021).
[25] Sanghvirajit, “A Complete Guide to Adam and RMSprop Optimizer,” *Medium*, 14 mai 2021. https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be (Kasutatud 22 mai 2021).
[26] V. Bushaev, “Understanding RMSprop — faster neural network learning,” *Medium*, 2 september 2018. https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a (Kasutatud 22 mai 2021).
[27] J. Duchi, E. Hazan, Y. Singer, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” p. 39.
[28] J. Brownlee, “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning,” *Machine Learning Mastery*, 2 juuli 2017. https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (Kasutatud 19 mai 2021).
[29] J. Brownlee, “How to Control the Stability of Training Neural Networks With the Batch Size,” *Machine Learning Mastery*, 20 jaanuar 2019. https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (Kasutatud 21 mai 2021).
[30] J. Brownlee, “Difference Between a Batch and an Epoch in a Neural Network,” *Machine Learning Mastery*, 19 juuli 2018. https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ (Kasutatud 21 mai 2021).
[31] “TFRecord and tf.train.Example | TensorFlow Core,” *TensorFlow*. https://www.tensorflow.org/tutorials/load_data/tfrecord (Kasutatud 20 mai 2021).
[32] J. Brownlee, “How to Configure the Learning Rate When Training Deep Learning Neural Networks,” *Machine Learning Mastery*, 22 jaanuar 2019. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/ (Kasutatud 25 mai 2021).
[33] “What is Machine Learning Inference?,” *Hazelcast*. https://hazelcast.com/glossary/machine-learning-inference/ (Kasutatud 25 mai 2021).
[34] “COCO - Common Objects in Context.” https://cocodataset.org/#detection-eval (Kasutatud 24 mai 2021).

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

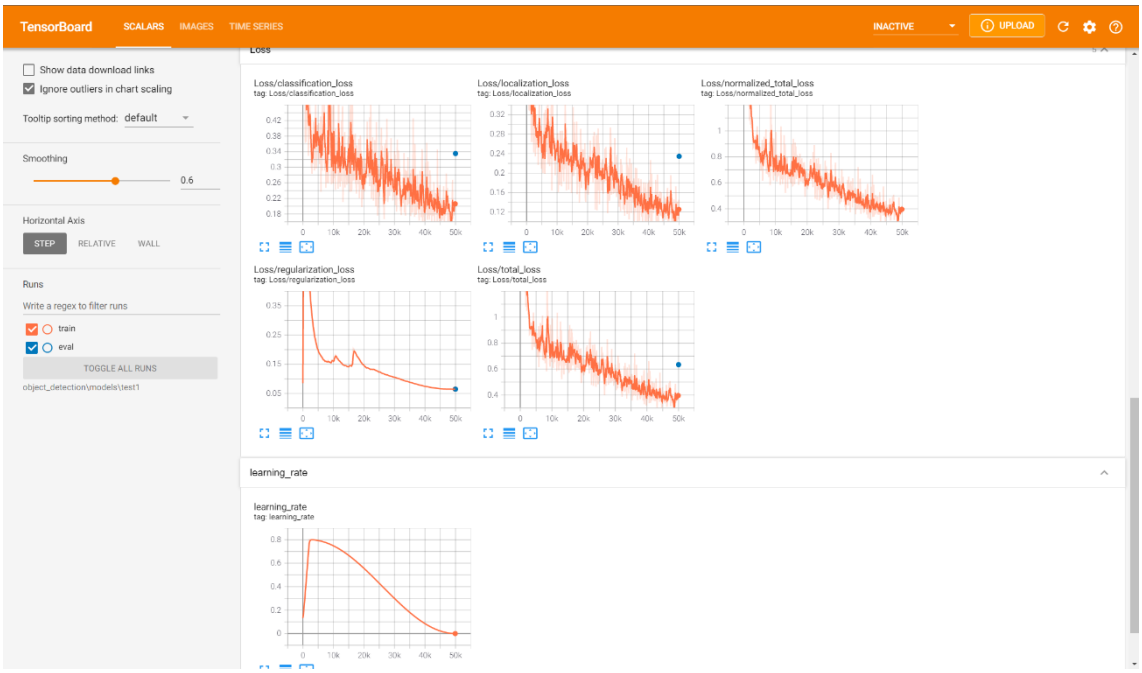
Meie, Siret Jorro ja Alexander Frolov

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Süvanärvivõrkude abil kaugseire pildiandmetest objektide tuvastamine ja integreerimine statistilise analüüsi kasutajaliidesega“, mille juhendaja on Evelin Halling ja kaasjuhendaja on Martin Simon
   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. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

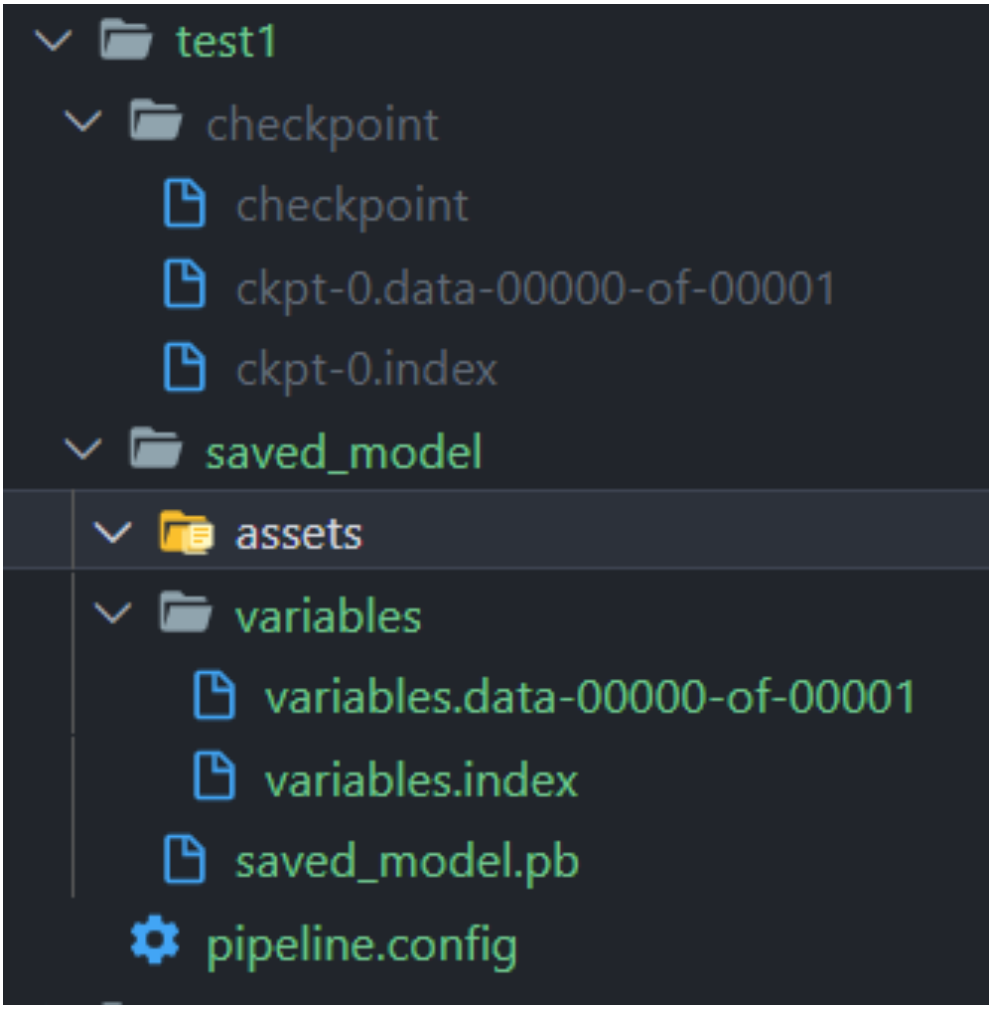
25.05.2021

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

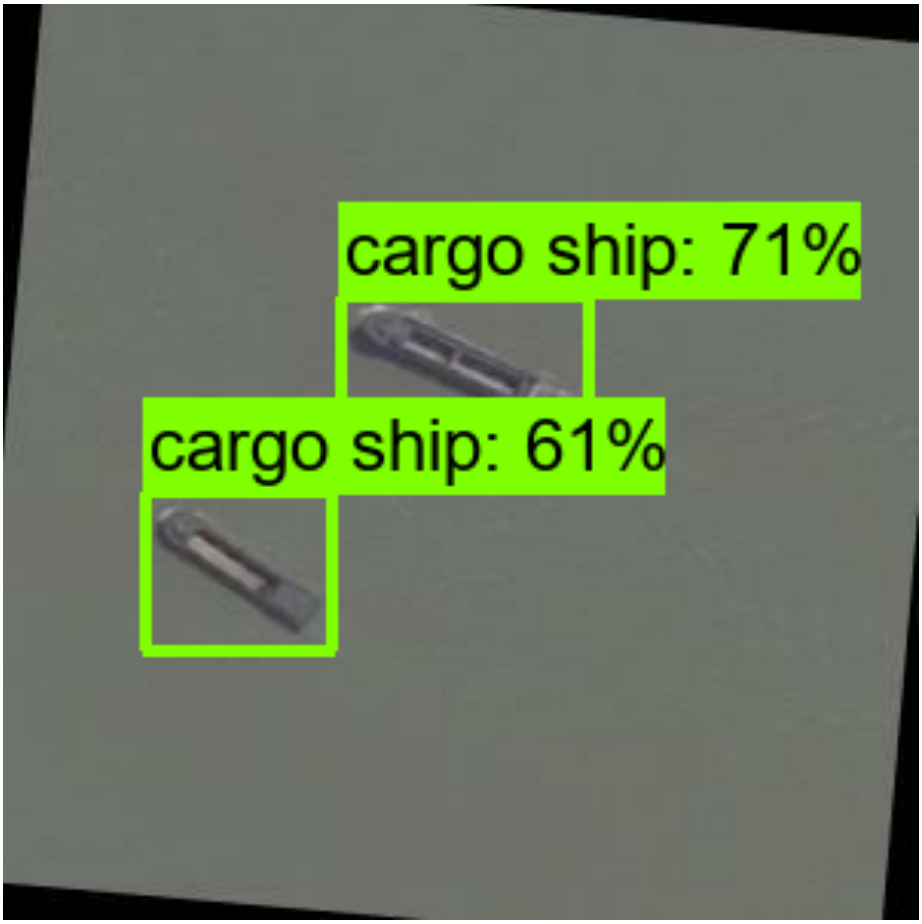
Lisa 2 – Jooksev treenimise aruanne TensorBoard kasutajaliideses



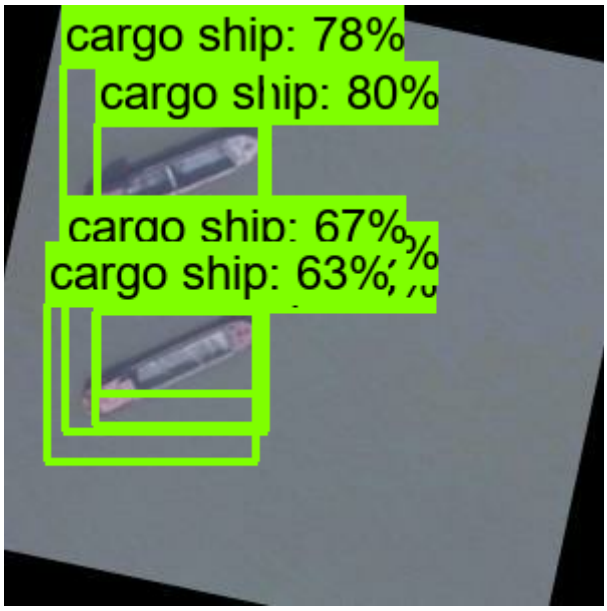
Lisa 3 – Eksporditud mudeli näidis



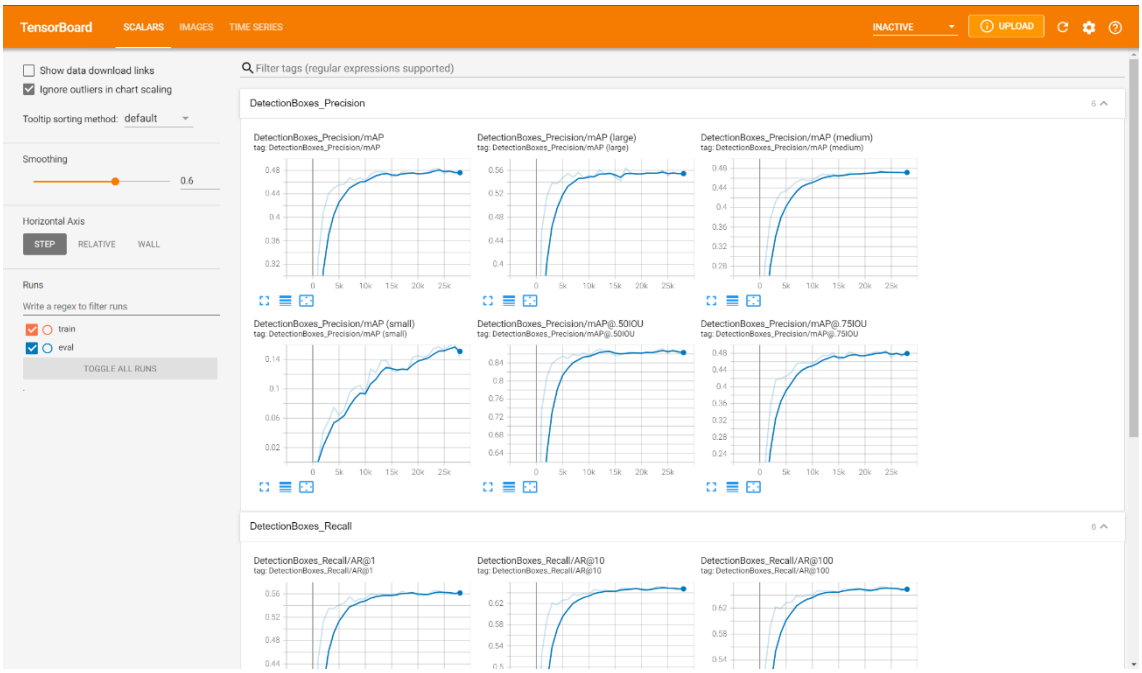
Lisa 4 – Testimisest saadud pilt koos tuvastatud objektidega



Lisa 5 – Näide tuvastusest, kus mudel tegi mitu tuvastust, kuid need käisid ühe kindla objekti kohta



Lisa 6 – Jooskev valideerimise aruanne TensorBoard kasutajaliideses



Lisa 7 – Valideerimise ajal tehtud tuvastuste aruanne TensorBoard kasutajaliideses