TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond
Tarkvaraarenduse instituut

Lilia Tünts 179549IAIB
Katarina Piven 179313IAIB
Marina Voskanjan 179764IAIB

AI JA PILTIDE TÖÖTLEMINE SATELLIITIDE PARDAL (TPU)

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: Lilia Tünts, Katarina Piven, Marina Voskanjan

18.05.2020

Tallinn 2020

Annotatsioon

Antud lõputöö eesmärgiks on treenida ja testida objektide tuvastamise mudeleid, mida oleks võimalik käivitada Coral USB akseleraatori kaudu; uurida erinevate parameetrite väärtuste mõju mudeli edukusele ja mõõta nende jooksutamisel kasutatav energiakulu. Selle järgi tuleb leida, milline mudelitest on kõige parem nii objekti tuvastamise, kui ka energiakulu poolest ning otsustada, mida saab hiljem kasutada satelliidi pardal. Tegemist on masinõppe ja närvivõrkude tehnoloogiate algoritmide põhjal koostatud mudelitega, mille peamine eesmärk otsida mustreid objektide vahel.

Meie valmistasime andmete hulga, mis koosneb 2000 piltidest, mille peal leiduvad lennuki ja lennujaama klassi objektid ning 1000 pilti ilma valitud klasside objektideta. Treenimine toimus tingimustele vastavate kahe mudelite peal: *ssd_mobilenet_v1* ja *ssd_mobilenet_v2*.

Kõigepealt kirjeldame meie töö protsessi, eelkõige kasutatud tehnoloogiad ja abivahendid ning selgitame oma valiku. Jätkame treenimise ja testimise protsesside ja tulemuste mõju erinevate hüper parameetrite muutmise tagajärjel samm-sammulise kirjeldusega. Lõpetame saadud tulemuste analüüsiga ning järeldustee tegemisega.

Töö lõplikuks tulemuseks on analüüs testimisest saadud väärtusest, mille eesmärk on näidata mudeli võimekust tuvastada selle peal treenitud klasside objekti õiget ilmnemist ning juhul, kui objekti pildil ei esine, siis selle õiget puudust. Lisaks, anda tagasiside energiakulust objekti tuvastamise protsessi ajal.

Töö valideerimiseks on mudeli testimise protsess, milles kasutati andmehulga, milleks on 150 markeeritud pilte ja 150 pilte ilma valitud klasside objektideta. Testimiseks eraldatud pildi ei tohi treenimise protsessis osaleda ehk need, millega mudel ei ole varem enne testimist kokku puutunud. Selle järgi uuritakse mudeli võimekus objekte õigesti tuvastada. Töö lõpus on analüüs objektide tuvastamise tulemustest, milles me võrdleme ja hindame mudelite võimekust ja energiakulu.

Mudelite treenimisega seotud projekt ning tehtud töö jooksvad tulemused ja ülevaade on avaldatud aadressil: https://gitlab.cs.ttu.ee/satellite_ai_image_processing/coral

Treenimise GUI rakenduse lähtekood on avaldatud aadressil: https://gitlab.cs.ttu.ee/litunt/model-training-gui

Abivahendina kasutatud tarkvara on avatud lähtekoodiga ning sellele on viidatud kasutatud allikate nimekirjas.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 68 leheküljel, 8 peatükki, 45 joonist ja 25 tabelit.

Abstract

AI and image processing onboard satellites (TPU)

The main goal of this thesis is to train and test object detection models that are runnable with Coral USB accelerator; investigate the impact of various parameters' values on model's performance success and measure the energy used during its launch. As the result of performed experiments we had to discover which model would be the best at both object detection and energy consumption, and decide which one is suitable for further use on the satellite board. We are working with models that are constructed using machine learning and neural nets technology algorithms to find patterns between objects being present in the picture.

We have prepared the dataset that consists of 2000 pictures in which there are objects that belong to plane and airport classes, and 1000 pictures without any objects of chosen classes. Training process was happening with two models that meet all the requirements: *ssd_mobilenet_v1* and *ssd_mobilenet_v2*.

First of all, we describe our working process, foremost used technologies and other facilities, and explain our choice. Then, we continue with training and testing processes step-by-step description, as well as the explanation of the effect that changing different hyperparameters has on the results. We finish with results analysis and conclusion.

The inferential part of the project is the analysis of the values collected from testing. The purpose is to show the model’s capability to detect the object of classes used in training process correctly, along with the model’s ability to correctly ignore other objects. In addition, it is necessary to compose a feedback about energy consumption during object detection process.

We validate our work by testing the models. For testing we have prepared 150 labelled images with plane and airport objects in it, and 150 without chosen classes objects. Pictures participating in testing are not allowed to be used for training. In other words, the model should not have encountered with testing images before testing itself. Such validation helps to research how is the model’s performance correct while discovering objects on unknown surfaces. There is the analysis of object detection result at the end of the paper, in which we compare and assess model’s capability and energy consumption.

The repository of model training project and all the results overview made throughout working process is available at: https://gitlab.cs.ttu.ee/satellite_ai_image_processing/coral

The training GUI application source code is available at: https://gitlab.cs.ttu.ee/litunt/model-training-gui

The software used as additional and/or helping tool is open source and is being referenced in the bibliography section.

The thesis is in Estonian and contains 68 pages of text, 8 chapters, 45 figures, 25 tables.

Lühendite ja mõistete sõnastik

Masinõpe	Arvuti treenimine etteantud andmete, reeglite põhjal otsuseid ja ennustusi tegema, vastavate algoritmide koostamine
YOLO	*You Only Look Once*, reaalaja objekti tuvastamise süsteem [2]
USB akseleraator	USB seade, mis annab Edge TPU lisaprotsessorina arvutile, et kiirendada masinõppe järelduse tegemise protsessi
COCO	*Common Objects in Context*, masinõppes laialt kasutatav andmehulk
SSD	*Single Shot MultiBox Detector*, objektide tuvastamine reaalajas
MobileNet	Masinõppe mudelite tüüp mobiilirakenduses kasutamiseks
pos	Positiivsed pildid
neg	Negatiivsed pildid
Adam	*Adaptive Moment estimation*, momentum ja RMSprop optimisaatorite kombinatsioon
SGD	*Stochastic Gradient Descent*, iteratiivne meetod sobivate siledus omadustega eesmärk funktsiooni optimeerimiseks
LR	*Learning Rate*, õppimiskiirus
RMSprop	*Root Mean Square*, ruutkeskmine

CRC32C	32-bitine CRC, mis kasutab Castagnoli polünoomi e genereeriva funktsiooni
dict	Python programmeerimiskeeles kasutatav räsi tabel, mille struktuur iseloomustatakse võtme-väärtuse paaride olemasoluga
API	*Application Program Interface*, teatud hulk protokolle ja vahendeid tarkvara rakenduse ehitamiseks
GUI	*Graphical User Interface*, graafiline kasutajaliides, graafikakuvamise võimalusi kasutatav tarkvaraliides
DNN	Deep Neural Network, närvivõrk, kus sisendi ja väljundi vahel esinevad mitu alakihti.
TPU	Tensor Processing Unit, programmeeritav AI akseleraator, mis on loodud madala täpsusega aritmeetika suure läbilaskevõime tagamiseks ja on orienteeritud mudelite kasutamisele või käitamisele, mitte nende treenimisele [1]
OOP	*Object Oriented Programming*, objektorienteeritud programmeerimine

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval närvivõrgud ja masinõpe on väga populaarsed teemad infotehnoloogia valdkonnas. Väljatöötatud tehnoloogiad võimaldavad lahendada selliseid ülesandeid, mis on liiga keerulised, et sellele saaks inimene kirjutada programmi arvestades kõike nüansse. Mõned probleemid pole võimalik niimoodi programmeerida. Kasutades suuri andmete kogusid masinõppe algoritmid teevad iseseisvalt uuringu kindla mudeli järgi, et saavutada otsitava vastuse. Näiteks võib kirjutada koodi, mis saaks tuvastada ainult lennukeid kindla vaate ja asendi järgi, aga kasutades tänapäevaseid masinõppe ja närvivõrkude tehnoloogiaid palju efektiivsem oleks ehitada mudel, mis võiks peale korraliku treenimist kaudu tuvastada neid objekte, mille jaoks ta oli treenitud.

Närvivõrgud on oma struktuuriga väga sarnased inimese ajule. Nad on algoritmide kogum, mille peamine ülesanne on mustrite leidmine. Andmete töötlemises närvivõrkude eesmärk on märgistada, klasterdada või rühmitada.[16] See on väga kasulik mudelite treenimises, kui on vaja tuvastada objekte. See on nagu lapsele uue sõna õpetamine piltide kaudu, kus pildid on andmete hulk ja uus sõna on objekt, mida soovitatakse õpetada. Treenimise kaudu laps näeb mitu korda erinevaid lennukite pilte ning talle öeldakse, et antud objekt on lennuk. Kindla hetke pärast laps näeb mustri , kus objekt on kindlate osadega on sarnane lennukiga ning juba edaspidises kokku puutes koos objektiga, saab ta tuvastada seal lennuki. Sama on närvivõrkudega, nad jaotuvad objekti mitmeks osadeks ja õpivad, et antud objektil on niisugused osad. Järgmisena saades testimise ajal objekti ja võrreldes seda koos endale tuttavatele objektidele, saab närvivõrk vastata, kas teab, mis antud objekt on või ei tea. Tavaliselt objektide tuvastamiseks kasutatakse tõenäosust. Kui mudel on rohkem kui 80% kindel ja tagastab õige otsuse objekti klassi kohta, võib öelda, et mudel töötab ja programm õigesti töötab.

Objekti tuvastamine muutub aina kasutatavamaks. Tänapäeval pakuvad paljud erinevad ettevõtted satelliidi saatmise võimalust nagu näiteks SpaceX Ameerikas. Eesti ettevõte Marduk Technologies OÜ kavatsevad tulevikkus saata satelliidi kosmosesse. Tegemist on feasibility study 3U satelliidiga, millel oleks võimekus teha kõrgresoga pilte, optiline resolutsioon annaks kätte 1m/px. Satelliit hakkab tegema teleskoop-optikaga pilte maast teatud intervalli tagant ning saatma need pildid tagasi. Satelliidi riistvarale paigutatud programm, AI’ks on DNN ehk sügav närvivõrgud,

peaks eelkõige tegema otsuse, kas kaameraga tehtud pildil on üldse nõutud objekt olemas. AI filtreerib välja vajalikud pildid ning saadab need maale. Selle otsuse tegemiseks firma on valinud Google Research poolt pakutava Coral USB akseleraatori, mis on võimeline tuua masinõppe järeldused olemasolevatesse süsteemidesse. Selle jaoks on juba olemas palju mudeleid objektide tuvastamiseks, aga nad ei saanud töötada pealtvaates olevate piltidega. Näitena toome välja, et laevasadam, testides seda koos mudeliga, mis pidi teoreetiliselt tuvastama seda, oli mudeli järgi pitsa tükk. Marduk Technologies pakkus meile uurida, mis mudel sobib paremini satelliidi jaoks ja milline nendest vajab energiat kõige vähem.

1.1 Ülesande püstitus

Projekti käigus tahaksime lahendada probleeme, mis eksisteerivad praegu juba olemasolevatel objektide tuvastamise algoritmidel.

Esimeseks probleemiks on objektide tüübid, mida on YOLO suuteline tuvastada. Kuna YOLO on treenitud maa peal ehk oskab tuvastada objekte vertikaalses asendis ehk külgvaates, satelliidi pealt oleksid tema jaoks kõik objektid ühesugused, kuna tegemist on horisontaalse asendiga ehk pealtvaatega maa peale.

Teiseks ülesandeks võiks olla erinevate juba olemasolevate mudelite võrdlus, mille käigus ja saadud tulemuste järgi oleks võimalik otsustada, millist mudelit edasi tellija töö käigus saaks treenida.

Kolmandaks on uurida, milliste mudeli arhitektuuri hüper parameetrite kaudu treenimine toimib kõige paremini. Tänu sellele teadmisele on võimalik luua kõige efektiivsema mudeli, mis järelikult saab objektide tuvastamisega paremini hakkama.

Nagu üleval pool oli kirjeldatud, satelliidi riistvaral on kindel piiratud energia maht, kogukulu ca 10-40W, mida peab jätkuma nii õigeaegse objekti tuvastamisele, kui ka pildistamisele ja pildi ära saatmisele. Selle jaoks peab mõõtma, kui palju energiat läheb objekti tuvastamisele iga valitud mudeli käivitamisel, võrdlema saadud tulemused ja otsustama, mille mudeli energiakulu on kõige madalam.

2 Tehnoloogiate valik ja rakendamine

2.1 Tensorflow

TensorFlow on avatud lähtekoodiga masinõppe tarkvarateek, mis kasutab arvutuste tegemiseks andmevoo graafe. Kuigi selle peamine eesmärk on seotud masinõppe ja tehisnärvivõrkude uurimisega, on TensorFlow piisavalt üldine, nii et seda saab rakendada ka muudes valdkondades [1].

TensorFlow looja on Google. See oli esialgu mõeldud ettevõttesiseseks kasutuseks, et see oleks abiks teadustööde kirjutamisel ja uute tehnoloogiate väljatöötamisel. Selle lähtekood tehti avalikkusele kättesaadavaks 2015. aasta 9. novembril [1].

Meie kasutasime oma projektis TensorFlow Lite versiooni, sest nimelt sellega töötab Coral USB akseleraator.

2.2 Bing Maps pildid

Piltide tegemiseks kasutasime avalikult kätte saadava rakenduse suure resolutsiooniga satelliidi abil tehtud piltide allalaadimiseks. Rakendus võimaldab etteantud koordinaatide piires tekitada ja automaatselt alla laadida õhupilte maksimaalse võimaliku resolutsiooniga, arvestades laius- ja pikkuskraadi [18].

2.3 LabelImg

LabelImg on rakendus, mida kasutasime objektide markeerimiseks pildi peal. Programmi abil saab joonistada objekti ümber nelinurkse raami ja kirjutada markeeritud objekti klass. Objekti asukoha koordinaate ja sellele ruudule vastava klassi salvestamiseks saab valida YOLO ja PASCAL VOC formaatide vahel. Meie oleme valinud PASCAL VOC formaadi, sest see sobib kõige paremini Coral USB akseleraatori kasutamise korral [17].

2.4 Tensorflow Object Detection API

Mudelitega opereerimiseks kasutasime avaliku lähtekoodiga TensorFlow peale ehitatud raamistiku. Selle abil saab iseseisvalt ehitada, treenida ja laadida mudelid üles. Object Detection on laialt kasutatav põhi mudelitega tegutsemiseks üle maailma [19].

3 Projekti kirjeldus

Meie projekti eesmärgiks on treenida ja võrrelda sobivad objekti tuvastamise mudelid ning valida pärast võrdlemist parimad selleks, et panna valitud mudel töötama satelliidi pardal, mis saaks tuleviku plaanis tuvastada nõutud objekte.

Ülesandeks oli valida antud tingimustes sobilikud mudelid ning töötada just nendega. Tingimused on järgmised:

- mudel peab olema treenitud tuvastama objekte pealtvaatest,
- ei tohiks võtta palju energiat,
- sellele sobilik riistvara peaks samamoodi olema hästi väike,
- energia mõttes mitte väga kulukas,
- samal ajal piisavalt kiire, sest satelliit liigub hästi kiiresti.

Selleks, et saavutada Marduk Technologies OÜ poolt püstitatud eesmärki ja leida kõige täpsema ja energia säästvama mudeli, oli vaja:

- valmistada treenimise ja testimise andmete hulga, mis kujutab ennast markeerituid pilte koos objektide klassifitseerimisega,
- uurida TensorFlow poolt pakutavad mudelid ja valida just need, mis sobiksid Coral-i peal jooksutamiseks,
- treenida ja testida mudelid,
- mõõta mudeli jooksutamisel kasutatav energiakulu.

3.1 Andmete ettevalmistamine

Mudelite treenimiseks tavaliselt valmistatakse suur hulk pilte. See on aja nõutav manuaalne protsess, mida kahjuks ei ole võimalik automatiseerida. Pilte peab olema kaks tüüpi:

- positiivsed - need pildid, mille peal on leitavad objektid, mida mudel treenitakse tuvastama

- negatiivsed - need pildid, kui ka negatiivseid ehk neid, kus objekte pole ning mille kogus peab olema täpselt pool positiivsete piltide arvust

Mida suurem on materjali kogus, seda paremini saab treenida mudeli ning järelikult objekti eduka tuvastamise tõenäosus kasvab.

3.1.1 Parima andmehulga valiku tegemine

1. Klasse ei tohi olla palju, eriti väikeste piltide hulga korral. See oluliselt mõjutab objektide ja nende raamide tuvastamist. Kui klasse on palju, siis tuvastamise täpsus kõikide objektide puhul hakkab langema ning tulemus ei ole järelikult piisavalt täpne. Projekti raames meie tegelesime kahe klassiga - lennuk ja lennujaam.

2. On parem, kui pildid on erineva suurusega ja objektid ise on ka erineva suuruse ja kujuga. Kuid meie poolt valitud mudelite tüüp koosneb paljudest kihtidest, mis skaleeruvad pilte ja ise muutuvad pilde meetrikaid. Järelikult, erineva suurusega piltide tegemine ei ole kohustuslik, sest mudel ise muudab kõik üheks resolutsiooniks.

3. Negatiivsed pildid tõstavad mudeli stabiilsust ja täpsustavad objekti kirjeldust.

4. Kasutada lisaks ka selliseid pilte, kus objekt on kujutatud osaliselt. Aitab korjata rohkem infot objektist ja selle välimusest.

3.2 Mudelid

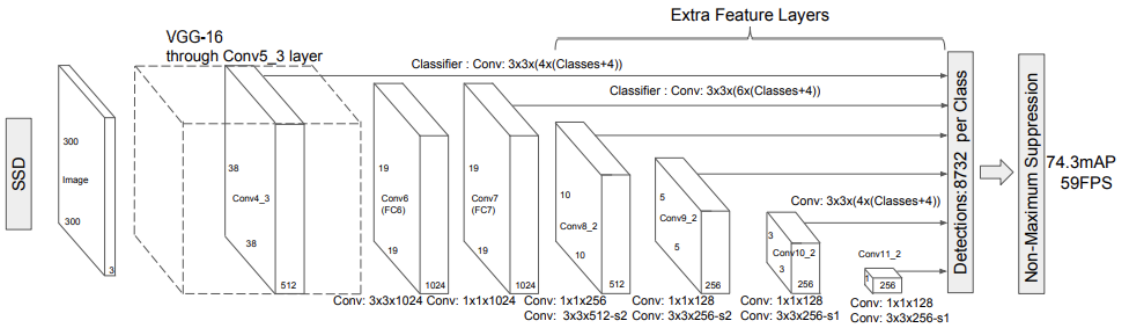
TensorFlow pakub erinevate eelnevalt koostatud ja treenitud mudelite kogust. Need mudelid erinevad teineteisest peamiselt arhitektuuri järgi. Seoses sellega, et Coral USB akseleraatoriga töötades valik on väga piiratud, pidime pöörduma Coral-i ametlikul veebilehel kirjutatud mudelite variantidele. Selleks sai TensorFlow poolt valmistatud eelnevalt treenitud mudelite nimekiri. Kuigi mudelid on juba treenitud COCO andmehulga põhjal, neid saab ilusti ka ümber treenida. Meie jaoks oli oluline saada vajalikud arhitektuurid kätte.

3.3 Energiakulu mõõtmine

Energia kulukuse kohta me otsustasime leida USB vahendit, mis võiks mõõta energiat testimise ajal. Selleks pöördusime Thomas Johann Seebecki elektroonikainstituudi poole.

3.4 SSD mudeli treenimise vajalikud meetrikad. Mudeli kihid

3.4.1 Treenimise protsess ja mudeli kihtide treenimine



Joonis 1. SSD mudeli arhitektuur

SSD mudel koosneb mitmetest kihtidest (Joonis 1). Iga kiht töötleb määratud klasse, korjab nendest andmeid ning muudab kaalusid ja täpsuse kaotuse näidu. Kihtide erinevus seisneb skaleerimises. Iga kiht teeb pildi väiksemaks ja pärast seda korjab info klassist, mis on määratud pildi peal. Need andmed salvestatakse vastava kihi väljundi sees. Väljundite arv sõltub konfiguratsioonist [5].

Kõike kihte treenitakse proportsionaalselt. Ainukene aspekt, mis võib muuta tulemust on esialgsed andmed, mille peal treenimine toimub.

4 Projekti sisu

Projekti töö oli mitmekesine ja hõlmas suures mahus nii teoreetilise aluse loomist, kui ka tehniliste ülesannete täitmist. Lisaks uurisime, kuidas töötavad närvivõrgud, sest varem pole sellega kokku puutunud.

4.1 Piltide otsing

Sobivate piltide otsimiseks leidsime kõik maailmas tegutsevad lennujaamad. Tööd jagasime maailmaosade järgi grupi liikmete vahel. Lennujaamade koordinaatide otsing ja piltide kätte saamine võttis palju aega. Lisaraskust tekitas ka lennukite ilmumine, kuna mõnede koordinaatidega tulid pildid kas ilma lennukita või pildi allalaadimine üldse ei õnnestunud allika poolse keelu tõttu.

Kui pilt sai alla laetud, siis seda oli vaja lisaks lõigata mitmeteks tükkideks sellepärast, et satelliidi pildil on suur resolutsioon. Nagu oli varem mainitud, mudel hakkab treenimise protsessi käigus seda veel väiksemaks tegema ning kokku surutud objektid ei ole enam korralikult eristatavad.

4.1.1 Piltide tegemise protsessi kiirendamise võimalus

Üldiselt koordinaatide sisestamine toimub käsurealt. Jäjrest sisestatakse ülemine vasak koordinaat (lat1 lon1) ja alumine parem koordinaat (lat2 lon2):

```
python aerialImageRetrieval.py lat1 lon1 lat2 lon2
```

Sellega märgitakse ala, millest tehakse pilt. [18]

Selle protsessi optimiseerimiseks tegime lühikese juhendi, mis on meie variant selle lahendamiseks.

1. Leia endale sobiv pind satelliitpildi salvestamiseks. Võta selle koordinaadid ning vali üks järgmist:

    a. arvuta ülemine vasak koordinaat. Salvesta need *.txt* failina "59.656565 24.800283". Kirjuta kood, mis arvutaks alumise parema koordinaadi ja tekitaks see juurde

    b. arvuta ise kohe ülemine ja alumine koordinaadid ja kirjuta faili

2. Kirjuta kood, mis loeb valminud faili ridade kaupa (üks rida kujutab endas kõik neli vajalikku punkti)

3. Selles sama koodis loo *AerialImageRetrieval* klassi objekt ja sisesta sinna järjest koordinaadid (selline koodi osa on leitav rakenduse vastavas failis)

4. Vaikimisi programm salvestab pildid automaatselt tekitavasse *output* kausta sama faili nimega. Selleks, et failid ei asendaks teineteist, esialgses koodis tuleb muuta nimetamise protsessi

    ```
    os.path.join(self.tgtfolder,'aerialImage_{}.jpeg'.format(levl))
    ```

    lisades nime formateerimise näiteks indeks, mis genereeritakse kas selles samas koodi osas, või mida antakse koordinaatidega kaasa (teine variant oleks isegi parem)

4.2 Treening andmete ettevalmistamine

Siis, kui pildid said töödeldud, pidime markeerima piltidel olevaid objekte. Negatiivsete piltide korral oli vaja ka panna “tühja” markeeringu ja tekitada vastav XML fail, et treenimise jooksul mudel saaks sellest “tühjusest” aru ning teeks järelduse, et selle pildi objektid kindlasti ei kuulu ühtegi määratud klassi. Lõpuks oli vaja markeerida kokku 2000 positiivset ja 1000 negatiivset pilti.

4.3 Mudelite valik

Kõige sobivamaks variandiks osutusid SSD tüübi MobileNet mudelid. Selle põhjuseks on see, et SSD mudelid on ainukesed, mida hetkel on võimalik konverteerida *.tflite* ehk TensorFlow Lite formaadiks, mis on omakorda ainuke formaat, mida toetab Coral USB akseleraator. MobileNet

arhitektuur sobib meie eesmärgi raames väga hästi, sest need on mõeldud mobiilsetes rakendustes kasutamiseks, mis on vähem kulukamad energia mõttes ning ei ole suurusest väga mahukad.

Alguses arvestasime sellega, et meil on olemas ligi kümme mudelit. Kuid töö jooksul pidime tegema muudatuse mudelite valikus. Kui hakkasime täpsemalt valituks osutunud mudelite hulgas konfiguratsioonid uurima, siis tuli välja, et ainult kaks sobivad just treenimise protsessi teostamise ja arhitektuuri mõttes.

Näiteks, mõned mudelid töödeldavad piltide andmed määratud suuruse vahemikus ehk *min_shape* ja *max_shape* vahel. Aga *.tflite* formaadiks sobib ainult *fixed_shape_resizer* suuruse muutmise parameeter. Mõne mudeli puhul selle arhitektuur nõudis moodustatud TFRecord failide järjekorda. See viis ei ole enam TensorFlow teegi poolt toetatud. Üldistades, mõnede mudelite struktuur oli vananenud ning TensorFlow teek on loobunud nende kasutamisest, järelikult need mudelid ei olnud võimalik ümber treenida meie sobivate piltide ja klassidega.

Lõpuks me saime kätte kaks mudelit, millega jätkasime tööd: **ssd_mobilenet_v1** ja **ssd_mobilenet_v2**. Erinevused:

- **ssd_mobilenet_v2** kasutab *depthwise convolution*[1]
- **ssd_mobilenet_v2** on kolm kernelit, **ssd_mobilenet_v1** kasutab ühte
- **ssd_mobilenet_v2** *min_negatives_per_image*[2] on 3, **ssd_mobilenet_v1** on 0

4.4 Energiakulu mõõteriist

Pärast kohtumist elektroonikainstituudi esindajaga ja seadet ülevaatamist jõudsime järeldusele, et instituudi poolt pakutav vahend on natuke liiga põhjalik meie töö jaoks. Seega Marduk Technologies OÜ tellis meile USB pinge mõõteriista, mida ühendatakse arvuti ja USB seadme vahel.

[1] konvolutsioon, mis tegeleb mitte ainult ruumi dimensiooniga, vaid ka sügavuse dimensiooniga ehk kanalite arvuga. Pildil neid kanaleid on 3: RGB. Peale konvolutsioone rakendamist pildile, pildil võib ilmuda mitu kanalit.

[2] parameeter, mis määrab võimalike negatiivsete objektide olemasolu pildil (mitte otseselt täpse arvu, aga nö nende võimalik kaal)

5 Treenimise protsessi kirjeldus

Treenimise tingimused iga mudeli jaoks olid samad antud tingimuste all. Muutmata tingimusteks terve protsessi käigus olid:

- ajaperiood - selleks valisime ühe terve ööpäeva ehk 24 tundi
- arvuti, mille peal treenimist käivitasime - DELL Latitude 7490, Intel Core i7-8650U, 1.9-2.1 GHz

5.1 Treenimise etapid

Erinevad katsed mudelitega olid tehtud kahes treenimise etapis.

Esimeses etapis põhirõhk oli andmehulga suuruse mõju uurimisele, kuidas treenimiseks eraldatud andmehulga suurus mõjutab mudeli poolt väljastava tulemust.

Teise etapi eesmärgiks oli uurida mudeli arhitektuuris sisalduvate hüper parameetreid tuunides nende väärtused ning leida, kuidas see mõjutab nii tulemust testimisel, kui ka tarbitava energuakulu.

5.1.1 Esimene etapp - erinevad andmehulgad

Katsete tegemise esimeses etapis keskendusime erinevate andmehulkade peal treenimisel. Selleks meil olid valmistatud järgmised piltide koguste variandid:

- 1000 pos
- 1000 pos ja 500 neg
- 2000 pos
- 2000 pos ja 1000 neg

Nende katsetega üritasime tõestada, et mida suurem ja mitmekesisem on andmehulk, seda kõrgem on objekti tuvastamise õige tõenäosus.

5.1.2 Teine etapp - hüper parameetrite väärtused

Teises etapis tegelesime arhitektuuri hüper parameetrite uurimisega. Tegime katsed erinevate parameetrite väärtustega. Kõige mõjuvamaks ja oluliseks parameetriks valitud mudelite arhitektuuris on *optimizer*, mille idee on täiustada mudeli kiirust ja tootlikkust. Nendest on olemas kolm tüüpi: *Adam, Momentum* ja *RMSprop*. See arhitektuuri osa vastutab peamiselt LR näidu määramise eest. Lisaks, uurisime *depth_multiplier*[3] ja *batch_size*[4] muutujate väärtuste mõju tuvastamise tulemusele. *depth_multiplier* vaikimisi väärtus mudelitel oli 1, *batch_size* vaikimisi väärtus - 24.

5.1.2.1 Learning rate

Õppimiskiirus on selline hüper parameeter, kui palju närvivõrgu kaalud praegusel hetkel korrigeeritakse arvestades kao gradiendi väärtust [7][6].

5.1.2.2 Adam optimizer

Stohhastiline gradiendi laskumise meetod, mis põhineb esimese ja teise järgu hetkede kohanemisvõimeline hindamisel. Selle optimiseerimise algoritmi kohta võib öelda, et tegemist on RMSprop ja SGD koos Momentum algoritmiga kombinatsiooniga. Adam on selline õppimiskiiruse meetod, mis arvutab individuaalsed LR väärtused erinevate parameetrite jaoks. See meetod kasutab esimest ja teist gradiendi momente[5], et adapteerida õppimiskiirus iga närvivõrgu kaalu jaoks [11] [12] [13].

5.1.2.3 Momentum optimizer

Selline optimeerimise meetod, mis aitab SGD kiirendada vastavas suunas ja summutab võnkumisi, lisaks hetke sammu gradiendile käsitleb ka eelnevate sammude gradiente, et valida liikumise suunda [8] [11] [13].

---

[3] muutuja, mis kasutatakse kanalite arvu vähendamiseks iga närvivõrgu kihi jaoks

[4] selle muutuja väärtus määrab, mitu valimi tükki korraga antakse närvuvõrgule analüüüsimiseks

[5] suvalise muutuja n-moment on selle muutuja eeldatav väärtus astmes n

5.1.2.4 RMSprop optimizer

RMSprop meetodi põhiidee seisneb ruutgradientide liikuva keskmise[6] hoidmine iga kaalu jaoks. Tulemuse saavutamiseks gradiendi väärtus koos kulu funktsiooniga jagatakse ruutkeskmise väärtusega (*root-mean-square*). Tänu sellele arvutusviisile antud optimisaator sai enda nime [11] [14]. See oli vaikimisi optimeerimise meetod kõikides valitud mudelites.

5.2 Treenimiseks ettevalmistamine

Iga allalaetud mudeliga kaustas on leitavad nii juba treenitud mudelid, kui ka esialgne arhitektuuri konfiguratsiooni fail, mille põhjal saab uuesti mudelit treenida enda poolt valmistatud andmetega. Enne selle seadistamist on olemas veel mõned sammud, mis järgnevad andmete ettevalmistamisele.

5.2.1 Klasside fail

Treenimise protsessi jooksul mudel peab saama vaatama ja kontrollima, kas pildil oleva objektile külge pandud klass on õige ja on üldse soovitud klasside nimekirjas olemas. Selleks on vaja käsitsi kirjutada *.pbtxt* formaadis tekstifaili, mille sisu hakkab nägema *dict* taolise kujul välja. Iga klassi jaoks kirjutatakse *item* objekt, mis sisaldab kaks võtmet: *id* ja *name*, mis on vastavalt klassi numbriline identifikaator (antakse järjest alates numbrist 1) ning klassi nimi (peab olema sama, nagu markeeritud failides kirjas).

5.2.2 TFRecord failide genereerimine

TFRecord faili formaat on lihtne formaat binaarsete kirjete järjestuse salvestamiseks. Iga kirje sisaldab bait-sõnet kasulike andmete[7] (*data-payload*) jaoks, andmete pikkust ja CRC32C räsid stabiilsuse kontrollimiseks [9].

Kõigepealt, markeerimise tulemuse failidest[8] tuleb luua *.csv* failid, vastavalt kas testi või treenimise jaoks mõeldud andmetega (Lisa 3). Nendes failides tekitatakse TFRecord formaadi

---

[6] arvutusmeetod, mille abil leitakse üldine trend teatud andmehulga vältel; numbrite hulga iga alamhulga keskmine väärtus

[7] kasulikud olulised andmed failis

[8] *.xml* formaadis failid, mis tekkisid pildist selle pildil olevate objektide markeerimisest

loomise jaoks mugav struktuur - iga objekti kohta pannakse kirja selle pildi faili nimi, pildi suurus (kõrgus ja laius), klass, millele objekt kuulub ning markeerimise raami koordinaadid pildi suhtes (alumine vasak koordinaat ja ülemine parem koordinaat). Programmikoodiga (Lisa 4) käiakse kõik read läbi ning ühest sellest reast genereeritakse üks binaarne kirje ning kokku need read moodustavad järjendi, millest omakorda kujuneb TFRecord fail. TFRecord failide genereerimise koodi käivitasime konsoolis käsuga, kirjutades vastavalt kas *test* või *train* andmete tüübi:

```
python generate_tfrecord.py --csv_input=data/test_labels.csv  --
output_path=test.record
```

5.2.3 Konfiguratsiooni allikate määramine

Nüüd on vaja määrata mudeli konfiguratsiooni failis need allikad, kust treenimise ajal mudel hakkab vajalikud andmed koguma:

- *num_classes* - meie klasside arv (2).
- *fine_tune_checkpoint* - teekond **model.ckpt** failide kogusele mudeli kaustas. Selliseid faile on kokku kolm: *.ckpt-data, .ckpt-meta, .ckpt-index*. *Data* osa sisaldab kõik muutujate väärtused ilma struktuurita, millest on võimalik taastata mudelit. *Meta* osas on metagraaf ehk arvutus graafi struktuur ilma muutujate väärtusteta. *Index* faili töö on siduda kaks eelmist faili omavahel.
- *label_map_path* - klasside sõnastiku faili asukoht.
- *train_input_reader -> input_path* - treenimise andmetest moodustatud TFRecord faili asukoht
- *eval_input_reader -> tf_record_input_reader -> input_path* - test andmetest moodustatud TFRecord faili asukoht

5.3 Treenimise protsessi käivitamine

TensorFlow Object Detection API pakub mitu võimalust mudeli treenimise kävitamiseks. Valik tehakse mudeli omaduste järgi. Meie juhul sobivamaks tundus kasutada Object Detection repositooriumi kaustas *model_main.py* faili koodi.

Samamoodi, nagu teiste protsesside käivitamine, treenimist jooksutasime konsoolis käsuga,

```
python model_main.py --logtostderr --model_dir=training/ --
pipeline_config_path=training/pipeline.config
```

kus

- `--model_dir` - kausta asukoht, kuhu soovime salvestada treenimise jooksul tekitavate *checkpoint* ehk sammude andmed,
- `--pipeline_config_path` - mudeli konfiguratsiooni (*.config* fail) asukoht.

Protsessi käigus konsooli prinditakse kõik hetke sammuga seotud informatsioon, näiteks: kao väärtus, hetke sammu järjekorranumber, sammule vastav õppimiskiirus jne (Lisa 5).

5.4 Treenimise järgsed sammud

Coral USB akseleraator nõuab väga konkreetset valminud mudeli formaati. Selleks on *.tflite* ehk TensorFlow Lite formaadiks konverteeritud mudel. Enne lõpptulemuseni jõudmist on vaja tekkinud treenimise sammudest tekitada mitu järeldusgraafe enne, kui genereeritakse mudel ise soovitud formaadis.

5.4.1 Järeldus- ehk tulemusgraaf

Järeldusgraafi genereerimine. Teiste sõnadega, objekti tuvastamise mudeli eksportimine tulemuse näitamise seisundiks. Selle jaoks kasutasime samamoodi Object Detection API poolt tehtud programmi, jooksutades seda konsoolist. Kuna lõpuks peame saama nimelt *.tflite* formaadis mudeli, siis tulemusgraaf peab ka olema juba vastavas formaadis genereeritud.

```
python export_tflite_ssd_graph.py --input_type image_tensor /
--pipeline_config_path training/pipeline.config /
--trained_checkpoint_prefix training/model.ckpt-XXXX /
--output_directory inference_graph
```

“XXXX” asemel on vaja sisestada treenimise ajal tekkinud viimase sammu number, sest see sisaldab kõike vajalikku infot. Tulemusena saame kask faili: otseselt graafi *.pb* fail ja graafi struktuuri tekstifail.

5.4.2 Mudeli genereerimine

Vaatamata loodud graafi formaadile, treenitud mudeli kasutamiseks on vaja tekitada sellest mudeli fail, mida saab edaspidi Coral USB akseleraator lugeda. Selleks kasutasime TensorFlow teegiga kaasaskäiva konverterit, mille käsu jooksutasime samamoodi konsoolis.

```
tflite_convert --graph_def_file=path/to/tflite_graph.pb /

--output_file=converted_model.tflite /

--input_format=TENSORFLOW_GRAPHDEF /

--output_format=TFLITE /

--input_shapes=1,300,300,3 /

--input_arrays=normalized_input_image_tensor /

--
output_arrays=TFLite_Detection_PostProcess,TFLite_Detection_PostProcess:1,TFL
ite_Detection_PostProcess:2,TFLite_Detection_PostProcess:3 /

--allow_custom_ops
```

- `--graph_def_file` - teekond graafi *.pb* failile,
- `--input_arrays` ja `--output_arrays` - sisendi ja väljundi tensorite[9]nimekirjad,
- `--input_shapes` - asetsevad kihtide ja tensorite vahel. Meie valitud mudelitel on 2D konvolutsioonilised kihid ning sellisel juhul sisend koosneb neljast parameetrist: *batch_size*, pildi pikkus, pildi laius, katalite arv (meie juhul 3 - RGB). *batch_size* väärtuseks pannakse sama number, mis on leitav konfiguratsiooni failis.

5.5 Treenimise tingimuste ja tulemuste analüüs

Tabel 1 ja Tabel 3 näitavad vastavate mudelite treenimise tulemusi, kui treenimisel on kasutatud järgmised vaikimisi väärtused:

- depth_multiplier - 1.0,
- batch_size - 24,
- initial_learning_rate - 0.00400000018999,
- optimizer - rms_prop_optimizer.

Tabel 2 ja Tabel 4 näitavad vastavate mudelite treenimise tulemusi, kui treenimisel kasutatud piltide kogus oli 2000 pos ja 1000 neg ning pildi suurus oli 300x300.

[9] andmestruktuuri tüüp, mida kasutatakse lineaaralgebras ning mis aitavad teostada aritmeetilisi operatsioone [15]

Tabel 1. Treenimise tulemused erinevate andmehulkadega (ssd_mobilenet_v1)

sammud	pos piltide arv	neg piltide arv	pildi suurus
10005	1000	0	500x500
3345	1000	500	500x500
10056	2000	0	300x300
9586	2000	1000	300x300

Tabel 2. Treenimise tulemused erinevate parameetritega (ssd_mobilenet_v1)

sammud	optimizer	depth_multiplier	batch_size	initial_learning_rate/ learning_rate_base
9830	adam	1.0	24	0.0005
11524	momentum	1.0	24	0.2
13885	momentum	0.75	24	0.2
	adam	1.0	30	0.0005

Tabel 3. Treenimise tulemused erinevate andmehulkadega (ssd_mobilenet_v2)

sammud	pos piltide arv	neg piltide arv	pildi suurus
7068	1000	0	500x500
2800	1000	500	500x500
7293	2000	0	300x300
7682	2000	1000	300x300

Tabel 4. Treenimise tulemused erinevate parameetritega (ssd_mobilenet_v2)

sammud	optimizer	depth_multiplier	batch_size	initial_learning_rate/ learning_rate_base
8394	adam	1.0	24	0.001
8450	momentum	1.0	24	0.001
9435	momentum	0.75	24	0.001
	adam	1.0	30	0.0005

6 Testimise protsessi kirjeldus

Testimise protsessi käigus me saime hinnata treenitud mudeli täpsust ja analüüsida võimalikke takistusi parima tulemuse saamiseks, et vältida neid tulevikus.

Kõik testimise tulemused on fikseeritud tabelites, igale pildile vastab tuvastamise protsessi võimsus, keskmine kuluv aeg, arvutatud energiakulu, lennujaama klassi ja lennuki klassi tõenäosused. Kõikide andmete põhjal on koostatud info üldistatavad tabelid, kus on lühikokkuvõte mudeli testi kohta: minimaalne, maksimaalne ja keskmine klassi tuvastamise tõenäosus ja energiakulu ning tulemuse klassifitseerimise tõenäosused.

6.1 Testimiseks ettevalmistamine

6.1.1 Kasutatud vahendid

Testimiseks kasutasime USB Voltage Meter mõõteriista (Joonis 2) ja Coral USB accelerator tööriista (Joonis 3).

Joonis 2. USB Voltage Meter

Joonis 3. Coral USB accelerator

Corali USB akseleraatori peal jooksutatakse testimise protsess. Ettevalmistamine on ametliku Coral'i veebilehel [4], kus on antud kõik juhendid, mille järgi on võimalik käivitada pildi tuvastamise programmi.

Seoses sellega, et tellija soov oli veel määrata objekti tuvastamise protsessi energiakulu, tellisime USB Voltage Meter seadme. Selle seadmega saime mõõta, kui palju võtab voolu Coral USB akseleraator arvutiga ühendamisel ja tuvastamise protsessi käivitamisel.

USB Voltage Meter oli ühendatud arvuti sisse, mille peal käivitasime tuvastamise protsessi ning mõõteriista sisendiks oli Coral USB akseleraator (Lisa 1).

Protsessi käivitamine toimub terminali kaudu. Terminalis kirjutatakse, kui palju pildi tuvastamisele aega läks ja milline on tuvastamise tõenäosus, juhul objekt leitakse üles. Vajadusel on võimalik saada markeeritud pilt, kus Coral'iga on märgitud tuvastatud objekt (Lisa 2).

6.1.2 Testimise protsessi optimeerimine

Testimisel on vaja teha palju manuaalset tööd, mis lisab koormust juurde ning oluliselt aeglustab kogu protsessi. Tänu sellele, et Coral'i poolt edastatavad programmid on kirjutatud Python programmeerimiskeeles, oli võimalik natukene muuta koodi nii, et kõik tulemused jooksvalt saaks automaatselt panna faili kirja. Lisaks see võimaldas arvutada tuvastamisele kuluva keskmise aja.

Pärast treenimist on vaja arvutada keskmised tõenäosused. Keskmised tõenäosused arvutatakse juhul, kui mudel leidis ühe pildi peal mittu objekte. keskmise väärtuse arvutamine aitas mugavalt saadud tulemused tabelisse lisada.

Optimeerimise käigus sai tehtud peamine fail *loop.py* (Lisa 7). Programm itereerib konverteeritud mudelid läbi ja genereerib mudeli, klasside dokument ja testpildi relatiivse asukoha, mis edastatakse Coral'i failile *detect_image.py* (Lisa 8).

Detect_image.py koodis muutsime meetodi argumentid, sest enam ei olnud vaja kasutada terminali kaudu edastatud parameetreid, vaid sai võtta meie poolt genereeritud. Lisasime koodi osad, mille abiga oleks võimalik kirjutada failidesse saadud tulemused.

Pärast testimist, saadud tulemused konverteerime kasutades *converter.py* (Lisa 9) faili, kus loetakse dokumenti *results.txt* sisu programmi sisse ja töödeldakse tulemused, mida oleme juba tabelisse lisanud.

Terminalis oli tarvis käivitada *loop.py* faili käsuga:

```
python3 loop.py
```

6.1.3 Test andmete moodustamine

Esialgselt testimine toimus 8 pildi peal, aga peale väikese uuringu oli arusaadav, et sellest kogusest ei piisa usaldusväärsuse testimiseks. Mudeli testimise andmete hulga kogum peab olema vähemalt 10% treenimise andmete hulga kogusest ning koosnema sellistest piltidest, mis ei osalenud treenimisel. Negatiivsete ja positiivsete piltide jaotus peab olema 1:1 ehk kui mudeli treenimisel kasutatakse 1000 pilte, siis sama mudeli testimises tuleb kasutada 100 pilti kokku, millest 50 on negatiivset ja 50 positiivset.

6.1.4 Energiakulu arvutamine

Energiakulu arvutatakse valemiga (1):

$$Võimsus\ (W)\ \cdot Keskmine\ tuvastamises\ kuluv\ aeg\ (s) = Energiakulu\ (Ws) \quad (1)$$

Tuvastamise protsessi jooksul Coral käivitab mudelit viis korda, millest esimesel korral laadib mudeli ja pildi. Sellest lähtudes, tuvastamisele kuluva aja keskmise väärtuse arvutasime viimastest neljast tulemusest.

6.1.5 Tulemuste klassifitseerimine

Tulemuste klassifitseerimisel kasutatakse neli klassi, mille järgi saab hinnata, kui õigesti sai mudel pildi töödelda:

- True Positive (TP) ehk õige positiivne — mudel tuvastab objekti klassi õigesti, mudeli poolt leitud objektile määratud klass vastab objekti kujule;
- False Positive (FP) ehk vale positiivne — mudel tuvastab objekti klassi valesti, mudeli poolt leitud objektile määratud klass ei vasta objekti kujule;
- True Negative (TN) ehk õige negatiivne — mudel ei tuvasta objekti klassi õigesti, mudeli poolt ei ole ühtegi objekti pildilt leitud, kui need objektid ei kuulu treenitud klasside hulka;
- False Negative (FN) ehk vale negatiivne — mudel ei tuvasta objekti klassi valesti, mudeli poolt ei ole objekti pildilt leitud, kuid pildil kujutatud objektid kuuluvad treenitud klasside hulka [10].

Näiteks, kui pildi peal on kass lennuk ja mudel tuvastas seda lennukiks — TP. Juhul kui mudel tuvastas seda teise klassina või ei tuvastanud üldse — FP. Tingimusel, kui mudel tuvastas lennuki pildil, aga päriselt lennukit seal ei ole — FN, aga kui ei tuvastanud — TN.

6.2 Erinevate andmekogustega mudelite testimise tulemused ja analüüs

Testimises oli kokku 150 positiivset ja 150 negatiivset pilti.

Esimeste mudelite testimise eesmärk oli uurida, kui suur treenimisel osalenud andmekogus peab olema parema tulemuse saavutamiseks ning tõestada, et mida rohkem pilte on mudel treenimise ajal kasutanud, seda kõrgem kasvab mudeli õige tuvastamise tõenäosus.

Proovisime neli erinevat varianti iga mudeli jaoks (kokku kaheksa treenitud mudelit lõpuks):

1. 1000 positiivset pilti;
2. 1000 positiivset ja 500 negatiivset pilti;
3. 2000 positiivset pilti;
4. 2000 positiivse ja 1000 negatiivse pilti;

Tabel 5. Erinevate andmekogustega testimise kirjeldus

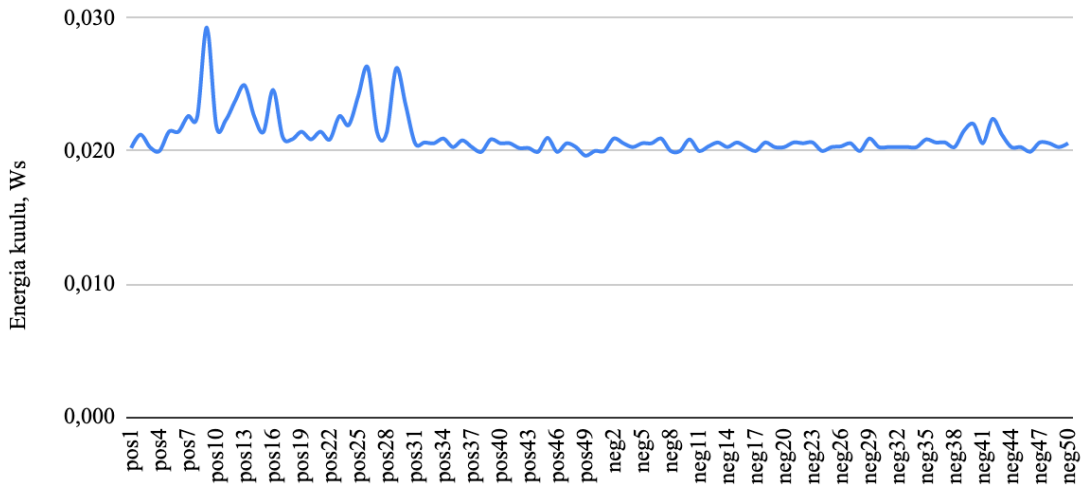
mudel	piltide kogus testimiseks
ssd_mobilenet_v1 1000 pos	50 pos ja 50 neg
ssd_mobilenet_v1 1000 pos 500 neg	75 pos ja 75 neg
ssd_mobilenet_v1 2000 pos	100 pos 100 neg
ssd_mobilenet_v1 2000 pos 1000 neg	150 pos 150 neg
ssd_mobilenet_v2 1000 pos	50 pos ja 50 neg
ssd_mobilenet_v2 1000 pos 500 neg	75 pos ja 75 neg
ssd_mobilenet_v2 2000 pos	100 pos 100 neg
ssd_mobilenet_v2 2000 pos 1000 neg	150 pos 150 neg

6.2.1 ssd_mobilenet_v1 tulemused

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v1* treenitud mudeli tulemustest erinevate andmekogustega.

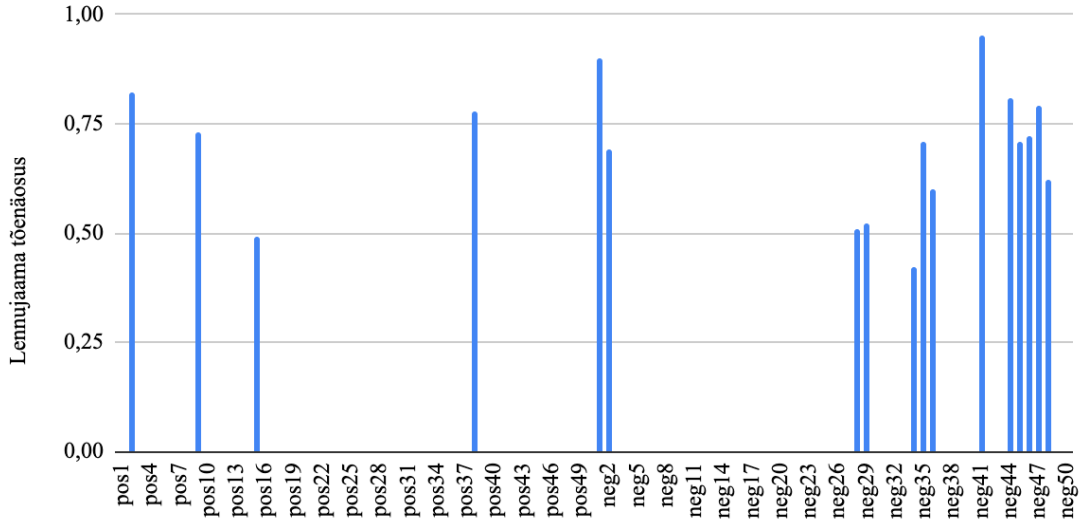
6.2.1.1 ssd_mobilenet_v1 1000 pos

Mudeli *ssd_mobilenet_v1* 1000 positiivsete piltide korral energiakulu iga testpildi kohta (Joonis 4).



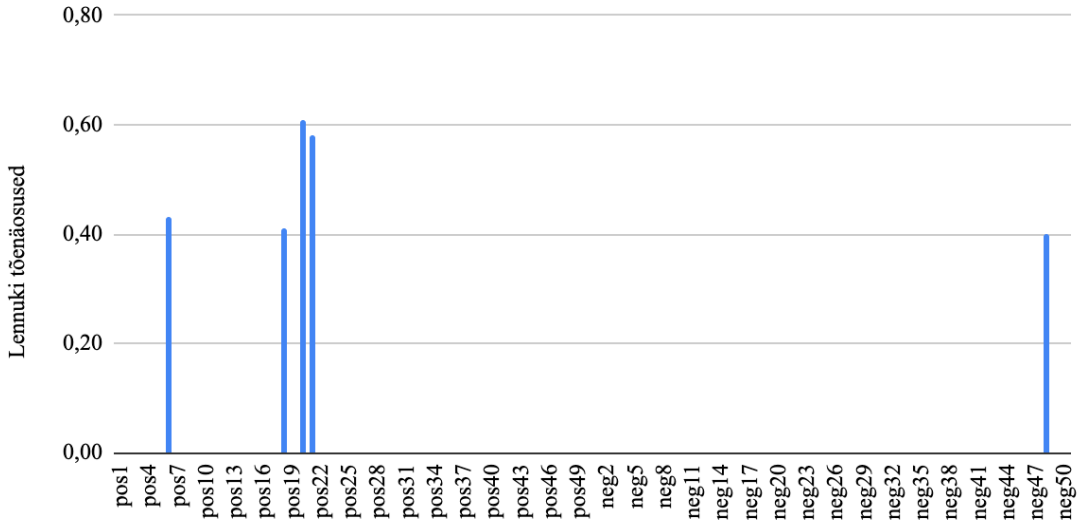
Joonis 4. ssd_mobilenet_v1 1000 pos energiakulu tulemused

Mudeli *ssd_mobilenet_v1* 1000 positiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 5).



Joonis 5. ssd_mobilenet_v1 1000 pos lennujaama tõenäosused

Mudeli *ssd_mobilenet_v1* 1000 positiivsete piltide korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 6).



Joonis 6. ssd mobilenet_v1 1000 pos lennuki tõenäosused

Tabel 6. Mudeli ssd_mobilenet_v1 1000 pos testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,41	0,61	0,51	0,08
pos lennujaam	0,49	0,82	0,705	0,08
pos energiakulu	0,020	0,029	0,022	
neg lennuk	0,4	0,4	0,4	0,02
neg lennujaam	0,42	0,95	0,67	0,26
neg energiakulu	0,020	0,022	0,021	

Selle mudeli treenimisel kasutasime ainult 1000 positiivset pilti ehk need, mille peal on lennujaam või lennuk kindlasti olemas.

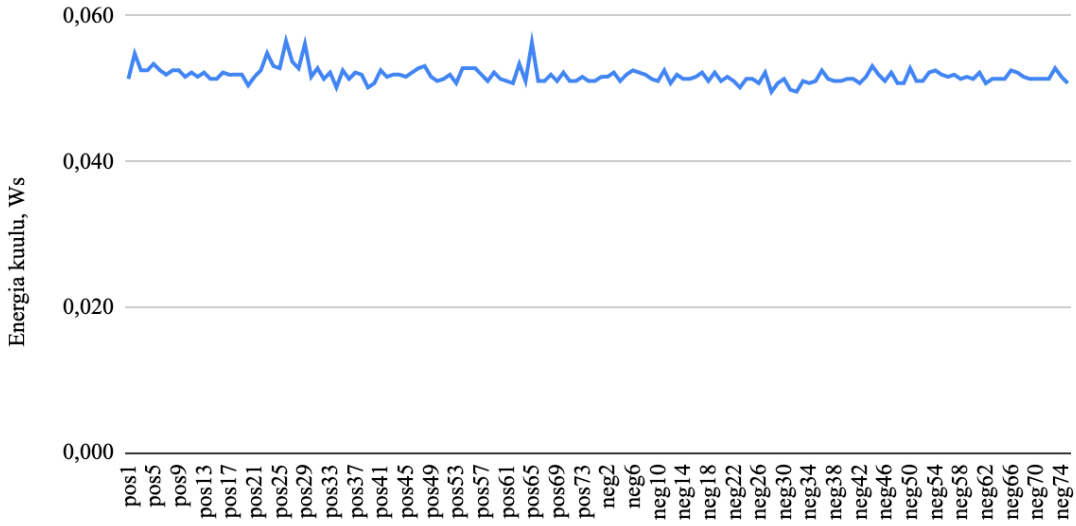
Graafiku järgi saime teada, et energiakulu on stabiilne, kuid esimese 35 pildi jooksul energiakulu oli kõige kõrgem. Keskmine energiakulu positiivsete piltide tuvastamisel oli 0,022 Ws ja negatiivsete puhul 0,021 Ws.

ssd_mobilenet_v1 (1000 pos) mudeli TP tõenäosus oli ainult 8% lennujaama ja 8% lennuki piltide peal, mis ei ole hea tulemus. Samal ajal lennuki FN tõenäosus tuli päris väike välja - 2% ning lennujaama FN - 26%.

Üldiselt lennujaama tuvastamise tõenäosus oli kõrgem. Keskmine lennujaama tuvastamise tõenäosus positiivse piltide peal on 71% ja lennuki puhul 51% (Tabel 6).

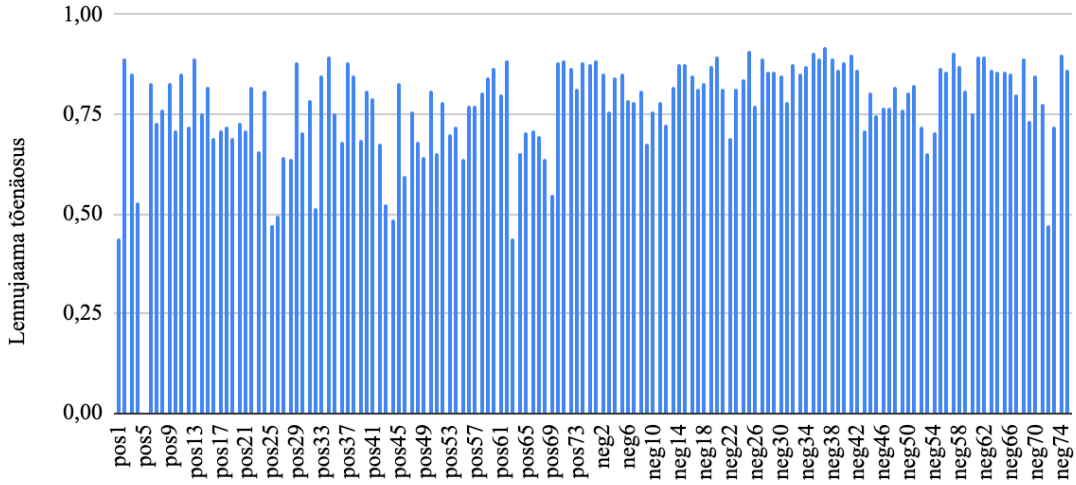
6.2.1.2 ssd_mobilenet_v1 1000 pos 500 neg

Mudeli *ssd_mobilenet_v1* 1000 positiivsete ja 500 negatiivsete piltide korral energiakulu iga testpildi kohta (Joonis 7).



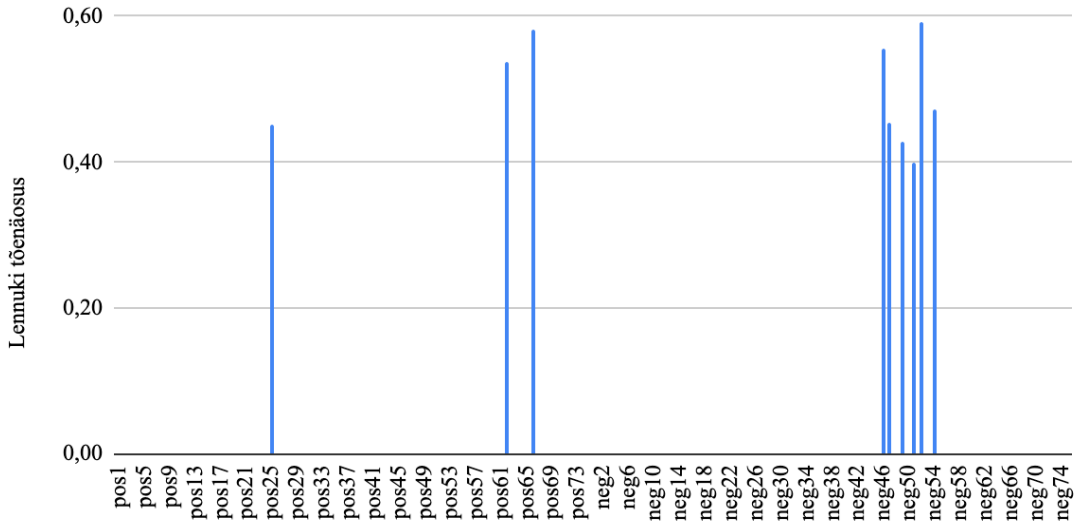
Joonis 7. ssd mobilenet_v1 1000 pos l500 neg energiakulu tulemused

Mudeli *ssd_mobilenet_v1* 1000 positiivsete ja 500 negatiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 8).



Joonis 8. ssd mobilenet_v1 1000 pos 500 neg lennujaama tõenäosused

Mudeli *ssd_mobilenet_v1* 1000 positiivsete ja 500 negatiivsete piltide korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 9).



Joonis 9. ssd mobilenet_v1 1000 pos 500 neg lennuki tõenäosused

Tabel 7. Mudeli ssd_mobilenet_v1 1000 pos 500 neg testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,45	0,58	0,52	0,04
pos lennujaam	0,44	0,90	0,73	0,99
pos energiakulu	0,050	0,057	0,052	
neg lennuk	0,40	0,59	0,48	0,08
neg lennujaam	0,47	0,92	0,82	1
neg energiakulu	0,050	0,053	0,051	

Selle mudeli treenimisel kasutasime andmekoguses lisaks 500 negatiivset pilti ehk need, mille peal ei leidu lennukit ega lennujaamat. Peamiselt kasutasime meie klasside objektidega satelliidi kaamera jaoks sarnaseid alasid, näiteks sadamat või staadioni.

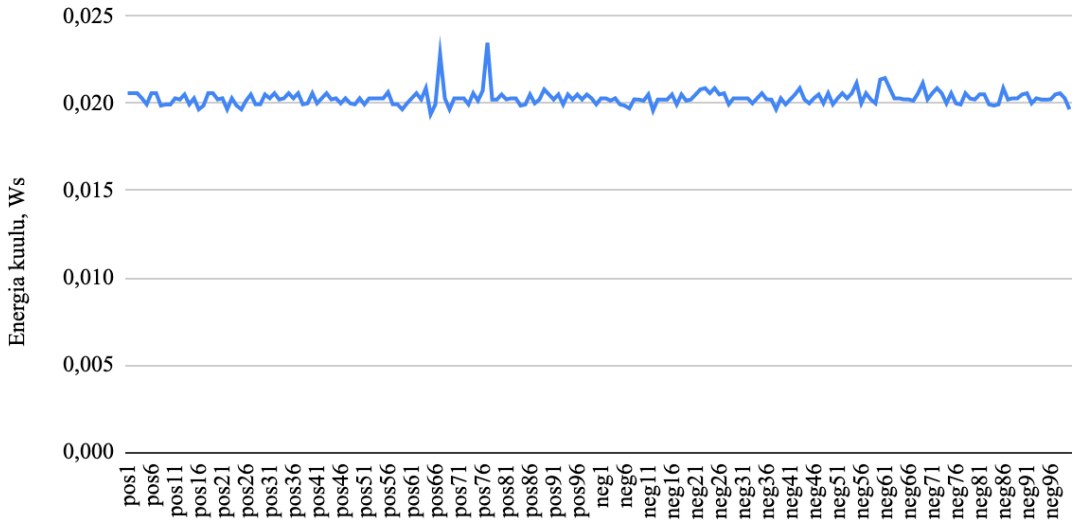
Graafikute peal on näha, et lennujaama tuvastamise TP protsent oli 99%, kuid sama ajal FN tõusis 100% peale. Lennuki tuvastamise TP sel korral oli halvem kui eelmise mudeliga - ainult 4% ning FN eelmisega tõusis ka mõne protsendi võrra ja sel korral oli 8%.

ssd_mobilenet_v1 (1000 pos 500 neg) lennuki ennustamise keskmine tõenäosus positiivse piltide peal oli 52%, lennujaama tõenäosus - 73%.

Positiivsete ja negatiivsete piltite energiakulu kahekordistus ja keskmiselt tõusis 0,05 Ws väärtuseni (Tabel 7).

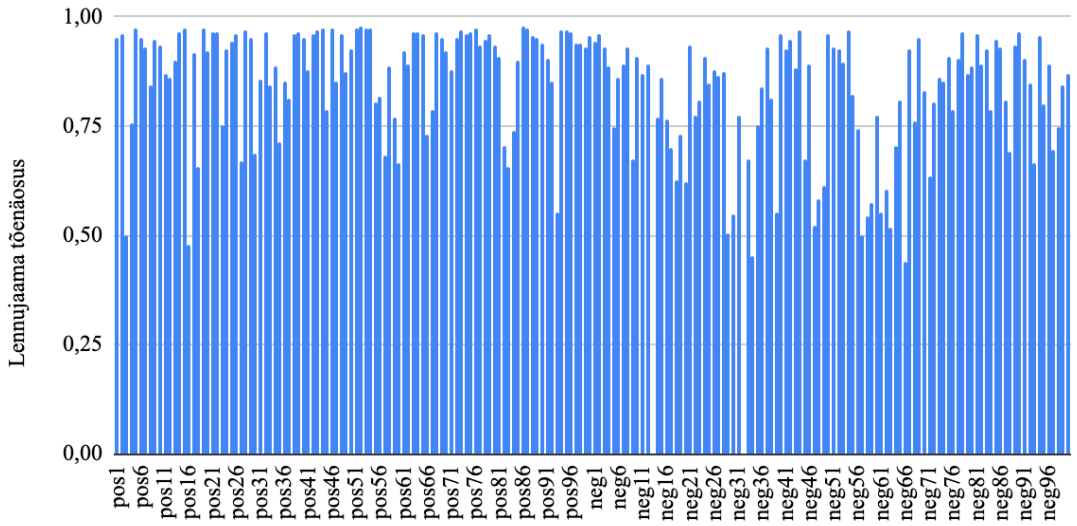
6.2.1.3 ssd_mobilenet_v1 2000 pos

Mudeli *ssd_mobilenet_v1* 2000 positiivsete piltide korral energiakulu iga testpildi kohta (Joonis 10).



Joonis 10. ssd mobilenet_v1 2000 pos energiakulu tulemused

Mudeli *ssd_mobilenet_v1* 2000 positiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 11).



Joonis 11. ssd mobilenet_v1 2000 pos lennujaama tõenäosused

Tabel 8. mudeli ssd_mobilenet_v1 2000 pos testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0	0	0	0
pos lennujaam	0,48	0,98	0,89	1
pos energiakulu	0,019	0,023	0,020	
neg lennuk	0	0	0	0
neg lennujaam	0,44	0,97	0,80	0,98
neg energiakulu	0,020	0,021	0,020	

Selle mudeli treenimisel kasutasime 2000 positiivset pilti. Lennukeid sellel korral mudel ei tuvastanud üldse.

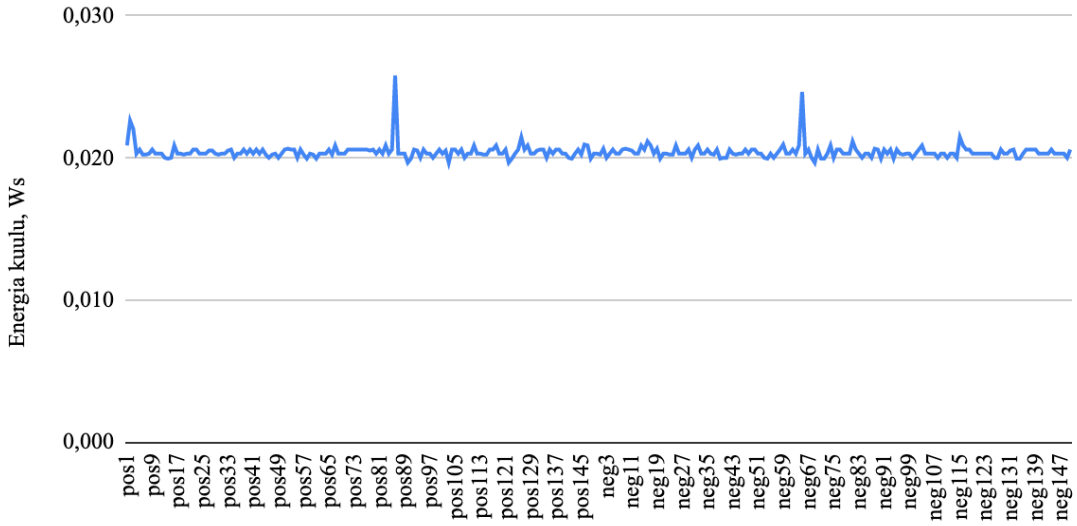
Lennujaamad said tuvastatud 100% TP tõenäosusega. Sama väärtus, mis ei ole hea mudeli tulemus, on ka FN tuvastamise korral, mille tõenäosus on peaaegu sama kõrge- 98%.

ssd_mobilenet_v1 (2000 pos) nüüd suudab ennustada lennujaamat keskmise tõenäosusega 89%.

Positiivsete ja negatiivsete piltide puhul keskmine energiakulu tuli välja sarnane esimese testimisega, mille korral kasutasime samamoodi ainult positiivseid pilte - 0,02 Ws (Tabel 8).

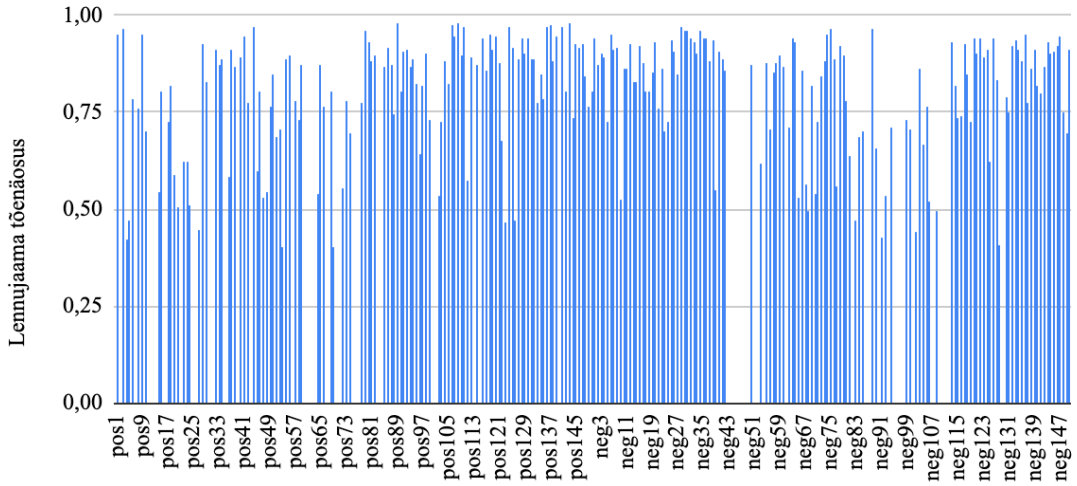
6.2.1.4 ssd_mobilenet_v1 2000 pos 1000 neg

Mudeli *ssd_mobilenet_v1* 2000 positiivsete ja 1000 negatiivsete piltide korral energiakulu iga testpildi kohta (Joonis 12).



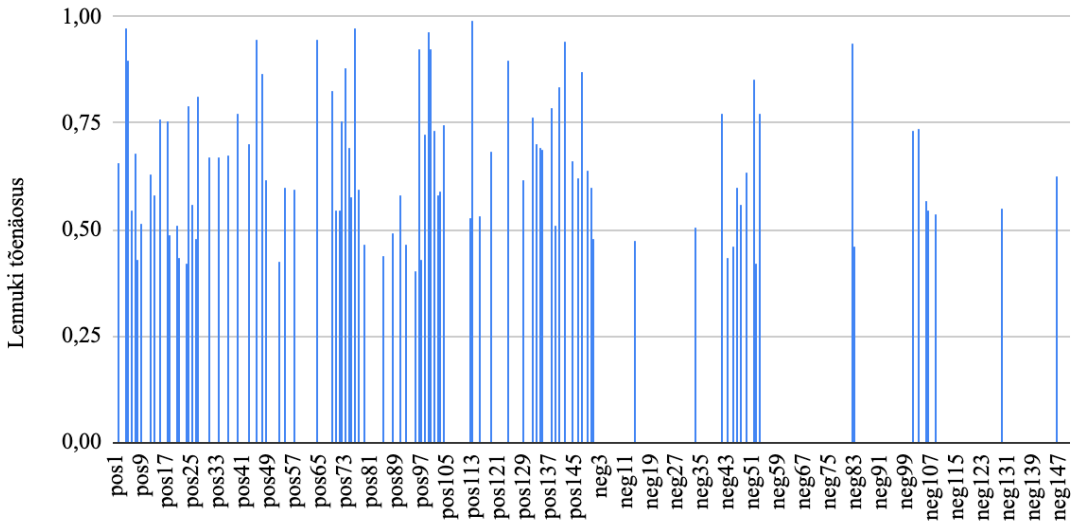
Joonis 12. ssd mobilenet_v1 2000 pos 1000 neg energiakulu tulemused

Mudeli *ssd_mobilenet_v1* 2000 positiivsete ja 1000 negatiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 13).



Joonis 13. ssd mobilenet_v1 2000 pos 1000 neg lennujaama tõenäosused

Mudeli *ssd_mobilenet_v1* 2000 positiivsete ja 1000 negatiivsete piltide korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 14).



Joonis 14. ssd mobilenet_v1 2000 pos 1000 neg lennuki tõenäosused

Tabel 9. Mudeli ssd_mobilenet_v1 2000 pos 1000 neg testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,40	0,99	0,67	0,49
pos lennujaam	0,40	0,98	0,80	0,79
pos energiakulu	0,020	0,026	0,020	
neg lennuk	0,42	0,94	0,60	0,14
neg lennujaam	0,41	0,97	0,81	0,83
neg energiakulu	0,020	0,025	0,020	

Mudeli treenimises kasutasime 2000 positiivset ja 1000 negatiivset pilti.

Graafikute järgi on nähtav, et andmekoguse kahekordistumine tõstis objektide tuvastamise tõenäosust võrreldes mudeliga, mille treenimisel kasutati näiteks 1000 pos ja 500 neg pilti.

TP tõenäosuse korral lennuki tuvastamine tõusis 4% kuni 49%, kuid lennujaama TP tõenäosuse väärtus langes umbes 20% võrra. FN tõenäosus lennuki tuvastamisel tõusis 8% kuni 14%, kui sama ajal lennujaama FN tõenäosuse väärtus langes 100% kuni 83%.

Lisaks mudeli ennustamise keskmiste tõenäosuste väärtused on muutunud kõrgemaks: lennuki puhul sai nüüd 67% nin lennujaama - 80%.

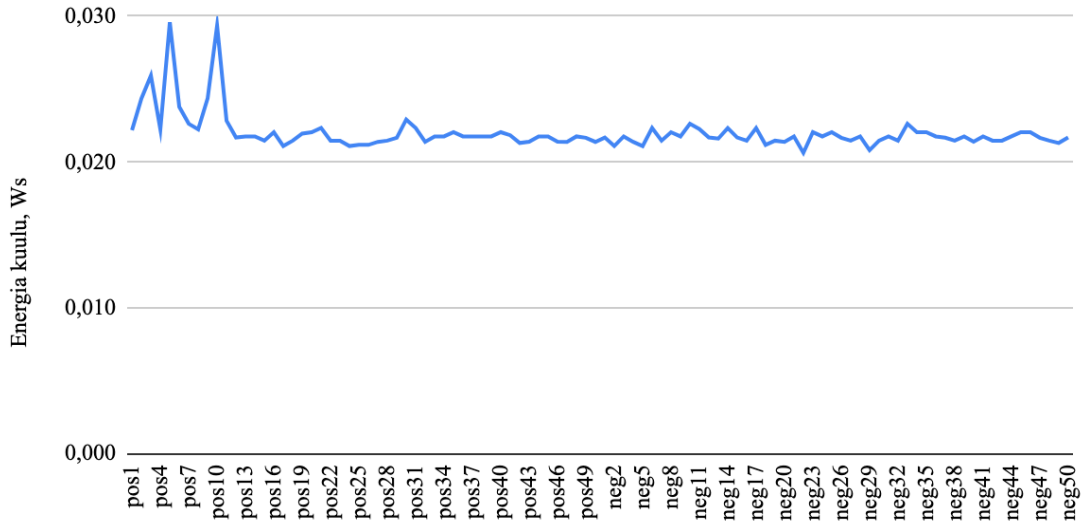
ssd_mobilenet_v1 (2000 pos 1000 neg) keskmine energiakulu langes 0,05 Ws kuni 0,02 Ws (Tabel 9).

6.2.2 ssd_mobilenet_v2 tulemused

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v2* treenitud mudeli tulemustest erinevate andmekogustega.

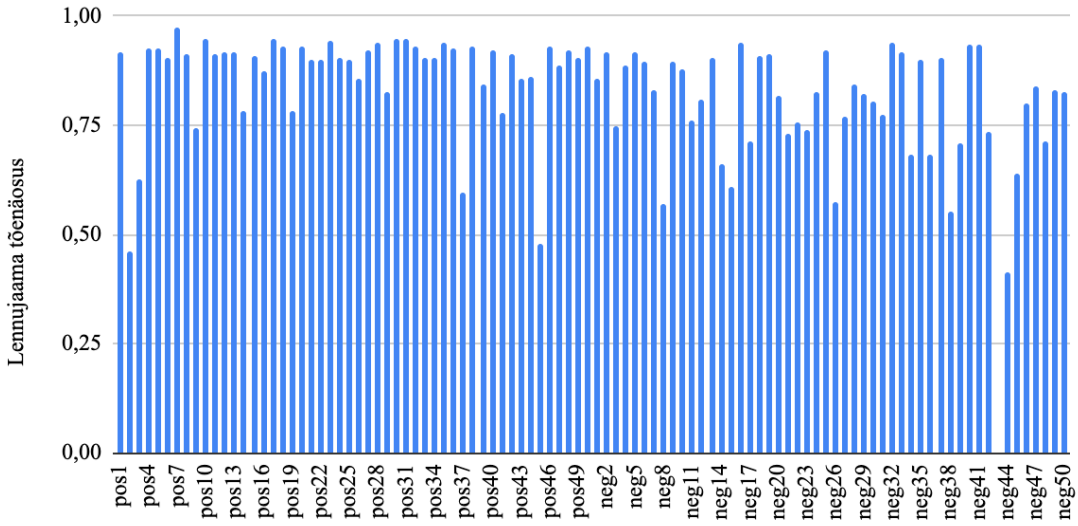
6.2.2.1 ssd_mobilenet_v2 1000 pos

Mudeli *ssd_mobilenet_v2* 1000 positiivsete piltide korral energiakulu iga testpildi kohta (Joonis 15).



Joonis 15. ssd mobilenet_v2 1000 pos energiakulu tulemused

Mudeli *ssd_mobilenet_v2* 1000 positiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 16).



Joonis 16. ssd mobilenet_v2 1000 pos lennujaama tõenäosused

Tabel 10. Mudeli ssd_mobilenet_v2 1000 pos testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0	0	0	0
pos lennujaam	0,46	0,97	0,87	1
pos energiakulu	0,021	0,030	0,022	
neg lennuk	0	0	0	0
neg lennujaam	0,42	0,94	0,79	0,98
neg energiakulu	0,021	0,023	0,022	

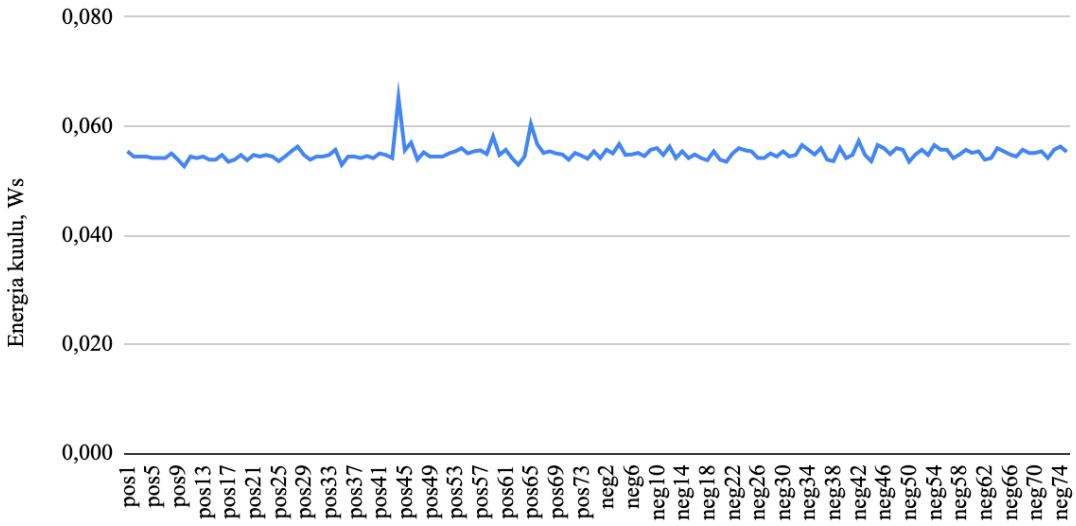
Mudeli treenimisel kasutasime 1000 positiivset pilti. Tulemuseks tuli välja, et mudel ei suutnud tuvastada ühtegi lennuki.

Nii TP, kui ka FN lennujaama tuvastamise tõenäosused olid päris kõrged - 100% ja 98% vastavalt, mis on samaaegselt nii hea, kui ka halb tulemus. *ssd_mobilenet_v2* (1000 pos) ennustamise keskmine tõenäosus positiivsete piltide peal on 87%.

Energiakulu statistika on hästi sarnane *ssd_mobilenet_v1* (1000 pos) mudeliga. Keskmine väärtus on 0,022 Ws.

6.2.2.2 ssd_mobilenet_v2 1000 pos 500 neg

Mudeli *ssd_mobilenet_v2* 1000 positiivsete ja 500 negatiivsete piltide korral energiakulu iga testpildi kohta (Joonis 17).



Joonis 17. ssd mobilenet_v2 1000 pos 500 neg energiakulu tulemused

Tabel 11. Mudeli ssd_mobilenet_v2 1000 pos 500 neg testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0	0	0	0
pos lennujaam	0	0	0	0
pos energiakulu	0,053	0,065	0,05	
neg lennuk	0	0	0	0
neg lennujaam	0	0	0	0
neg energiakulu	0,053	0,057	0,055	

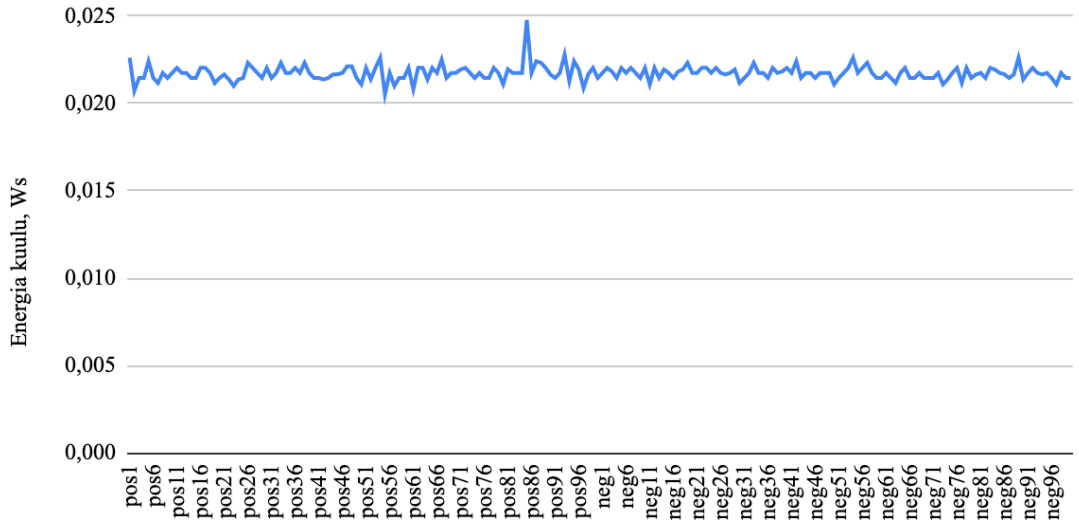
Mudelit treenitasime 1000 positiivsete ja 500 negatiivsete piltide peal.

Mudel ei saanud tuvastada lennukit ega lennujaama. Kõikidest mudelitest *ssd_mobilenet_v2* (1000 pos 500 neg) osutus kõige halvemaks.

Graafiku järgi näeme, et objekti tuvastamise energiakulu tõusis ning selle keskmine väärtus nüüd on 0,055 Ws.

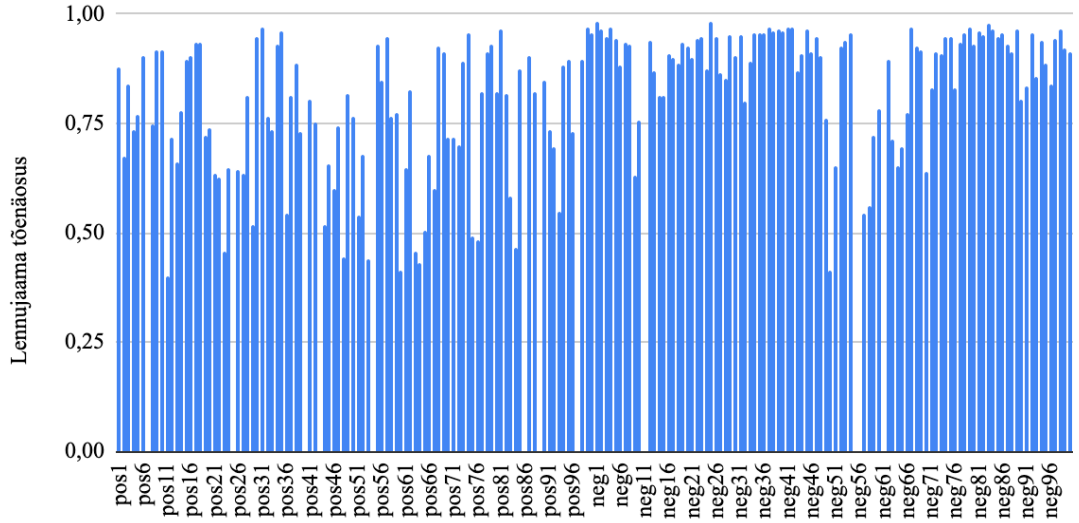
6.2.2.3 Mudeli ssd_mobilenet_v2 2000 pos testimise tulemused

Mudeli *ssd_mobilenet_v2* 2000 positiivsete piltide korral energiakulu iga testpildi kohta (Joonis 18).



Joonis 18. ssd mobilenet_v2 2000 pos energiakulu tulemused

Mudeli *ssd_mobilenet_v2* 2000 positiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 19).



Joonis 19. ssd mobilenet_v2 2000 pos lennujaama tõenäosused

Tabel 12. Mudeli ssd_mobilenet_v2 2000 pos testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0	0	0	0
pos lennujaam	0,40	0,97	0,75	0,93
pos energiakulu	0,021	0,025	0,022	
neg lennuk	0	0	0	0
neg lennujaam	0,41	0,98	0,88	0,96
neg energiakulu	0,021	0,023	0,022	

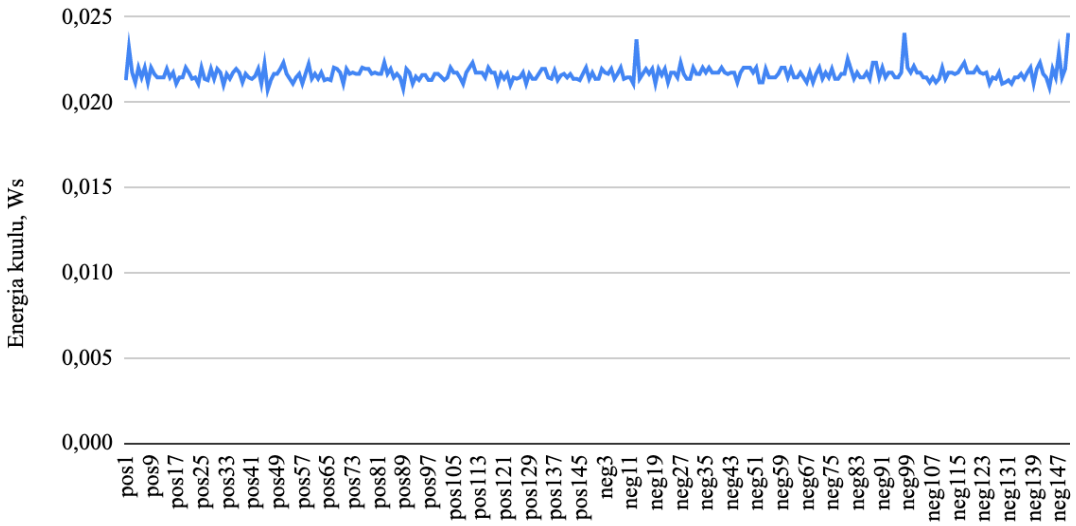
Mudeli oli treenitud 2000 positiivsete piltide peal.

ssd_mobilenet_v2 (2000 pos) sai ennustada ainult lennujaamad. Lennujaama tuvastamise TP tõenäosus oli 93% ning FN oli samamoodi kõrge - 96%. Ennustamise keskmine tõenäosus positiivse piltide peal oli 75%.

Energiakulu langes võrreldes eelmise mudeliga. Mudeli keskmine energiakulu oli 0,022 Ws (Tabel 12).

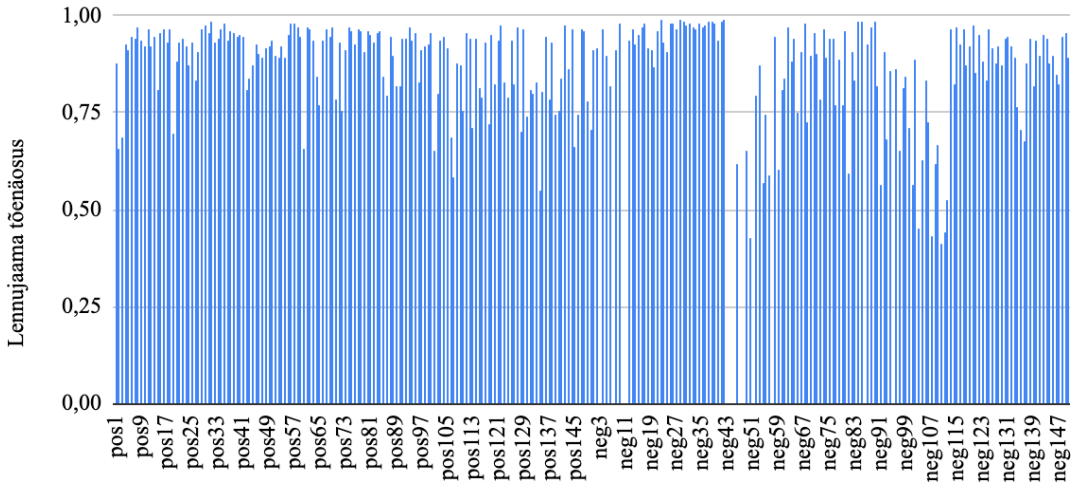
6.2.2.4 ssd_mobilenet_v2 2000 pos 1000 neg

Mudeli *ssd_mobilenet_v2* 2000 positiivsete ja 1000 negatiivsete piltide korral energiakulu iga testpildi kohta (Joonis 20).



Joonis 20. ssd mobilenet_v2 2000 pos 1000 neg energiakulu tulemused

Mudeli *ssd_mobilenet_v2* 2000 positiivsete ja 1000 negatiivsete piltide korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 21).



Joonis 21. ssd mobilenet_v2 2000 pos 1000 neg lennujaama tõenäosused

Tabel 13. Mudeli ssd_mobilenet_v2 2000 pos 1000 neg testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0	0	0	0
pos lennujaam	0,55	0,98	0,89	1
pos energiakulu	0,021	0,023	0,022	
neg lennuk	0	0	0	0
neg lennujaam	0,41	0,99	0,86	0,91
neg energiakulu	0,021	0,024	0,022	

Selle mudeli treenimisel kasutasime 2000 positiivse ja 1000 negatiivse pite.

Tulemuseks saime, et mudel ei saanud tuvastada lennukit. Aga lennujaamat sai ennustada TP tõenäosusega 100% ning iga positiivse piltide peal keskmise tõenäosusega 89%. Lisaks, lennujaama FN tõenäosus oli samamoodi kõrge - 91%.

Keskmine energiakulu jäi samaks võrreldes eelmise mudeliga ehk 0,022 Ws (Tabel 13).

6.3 Erinevate optimeerimise parameetritega mudelite testimine ja analüüs

Testimise eesmärk oli leida mudelite parameetrite väärtused, millega saime treenida mudelit parem ja mudel sai parem töödelda pildi. Kõik mudelite variandid olid treenitud 2000 positiivse ja 1000 negatiivse pilti peal.

Proovisime optimeerida järgmiste parameetrite abil:

- *optimizer* - peamine optimeerimiseks mõeldud arhitektuuri osa. SSD mudelite puhul on olemas kolm varianti: RMSprop, Adam ja Momentum. RMSprop oli vaikimisi eelnevates katsetes kasutatud, seega proovisime kasutada Adam ja Momentum optimeerimise meetodeid;
- *depth_multiplier* - võimaldab mudelil kiiremini õppida vähendades kihis kasutatud kanalite arvu. Vaikimisi väärtus oli 1.0, seega proovisime seda natuke vähendada kuni 0.75;
- *batch_size* - näitab, kui palju andmetükki analüüsimiseks mudel saab ühe sammu jooksul. Vaikimisi väärtus oli 24, ühe mudeli puhul tõstsime väärtust kuni 30 tükki ning teise mudeliga vähendasime väärtust võimaliku miinimumini ehk väärtuseni 1.

Lisaks proovisime *ssd_mobilenet_v2* mudeli arhitektuuri peal koostata võrdeliselt kõige optimeeritud variandi ning sellest saime kätte uue *ssdlite_mobilenet_v2* mudeli

Piltide kogus järgmiste katsete jaoks jäi kõige suuremaks ehk 2000 positiivset ja 1000 negatiivset pilti, sest nimelt nii suure andmekogusega mudelid näitasid enda parimad tulemused.

Tabel 14. Optimeeritud mudelite testimise kirjeldus

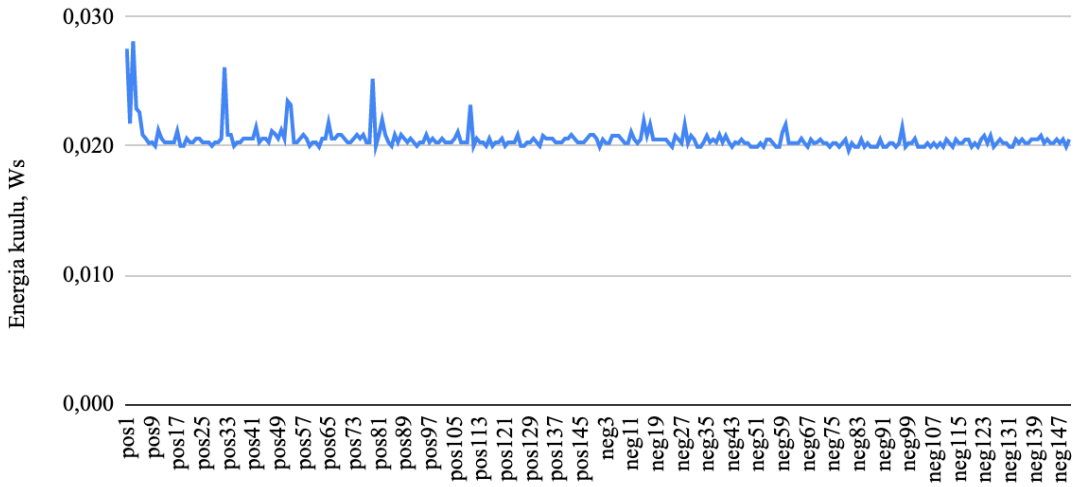
mudel	piltide kogus testimiseks
ssd_mobilenet_v1 2000 pos 1000 neg adam optimizer	150 pos ja 150 neg
ssd_mobilenet_v1 2000 pos 1000 neg momentum optimizer	150 pos ja 150 neg
ssd_mobilenet_v2 2000 pos 1000 neg adam optimizer	150 pos ja 150 neg
ssd_mobilenet_v2 2000 pos 1000 neg momentum optimizer	150 pos ja 150 neg
sdd_mobilenet_v1 2000 pos 1000 neg momentum depth mult 0.75	150 pos ja 150 neg
sdd_mobilenet_v2 2000 pos 1000 neg momentum depth mult 0.75	150 pos ja 150 neg
sdd_mobilenet_v2 2000 pos 1000 neg adam optimizer best values	150 pos ja 150 neg
ssd_mobilenet_v1 2000 pos 1000 neg adam batch 30	150 pos ja 150 neg
ssd_mobilenet_v2 2000 pos 1000 neg adam batch 1	150 pos ja 150 neg

6.3.1 ssd_mobilenet_v1 tulemused *optimizer* parameetritega

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v1* treenitud mudeli tulemustest erinevate optimeerimist võimaldavate meetoditega *optimizer* parameetri juures.

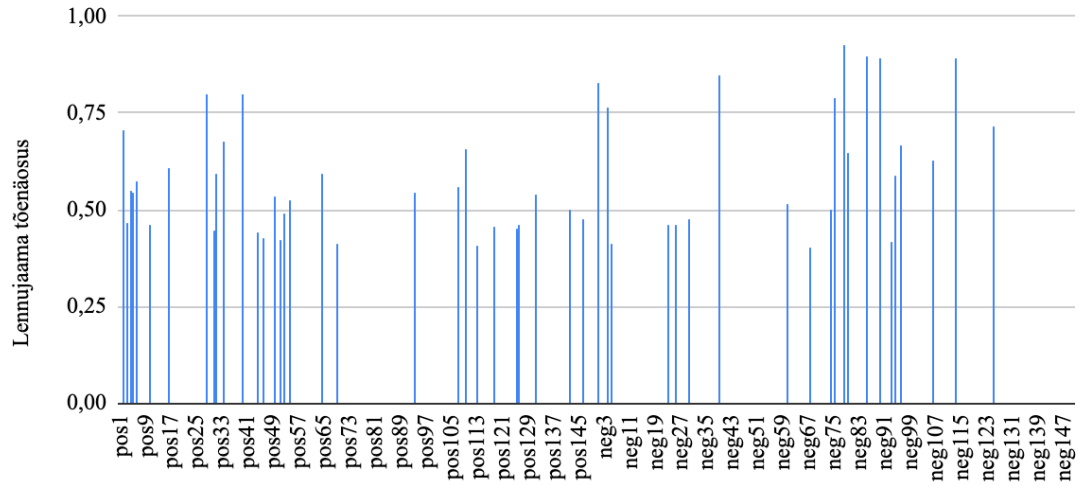
6.3.1.1 ssd_mobilenet_v1 adam optimizer

Mudeli *ssd_mobilenet_v1* Adam optimeerija korral energiakulu iga testpildi kohta (Joonis 22).



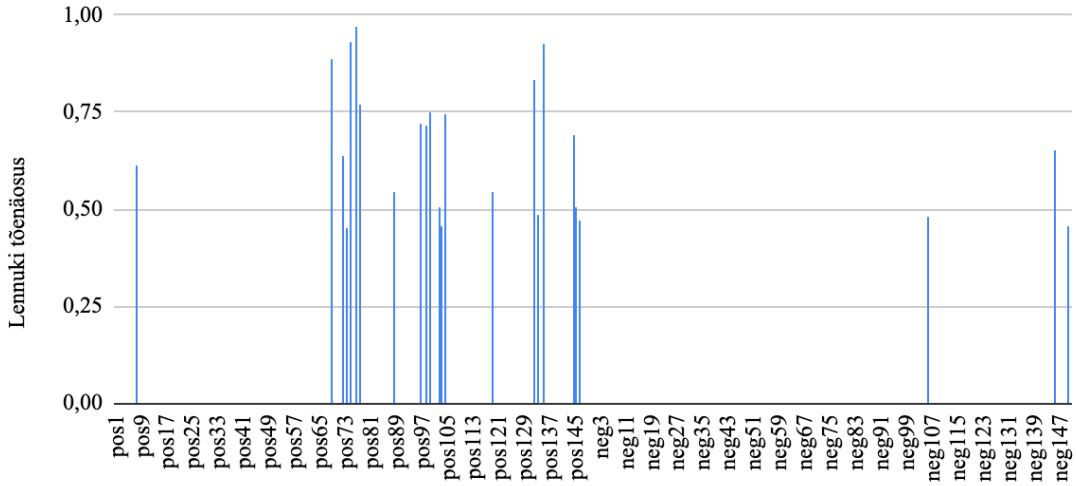
Joonis 22. ssd mobilenet_v1 adam optimizer energiakulu tulemused

Mudeli *ssd_mobilenet_v1* Adam optimeerija korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 23).



Joonis 23. ssd mobilenet_v1 adam optimizer lennujaama tõenäosused

Mudeli *ssd_mobilenet_v1* Adam optimeerija korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 24).



Joonis 24. ssd mobilenet_v1 adam optimizer lennuki tõenäosused

Tabel 15. Mudeli ssd_mobilenet_v1 adam optimizer testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,45	0,97	0,67	0,14
pos lennujaam	0,41	0,80	0,54	0,20
pos energiakulu	0,020	0,028	0,021	
neg lennuk	0,46	0,65	0,53	0,02
neg lennujaam	0,40	0,93	0,65	0,14
neg energiakulu	0,020	0,022	0,020	

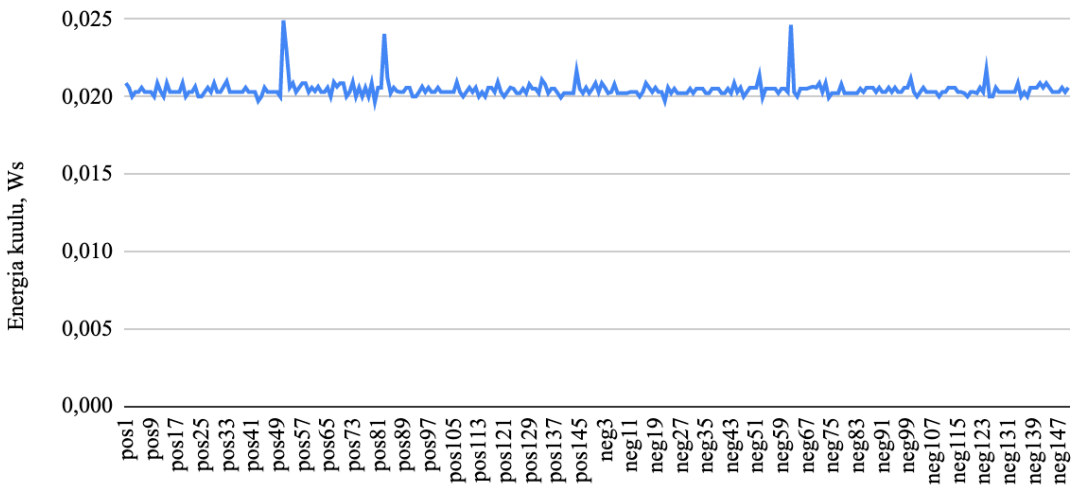
Adam optimeerija kasutamine langetas mudeli *ssd_mobilenet_v1* õige objekti tuvastamise tõenäosust. Kui varem positiivsete piltide kohta lennuki tõenäosus oli 49% ja lennujaama tõenäosus oli 79%, siis nüüd nad langesid 14% ja 20% peale.

Samas, langesid ka FN: lennuki tõenäosuse väärtus langes 14% kuni 2% ning lennujaama tõenäosuse väärtus 83% kuni 14%. FN tõenäosuse väärtuse vähendamine on juba positiivne tulemus.

Energiakulu jäi samaks ligi 0,02 Ws (Tabel 15).

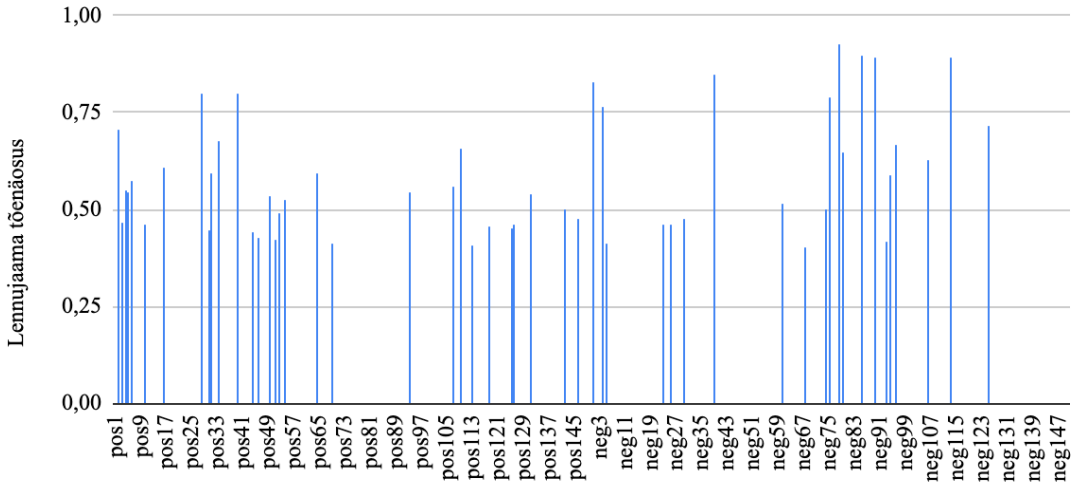
6.3.1.2 ssd_mobilenet_v1 momentum optimizer

Mudeli *ssd_mobilenet_v1* Momentum optimeerija korral energiakulu iga testpildi kohta (Joonis 25).



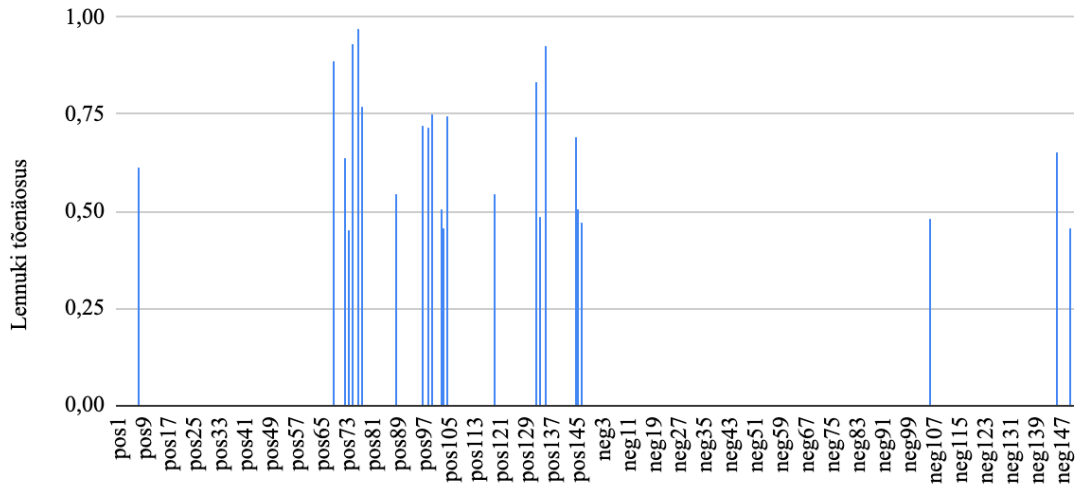
Joonis 25. ssd mobilenet_v1 momentum optimizer energiakulu tulemused

Mudeli *ssd_mobilenet_v1* Momentum optimeerija korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 26).



Joonis 26. ssd mobilenet_v1 momentum optimizer lennujaama tõenäosused

Mudeli *ssd_mobilenet_v1* Momentum optimeerija korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 27).



Joonis 27. ssd mobilenet_v1 momentum optimizer lennuki tõenäosused

Tabel 16. Mudeli ssd_mobilenet_v1 momentum optimizer testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,45	0,97	0,67	0,14
pos lennujaam	0,41	0,80	0,54	0,20
pos energiakulu	0,020	0,025	0,020	
neg lennuk	0,46	0,65	0,53	0,02
neg lennujaam	0,40	0,93	0,65	0,14
neg energiakulu	0,020	0,025	0,020	

Peale Momentum parameetri optimiseerimist mudeli ssd_mobilenet_v1 pildi tuvastamise tõenäosus langes võrreldes vaikimisi kasutatud RMSprop parameetritega. Kui varem TP tõenäosuse väärtus lennuki korral oli 49% ja lennujaama korral - 79%, siis nüüd need näitajad langesid vastavalt 14% ja 20% peale.

Samal ajal langes ka FN tõenäosuse väärtused: lennuki tõenäosuse väärtus muutis 14% kuni 2% ja lennujaama tõenäosuse väärtus langes 83% kuni 14%.

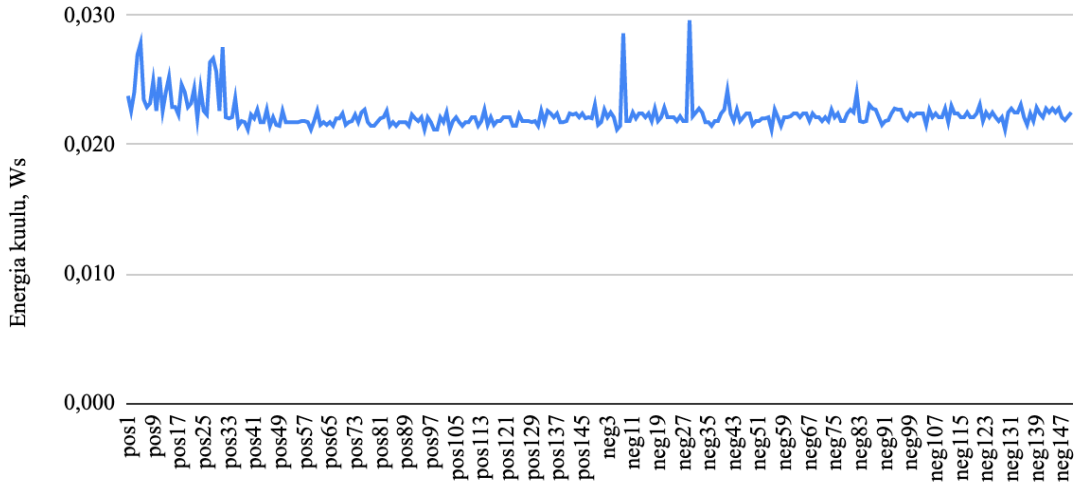
Energiakulu jäi samaks ligi 0,02 Ws.

6.3.2 ssd_mobilenet_v2 tulemused *optimiser* parameetritega

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v2* treenitud mudeli tulemustest erinevate optimeerimist võimaldavate meetoditega *optimizer* parameetri juures..

6.3.2.1 ssd_mobilenet_v2 adam optimizer testimise tulemused

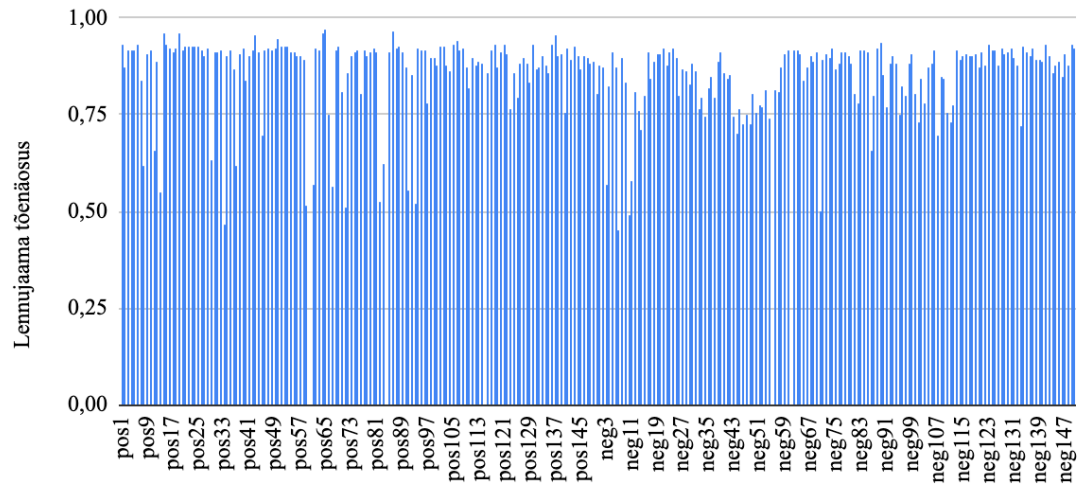
Mudeli *ssd_mobilenet_v2* Adam optimeerija korral energiakulu iga testpildi kohta (Joonis 28).



Joonis 28. ssd mobilenet_v2 adam optimizer energiakulu tulemused

Mudeli *ssd_mobilenet_v2* Adam optimeerija korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 29).

Joonis 29. ssd mobilenet_v2 adam optimizer lennujaama tõenäosused



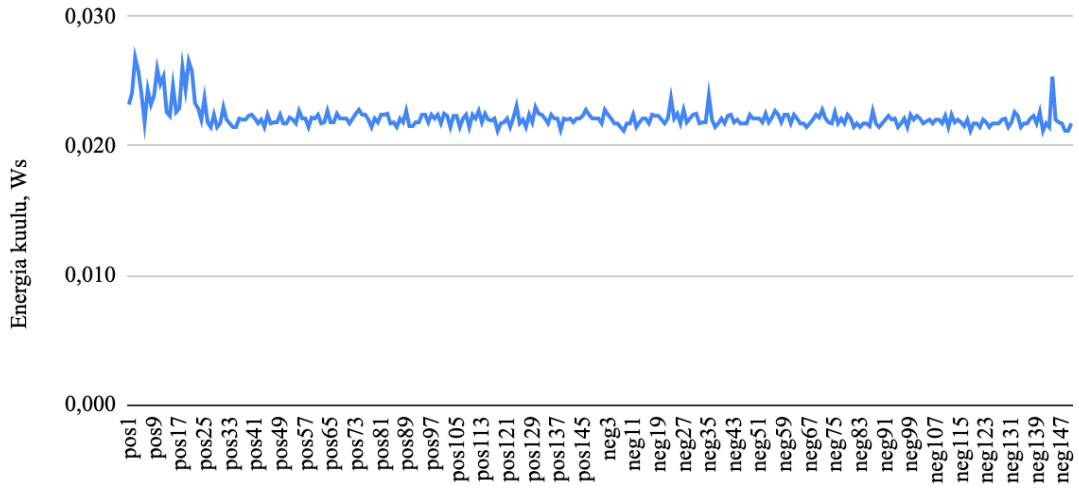
Tabel 17. Mudeli ssd_mobilenet_v2 adam optimizer testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,00	0,00	0,00	0,00
pos lennujaam	0,47	0,97	0,87	0,98
pos energiakulu	0,021	0,028	0,022	
neg lennuk	0,00	0,00	0,00	0,00
neg lennujaam	0,45	0,94	0,85	0,99
neg energiakulu	0,021	0,030	0,022	

Peale *optimizer* parameetri Adam peale muutmist mudeli *ssd_mobilenet_v2* õige objekti tuvastamise tõenäosus ei ole muutnud. Nii nagu varem mudel ei tuvastanud ühtegi lennukit. Lennujaama tuvastamise nii TP, kui ka FN tõenäosused jäid ligi 100% juures. Energiakulu jäi samaks - 0,02 Ws (Tabel 17).

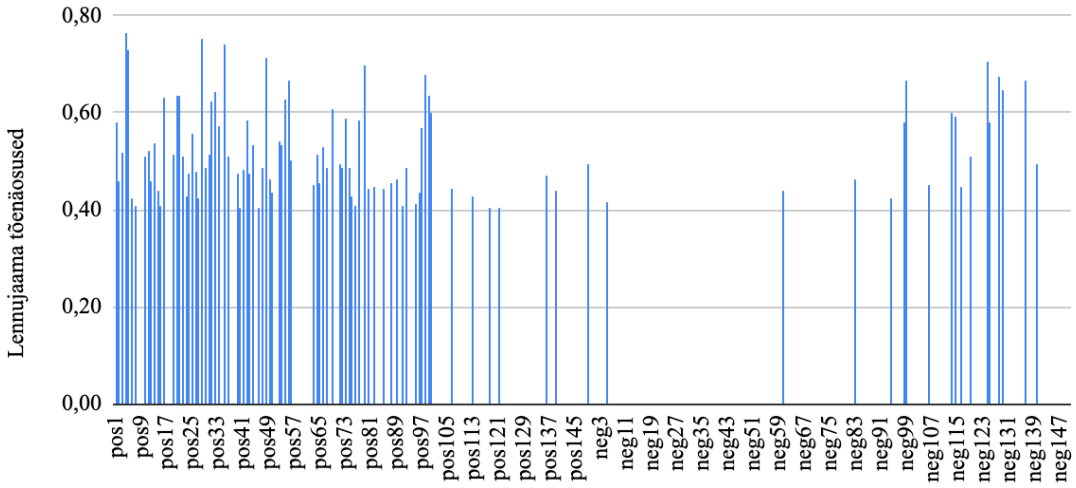
6.3.2.2 ssd_mobilenet_v2 momentum optimizer

Mudeli *ssd_mobilenet_v2* Momentum optimeerija korral energiakulu iga testpildi kohta (Joonis 30).'



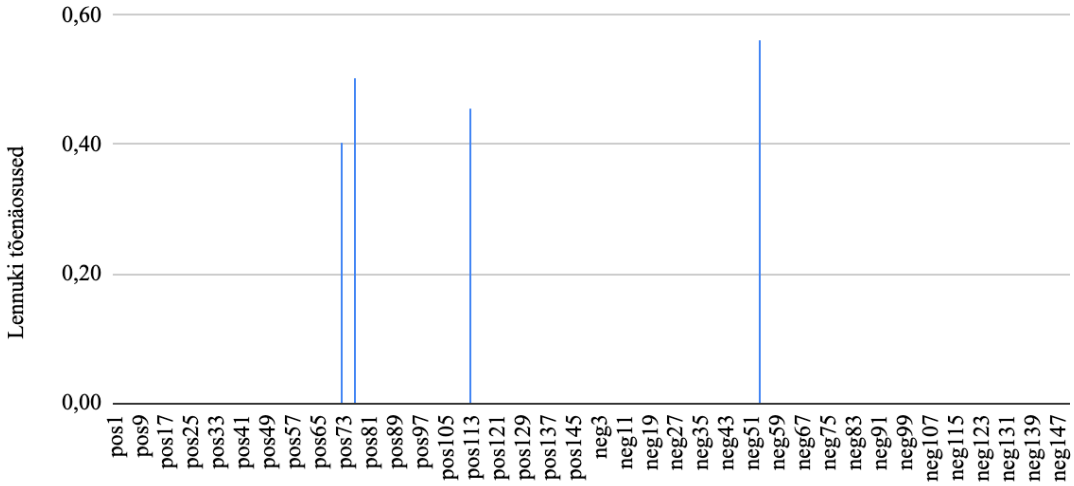
Joonis 30. ssd mobilenet_v2 momentum optimizer energiakulu tulemused

Mudeli *ssd_mobilenet_v2* Momentum optimeerija korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 31).



Joonis 31. ssd mobilenet_v2 momentum optimizer lennujaama tõenäosused

Mudeli *ssd_mobilenet_v2* Momentum optimeerija korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 32).



Joonis 32. ssd mobilenet_v2 momentum optimizer lennuki tõenäosused

Tabel 18. mudeli ssd_mobilenet_v2 momentum optimizer testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,40	0,50	0,45	0,02
pos lennujaam	0,40	0,76	0,52	0,55
pos energiakulu	0,021	0,027	0,022	
neg lennuk	0,56	0,56	0,56	0,01
neg lennujaam	0,42	0,71	0,55	0,11
neg energiakulu	0,021	0,025	0,022	

Momentum optimeerimise meetodiga mudeli *ssd_mobilenet_v2* objekti tuvastamise õige tuvastamise tõenäosus lennukite puhul natuke tõusis. Kui varem lennuki ennustamise TP ja FN tõenäosused olid 0% , siis nüüd need väärtused on tõusnud 2% ja 1% peale vastavalt.

Kahjuks lennujaamade TP tõenäosus langes 9% võrra kuni 91% väärtuseni, aga sama ajal FN väärtuse langemine tähendas paremat tulemust 55% kuni 11%.

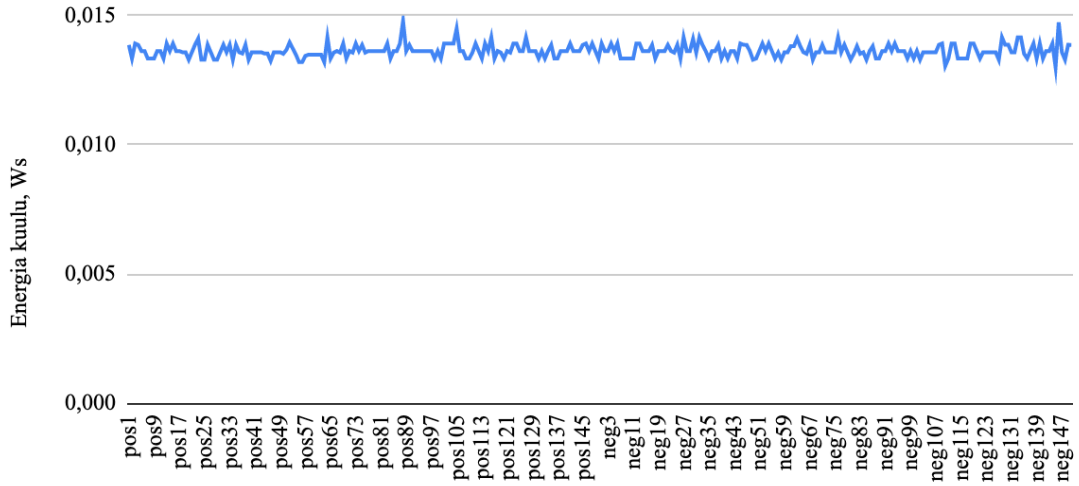
Energiakulu ei muutnud - 0,02 Ws (Tabel 18).

6.3.3 ssd_mobilenet_v1 tulemused *depth_multiplier* uue väärtusega

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v1* treenitud mudeli tulemustest uue *depth_multiplier* väärtusega - 0,75. *optimizer* parameetri väärtuseks on jäänud Momentum optimeerija.

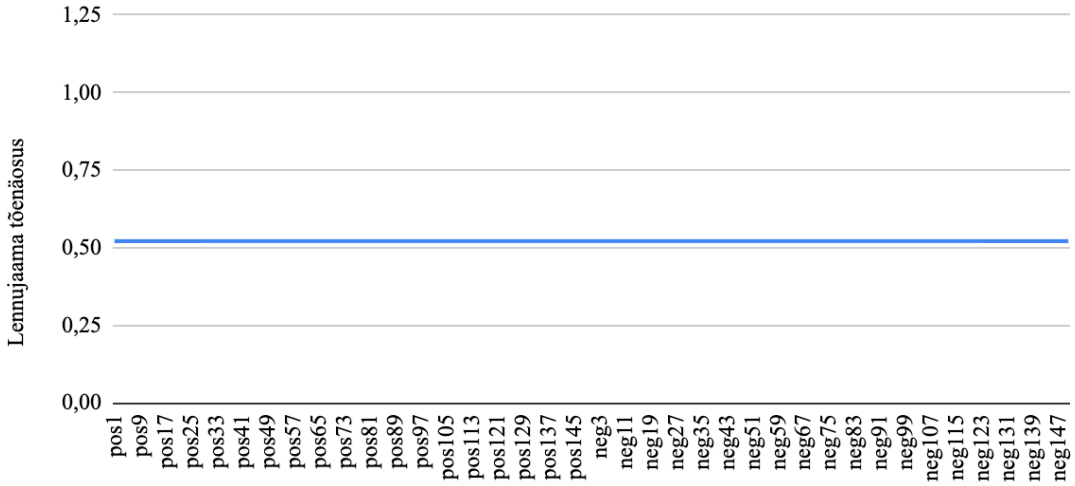
6.3.3.1 ssd_mobilenet_v1 momentum optimizer depth mult 0.75

Mudeli *ssd_mobilenet_v1* Momentum optimeerija ja sügavuse kordaja 0,75 korral energiakulu iga testpildi kohta (Joonis 33).



Joonis 33. ssd mobilenet_v1 momentum optimizer depth mult 0.75 energiakulu tulemused

Mudeli *ssd_mobilenet_v1* Momentum optimeerija ja sügavuse kordaja 0,75 korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 34).



Joonis 34. ssd mobilenet_v1 momentum optimizer depth mult 0.75 lennujaama tõenäosused

Tabel 19. mudeli ssd_mobilenet_v1 momentum optimizer depth mult 0.75 testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,00	0,00	0,00	0,00
pos lennujaam	0,52	0,52	0,52	1,00
pos energiakulu	0,013	0,015	0,014	
neg lennuk	0,00	0,00	0,00	0,00
neg lennujaam	0,52	0,52	0,52	1,00
neg energiakulu	0,013	0,015	0,014	

Koos Momentum optimeerimise meetodiga ja sügavuse kordajaga väärtusega 0,75 *ssd_mobilenet_v1* lennuki tuvastamine ei õnnestunud. Lennujaamade nii TP, kui ka FN tõenäosuste väärtused suurenesid kuni 100%, mis on vaieldamata halb tulemus, sest FN kindlasti ei tohi nii kõrge olla. Lennujaama keskmine tuvastamise tõenäosus pildil oli 52%.

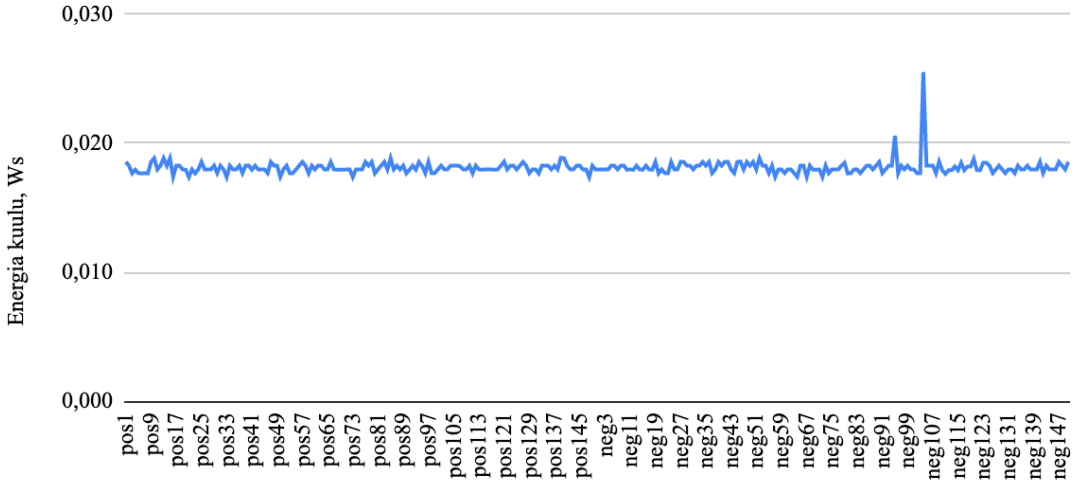
Võrreldes mudeliga, milles muudetud osaks oli ainult Moment optimeerija kasutamine, praegusel korral langes energiakulu. Nüüd keskmine energiakulu oli 0,014 Ws (Tabel 19).

6.3.2 ssd_mobilenet_v2 tulemused *depth_multiplier* uue väärtusega

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v2* treenitud mudeli tulemustest uue *depth_multiplier* väärtusega - 0,75. *optimizer* parameetri väärtuseks on jäänud Momentum optimeerija.

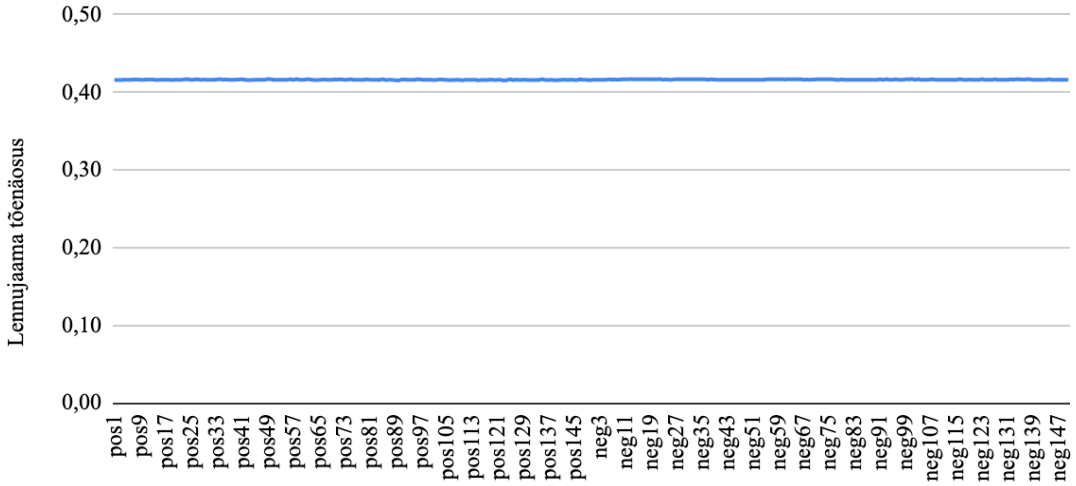
6.3.2.1 ssd_mobilenet_v2 momentum optimizer depth mult 0.75

Mudeli *ssd_mobilenet_v2* Momentum optimeerija ja sügavuse kordaja 0,75 korral energiakulu iga testpildi kohta (Joonis 35).



Joonis 35. ssd mobilenet_v2 momentum optimizer depth mult 0.75 energiakulu tulemused

Mudeli *ssd_mobilenet_v2* Momentum optimeerija ja sügavuse kordaja 0,75 korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 36).



Joonis 36. ssd mobilenet_v2 momentum optimizer depth mult 0.75 lennujaama tõenäosused

Tabel 20. mudeli ssd_mobilenet_v2 momentum optimizer depth mult 0.75 testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,00	0,00	0,00	0,00
pos lennujaam	0,42	0,42	0,42	1,00
pos energiakulu	0,017	0,019	0,018	
neg lennuk	0,00	0,00	0,00	0,00
neg lennujaam	0,42	0,42	0,42	1,00
neg energiakulu	0,017	0,026	0,018	

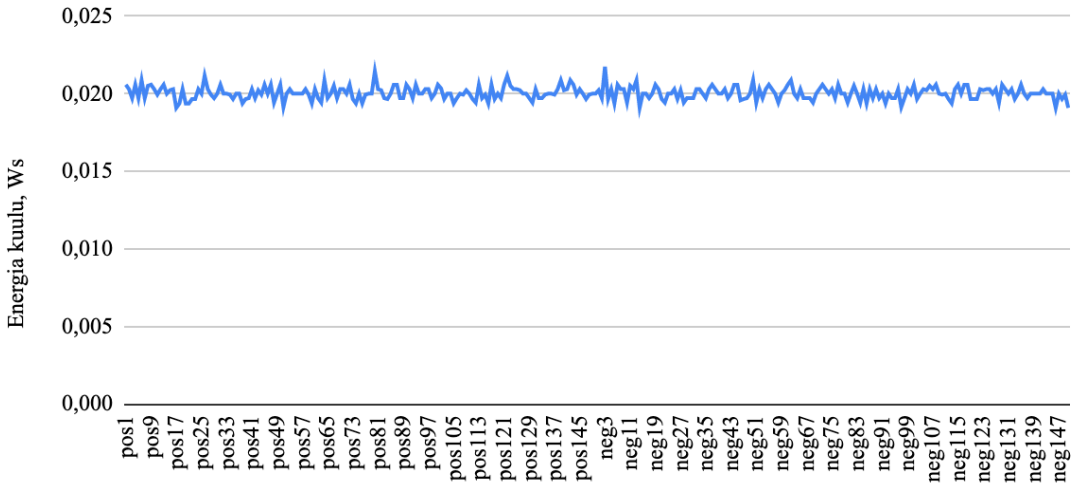
Koos Momentum optimeerimise meetodiga ja sügavuse kordajaga väärtusega 0,75 *ssd_mobilenet_v2* mudeli lennuki objekti tuvastamine ei õnnestunud. Lennujaama TP ja FN väärtused suurenesid kuni 100%, mis ainult TP väärtuse tõusu korral oleks väga positiivne, aga nii suure FN väärtuse tõttu ei saa saadud tulemust nimetada heaks. Lennujaama keskmine tuvastamise tõenäosus oli 42%.

Võrreldes katsega, milles kasutasime ainult Momentum optimeerijat ilma sügavuse kordajat muutmata, energiakulu vähenes.. Nüüd keskmine energiakulu oli 0,018 Ws (Tabel 20).

6.3.4 ssdlite_mobilenet_v2 uue mudeli tulemused

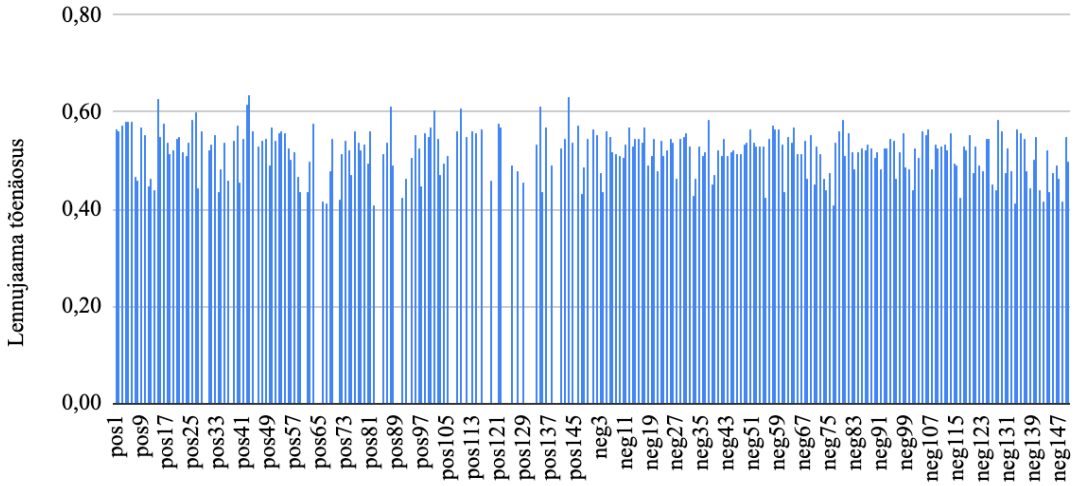
Edasi teeme ülevaadet ja analüüsi *ssdlite_mobilenet_v2 uue* treenitud mudeli tulemustest. See mudel erineb teistest arhitektuuri osadest, on mõeldud olema kergem ja kiirem käivitamise mõttes. Kasutasime selles mudelis Adam optimeerimise meetodi kõige paremate LR väärtustega treenimise jooksul (0,0005, 0,001, 0,00146), määrates need väärtused teatud sammude arvu läbimisel.

Mudeli *ssdlite_mobilenet_v2* energiakulu iga testpildi kohta (Joonis 37).



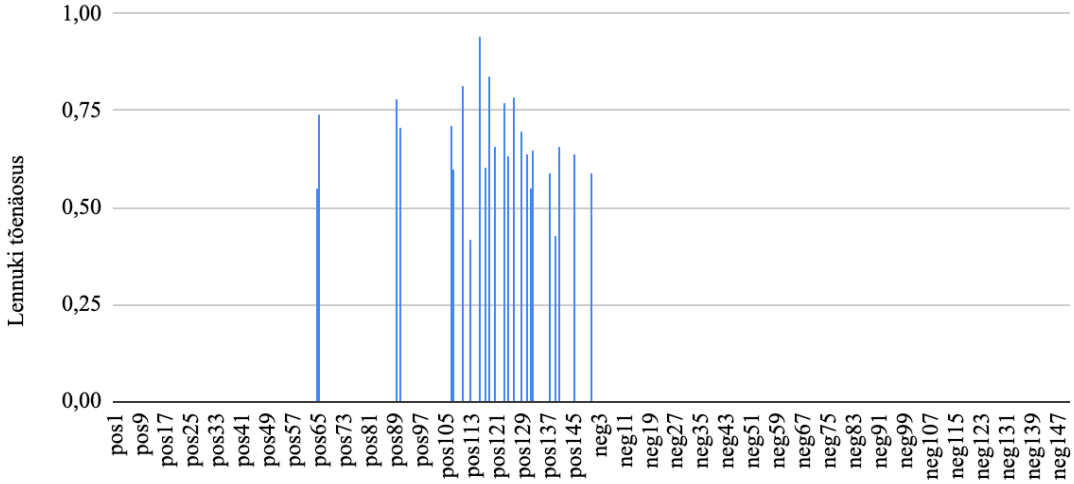
Joonis 37. ssdlite_mobilenet_v2 energiakulu tulemused

Mudeli *ssdlite_mobilenet_v2* lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 38).



Joonis 38. ssdlite_mobilenet_v2 lennujaama tõenäosused

Mudeli *ssdlite_mobilenet_v2* lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 39).



Joonis 39. ssdlite_mobilenet_v2 lennuki tõenäosused

Tabel 21. mudeli ssdlite_mobilenet_v2 testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,42	0,94	0,66	0,16
pos lennujaam	0,41	0,64	0,52	0,79
pos energiakulu	0,019	0,021	0,020	
neg lennuk	0,00	0,00	0,00	0,00
neg lennujaam	0,41	0,58	0,51	1,00
neg energiakulu	0,019	0,022	0,020	

Viimases mudelis kasutasime parimad Adam optimeerimise parameetri LR väärtused terve treenimise protsessi vältel. Tulemusena saime, et lennuki ja lennujaama TP ennustamine tõusis ning nüüd on 16% ja 79% vastavalt.

Kahjuks FN lennujaama ennustamine jai sama suureks ehk 100%, kuid lennuki FN väärtus langes kuni 0%, mida võib nimetada heaks tulemuseks.

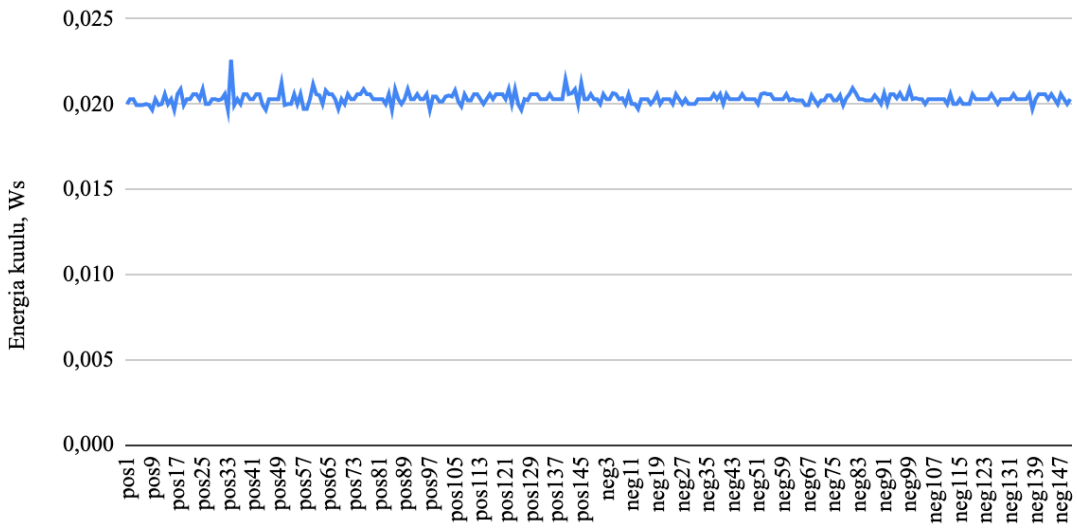
Energiakulu väärtust mudeli sisu ei mõjutanud ning see jäi 0,02 Ws juures (Tabel 21).

6.3.5 ssd_mobilenet_v1 tulemused *batch_size* uue väärtusega

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v1* treenitud mudeli tulemustest uue *batch_size* väärtusega - 30 tükki ühe sammu peale. Lisaks *optimizer* parameetri väärtusena kasutasime Adam optimeerijat.

6.3.5.1 ssd_mobilenet_v1 batch_size 30

Mudeli *ssd_mobilenet_v1 batch_size* väärtuse 30 korral energiakulu iga testpildi kohta (Joonis 40).



Joonis 40. ssd mobilenet_v1 adam batch 30 energiakulu tulemused pildi kohta

Tabel 22. Mudeli ssd_mobilenet_v1 adam batch 30 testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,00	0,00	0,00	0,00
pos lennujaam	0,00	0,00	0,00	0,00
pos energiakulu	0,020	0,023	0,020	
neg lennuk	0,00	0,00	0,00	0,00
neg lennujaam	0,00	0,00	0,00	0,00
neg energiakulu	0,020	0,021	0,020	

ssd_mobilenet_v1 (batch_size 30 ja Adam optimeerija) mudel osutus kõige ebaõnnestunuks ning ei saanud tuvastada lennukit ega lennujaamat.

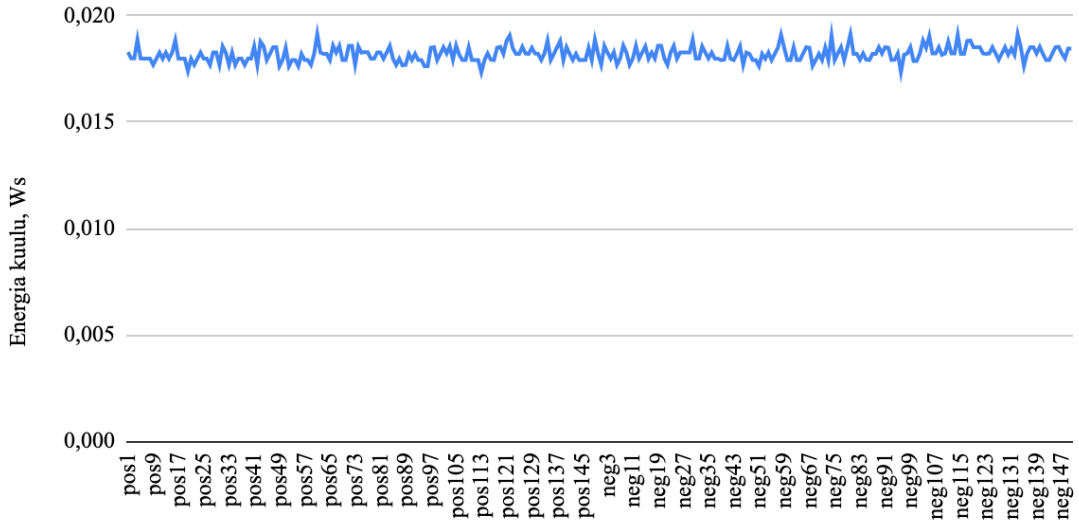
Energiakulu ei muutnud (Tabel 22).

6.3.6 ssd_mobilenet_v2 tulemused *batch_size* uue väärtusega

Edasi teeme ülevaadet ja analüüsi *ssd_mobilenet_v1* treenitud mudeli tulemustest uue *batch_size* minimaalse võimaliku väärtusega - 1 tükk ühe sammu peale. Lisaks *optimizer* parameetri väärtusena kasutasime Adam optimeerijat.

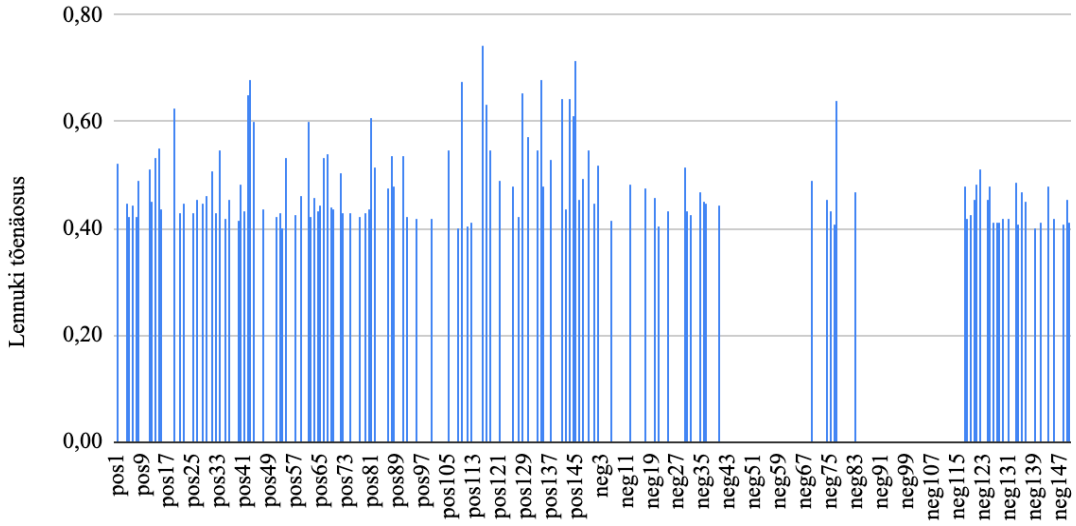
6.3.6.1 ssd_mobilenet_v2 batch_size 1

Mudeli *ssd_mobilenet_v2 batch_size* väärtuse 1 korral energiakulu iga testpildi kohta (Joonis 41).



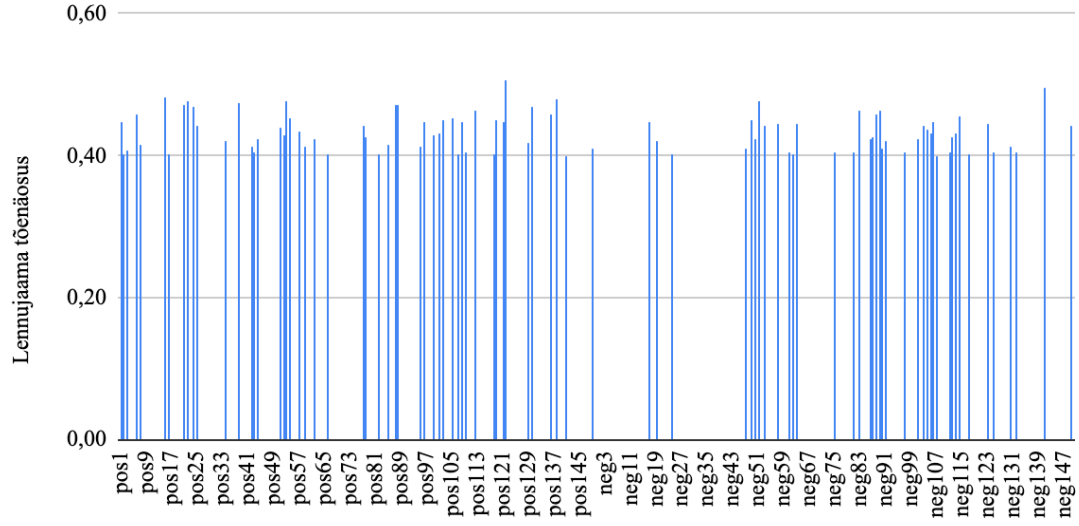
Joonis 41. ssd mobilenet_v2 adam batch 1 energiakulu tulemused pildi kohta

Mudeli *ssd_mobilenet_v2 batch_size* väärtuse 1 korral lennujaama tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 42).



Joonis 42. ssd mobilenet_v2 adam batch 1 lennujaama tõenäosused pildi kohta

Mudeli *ssd_mobilenet_v2 batch_size* väärtuse 1 korral lennuki tuvastamise tõenäosuse väärtused iga testpildi kohta (Joonis 43).



Joonis 43. ssd mobilenet_v2 adam batch 1 lennuki tõenäosused pildi kohta

Tabel 23. Mudeli ssd_mobilenet_v2 adam batch 1 testimise kokkuvõte

Mudeli analüüs	min	max	keskmine	pildi tuvastamise tõenäosus
pos lennuk	0,40	0,74	0,50	0,57
pos lennujaam	0,40	0,51	0,44	0,33
pos energiakulu	0,017	0,019	0,018	
neg lennuk	0,40	0,64	0,45	0,30
neg lennujaam	0,40	0,50	0,43	0,26
neg energiakulu	0,017	0,019	0,018	

Võrreldes *ssd_mobilenet_v2* Adam optimeerijat vaikimisi *batch_size* väärtusega 24 kasutamise variandiga lennuki TP tõenäosuse väärtus kasvas kuni 57%. Samal ajal vähenes lennujaama TP tõenäosus 33 % peale.

lennujaama FN tõenäosus näitas positiivse langu - 99% kuni 26%. Lennuki puhul kasvas kahjuks mitte ainult TP tõenäosus, vaid ka FN tõenäosuse väärtus nullist kuni 30%.

Keskmine energiakulu vähenes kuni 0,018 Ws.

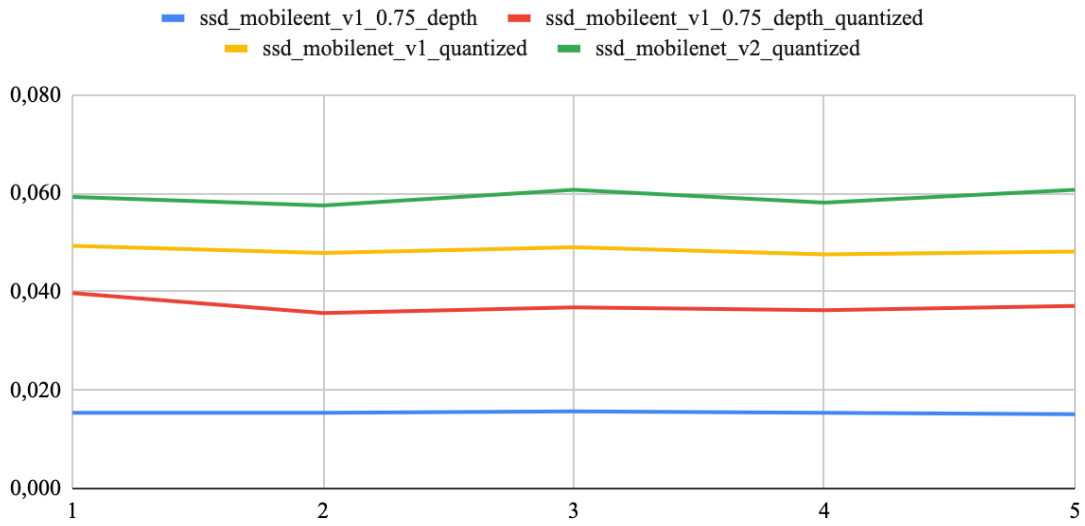
6.8. Teise mudelite testimine

Testimise eesmärk oli teha katsed teiste mudelitega, mida me ise ei treeninud ehk nende mudelitega, mis sobivad arhitektuuri poolest Coral USB akseleraatori peal jooksutamiseks, vaid mis ei sobinud meile nendega opereerimiseks. See oli tellija soov, et uurida rohkem erinevaid energiakulu variante. Kuna tuvastamise täpsus sellel korral ei olnud oluline, siis testimine toimus ainult viie pildi peal, põhirõhk oli energiakulu mõõtmisele.

Kõik viis pilti olid peaaegu sama suurusega ja samas pildi formaadis *.jpg*.

Tabel 24. Mitte treenitud mudelite testimise kirjeldus

mudel	piltide kogus testimiseks
ssd_mobilenet_v1_0.75_depth	5
ssd_mobilenet_v1_0.75_depth_quantized	5
ssd_mobilenet_v1_quantized	5
ssd_mobilenet_v2_quantized	5



Joonis 44. Energiakulu tulemused nelja mudeli jaoks

Võib järeldada, et kõige väiksema energiakuluga oli *ssd_mobilenet_v1_0.75_depth* mudel, mille keskmine väärtus oli 0,015 Ws.

Kõige kulukam energiakulu mõttes oli mudel *ssd_mobilenet_v2_quantized*, mille keskmine väärtus oli 0,059 Ws.

Teised mudelid, võrreldes meie omadega, olid kõrgema energiakuluga ning nende keskmised väärtused olid vastavalt 0,037 Ws ja 0,048 Ws (Joonis 44).

7 Testimise tulemuste kokkuvõte

7.1 Erinevate andmekogustega testimise tulemuse kokkuvõte

7.1.1 1000 positiivset pilti

ssd_mobilenet_v1 õige klassi ennustamine on ligi 50% ning TP tõenäosused kahe klassi puhul on väikesed: mõlemad - 4%. Samal ajal TN lennujaama puhul on 98% ja lennuki puhul on 74%. Keskmine energiakulu on 0,022 Ws, mis ei ole väga suur.

ssd_mobile_net_v2 sai tuvastada ainult lennujaamad: lennuki TP on 0%, aga TN on 100%, lennujaama TP on 100% ja FN tõenäosus on 100%. Sellest võib järeldada, et 1000 positiivset pilti ei ole piisav usaldusväärseks töötlemiseks *ssd_mobile_net_v2* mudeli korral. Keskmine energiakulu on sama nagu *ssd_mobilenet_v1* mudeli korral ehk 0,022 Ws.

7.1.2 1000 positiivset ja 500 negatiivset pilti

Mudeli *ssd_mobilenet_v1* õige klassi ennustamine tõusis, aga FN tõenäosus tõusis lennujaama puhul kuni 100%.

ssd_mobile_net_v2 mudel ei ennustanud lennuki ega lennujaama üldse, seega selline treenimise piltide kogus ei sobi ka. Energiakulu selle andmekogusega on kõige kõrgem: *ssd_mobilenet_v1* keskmine energiakulu on 0,052 Ws ning *ssd_mobile_net_v2* on 0,055 Ws.

7.1.3 2000 positiivset pilti

Mõlemad versioonid said tuvastada ainult lennujaamad, ning mudelite FN tõenäosus on 100% juures. *ssd_mobilenet_v1* ja *ssd_mobile_net_v2* versioonid ei olnud piisavalt edukad selle andmekoguse variandiga.

7.1.4 2000 positiivset ja 1000 negatiivset pilti

ssd_mobilenet_v1 sai ennustada rohkem lennukeid positiivse piltide peal keskmine tõenäosusega 67%, kuid maksimaalne väärtus oli 99%, mis on kõige parem tulemus antud katsete hulgas.

Lennuki FN tõenäosus on väike - 14%. Kahjuks lennujaama FN tõenäosus on 86%, mis näitab, et mudel veel ei oska hästi eristada lennujaamasid.

ssd_mobilenet_v1 keskmine energiakulu on 0,020 Ws.

ssd_mobilenet_v2 versiooni tulemused on täpselt samad, nagu 2000 positiivsete piltide korral.

7.1.5 Järeldused

Kokkuvõtteks saame öelda, et kõige parema tulemuse on näidanud on viimane andmekoguse variant - 2000 positiivset ja 1000 negatiivset pilti *ssd_mobilenet_v1* mudeli peal. Energia kulu testimise käigus oli 0,02 Ws juures, aga mõnikord tekkisid vooluhüpped kuni 0,3 Ws. Energiakulu mõõtmises sai selgeks, et väärtus sõltub peamiselt pildi suurusest ja mudeli konfiguratsioonist. Seoses sellega, et testpildid olid peaaegu sama suurusega, energiakulu tulemused olid hästi sarnased ja stabiilsed.

7.2 Teise testimise tulemuse kokkuvõte

7.2.1 Adam optimeerija kasutamine

Võrreldes *ssd_mobilenet_v1 adam optimizer* mudelit *ssd_mobilenet_v1* (2000 pos 1000 neg) mudeliga, objekti tuvastamise TP tõenäosused langesid: lennuki puhul nüüd on 14%, lennujaama - 20%. Samal ajal FN tõenäosus langes ka, seega mudel oskab juba paremini objekte eristada: nüüd lennuki FN on 2%, lennujaama FN on 14%. Klassi ennustamise tõenäosused ei muutunud ja energiakulu jäi samaks.

ssd_mobilenet_v2 puhul Adam optimeerija rakendamine ei aidanud ning tulemused jäid samaks.

7.2.2 Momentum optimeerija kasutamine

Momentum optimeerimise meetodiga *ssd_mobilenet_v1* tulemused jäid Adam parameetri kasutamise korraga samad.

ssd_mobilenet_v2 mudeli TP tõenäosused lennuki ja lennujaama puhul on vastavalt 2% ja 55%. Klassi enustamise tõenäosused on 45% ja 55%. FN tõenäosused langesid: lennujaama korral on 11%, lennuki korral on 1%. Keskmine energiakulu jäi samaks.

Analüüs näitas, et Momentum optimeerija rakendamine aitas *ssd_mpbilenet_v2* mudelil paremini ennustada klasse ja eristada objekte negatiivsete piltide peal.

7.2.3 Depth_multiplier väärtuse vähendamine

Sügavuse kordaja väärtuse väiksemaks tegemine ei aidanud, sest mõlemad mudelid said ennustada ainult lennujaamasid, kuid samal ajal lennujaama FN tõenäosus on 100%, mis on päris halb tulemus. Lisaks, *ssd_mobilenet_v1* mudeli klassi tuvastamise tõenäosus on stabiilselt 52%, *ssd_mobilenet_v2* puhul stabiilselt 42%.

Mõlemate mudelite käivitamise energiakulu langes ja nüüd see on 0,02 Ws väärtusest madalam.

7.2.4 Adam optimeerija LR parimad väärtused

ssdlite_mobilenet_v2 mudeli kasutamise korral lennuki ja lennujaama TP tõenäosused on vastavalt 16% ja 76%. FN tõenäosused on 0% ja 100%, mis ei ole eriti hea tulemus.

Energiakulu jäi samaks võrreldes *ssd_mobilenet_v1* ja *ssd_mobilenet_v2* mudelite väärtustega.

7.2.5 Batch_size väärtuse muutmine

ssd_mobilenet_v1 puhul *batch_size* parameetri väärtus suurendati ning see oli 30. Selle parameetri väärtuse suurendamine ei aidanud tulemust parandada, aga vastupidi tegi mudeli tööd halvemaks: mudel ei suutnud tuvastada lennukit ega lennujaama. Energiakulu ei muutnud võrreldest teiste katsetega.

ssd_mobilenet_v2 puhul *batch_size* parameetri väärtus vähendati võimaliku miinimumini ning see oli 1. Võrreldes *ssd_mobielenet_v2* (2000 pos ja 1000 neg) mudeli katsega, vähendatud *batch_size* väärtusega mudel sai tuvastada lennukeid õigesti, aga samal ajal lennujaama tuvastamise tõenäosus langes. Lennuki FN tõenäosus tõusis.

Keskmine energiakulu langes ja nüüd on 0,018 Ws, mis energia kasutamise osas hea tulemus.

7.2.5 Järeldused

Kokkuvõtteks saame öelda, et Momentum optimeerimise meetodi ja *batch_size* väärtuse vähendamine aitas mudelitel *ssd_mobilenet_v2* paremini töödelda pilte ning ei rikkunud energiakulu tulemused. Kahjuks, Adam optimeerija kasutamine ja *depth_multiplier* väärtuse vähendamine ei aidanud parandada tulemusi.

Energia mõõtmise küljest *depth_multiplier* väärtuse väiksemaks ja *batch_size* väärtuse minimaalseks tegemine aitas vähendada kulu.

ssd_mobilenet_v1 korral Momentum ja Adam optimeerijate rakendamine aitas oluliselt vähendada FN tõenäosuse väärtust - see on hea tulemus, sest TP tõenäosus sõltub rohkem treenimise protsessi pikkusest ja piltide arvust, mida saab alati suurendada.

7.3 Kolmanda testimise tulemuste kokkuvõte

Praeguseks hetkeks meil õnnestunud konverteerida *.tflite* formaadiks ainult neli erinevat mudelit kõikidest ülejäänutest. Ebaõnnestunute mudelite korral tuli välja, et nimelt nende struktuurist ei ole võimalik tekitada soovitud formaati.

Coral'i ametliku veebilehe poolt edastatavad mudelit samuti ei sobinud, sest nende sees leitav klasside fail ei vastanud mudelite tingimustele.

Lõpuks saime testida neli mudelit:

1. ssd_mobilenet_v1 0.75 depth
2. ssd_mobilenet_v1 0.75 depth quantized
3. ssd_mobilenet_v1 quantized
4. ssd_mobilenet_v2 quantized

Võrreldes mudelid meie omadega, *ssd_mobilenet_v1 0.75 depth* mudel on näidanud umbes sama energiakulu väärtust - 0,05 Ws. Kõige kulukama mudelina osutus *ssd_mobilenet_v2 quantized*, mille keskmine energiakulu oli 0,059 Ws.

Saadud tulemuste põhjal võib järeldada, et suurem energiakulu võib olla põhjustatud järgmiselt:

1. need mudelid olid vaikimisi treenitud ligi 90 klasside peal, kui samal ajal meie poolt taastreenitud kasutavad ainult 2 klassi,
2. nende mudelite arhitektuurid erinevad meie valitud mudelitest ning nimelt need arhitektuuri erinevused võivad nõuda rohkem energiat.

Kõikide mudelite lühendatud kokkuvõte (Tabel 25).

Tabel 25. Kogu testimise kokkuvõte

		Pildi töötluse kokkuvõte		Energiakulu kokkuvõte		
	Mudel	Lennujaama täpsus	Lennuki täpsus	Minimaalne energiakulu, *Ws*	Maksimaalne energiakulu, *Ws*	Keskmine energiakulu, *Ws*
1 test	ssd_mobilenet_v1 1000 pos	41%	53%	0,020	0,029	0,021
	ssd_mobilenet_v2 1000 pos	51%	50%	0,021	0,030	0,022
	ssd_mobilenet_v1 1000 pos 500 neg	49%	48%	0,050	0,057	0,052
	ssd_mobilenet_v2 1000 pos 500 neg	50%	50%	0,053	0,065	0,055
	ssd_mobilenet_v1 2000 pos	51%	50%	0,019	0,023	0,020
	ssd_mobilenet_v2 2000 pos	50%	50%	0,021	0,025	0,022
	ssd_mobilenet_v1 2000 pos 1000 neg	48%	68%	0,020	0,026	0,020
	ssd_mobilenet_v2 2000 pos 1000 neg	54%	50%	0,021	0,021	0,022
2 test	ssd_mobilenet_v1 2000 pos 1000 neg adam optimizer	53%	56%	0,020	0,028	0,021
	ssd_mobilenet_v2 2000 pos 1000 neg adam optimizer	50%	50%	0,021	0,030	0,022
	ssd_mobilenet_v1 2000 pos 1000 neg momentum optimizer	53%	56%	0,020	0,025	0,020
	ssd_mobilenet_v2 2000 pos 1000 neg momentum optimizer	71%	51%	0,021	0,027	0,022
	ssd_mobilenet_v1 2000 pos 1000 neg momentum depth mult 0.75	50%	50%	0,013	0,015	0,014

	ssd_mobilenet_v2 2000 pos 1000 neg momentum depth mult 0.75	50%	50%	0,017	0,026	0,018
	ssd_mobilenet_v1 2000 pos 1000 neg adam batch 30	50%	50%	0,020	0,023	0,020
	ssd_mobilenet_v2 2000 pos 1000 neg adam batch 1	54%	63%	0,017	0,019	0,018
	ssdlite_mobilenet_v2 2000 pos 1000 neg adam optimzier	39%	58%	0,019	0,022	0,020
3 test	ssd_mobilenet_v1 0.75 depth			0,015	0,015	0,015
	ssd_mobilenet_v1 0.75 depth quantized			0,036	0,040	0,037
	ssd_mobilenet_v1 quantized			0,048	0,049	0,048
	ssd_mobilenet_v2 quantized			0,058	0,061	0,059

8 Treenimise GUI rakendus

Projekti töö käigus me tegime mitu erinevat eraldi koodi tükki protsessi optimeerimiseks ja kiirendamiseks. Sellest lähtudes hakkasime mõtlema ka treenimiseks ettevalmistamise ja treenimise protsessi ise mugavamaks tegemisest ja lihtsustamisest.

8.1 Idee ja kasu

Praegusel hetkel kõik treenimiseelsed sammud oli vaja läbida manuaalselt ning mingil viisil optimeerida seda ei olnud võimalik. Selle osa paremaks muutmiseks mõtlesime GUI rakendus välja, mille raames kasutaja saaks ainult klikkides valmistada kõik vajalikud andmed ette ja käivitada treenimist valitud ajaperioodiks ehk ei pea ise aega jälgima.

8.2 Võimalused

Treenimise GUI rakenduse põhirõhk on treenimiseks ettevalmistamisele. Programm ei lase genereerida mudeli jaoks nõutud failid enne, kui kõik esialgsed andmed on kasutaja poolt rakendusse üleslaetud. Kõik laetavad failid ja tehtavad valikud valideeritakse. Valideerimise ebaõnnestumise korral näidatakse kasutajale aken, kus on kirjutatud informatsioon tekkinud vea kohta, et probleemi oleks võimalik parandada ja edasi jätkata. Juhul, kui sammu täitmine õnnestus, siis kuvatakse kasutajale andmete lisamise järjekorras tehtud sammu tulemus.

8.2 Struktuur

Rakenduse avamisel kõigepealt kasutaja näeb pealehte (Lisa 10), mis suunab kas edasi treenimiseks ettevalmistumise või piltide markeerimise info juurde. Kui kasutaja ei ole eelnevalt pilte markeerinud, siis programmis seletatakse kõik see protsess lahti.

Enne treenimise rakenduse põhilehele sattumist kasutaja näeb juhendite lehte (Lisa 11). Sellel lehel kirjeldatakse mõnede sõnade tähendust antud protsessi raames. Sellele järgneb põhileht.

8.3 Rakenduse funktsionaalsus

Treenimise rakenduses on kokku kuus sammu, mida kasutaja peab ise täitma ning mõned toimuvad taustal vajalike andmete olemasolul.

Iga sammu eduka täitmise korral ilmub selle kohta info akna paremas osas - mis toimus ning nupp. Selle nupu peale vajutades kasutaja näeb täiendavat infot - mis sai tehtud, milline valik on tehtud või kus asub valitud fail või kaust (Lisa 14) (Lisa 15).

8.3.1 Kasutaja sammud

Kasutajalt küsitakse peamiselt laadida treenimiseks vajalikud andmed (pildid) ja mudeli kausta, koostata klasside faili ning valida treenimise protsessi mõned parameetrid.

8.3.1.1 Piltide üleslaadimine

Esimese sammuna kasutajalt küsitakse valida kolm kausta positiivsete, negatiivsete ja testimiseks mõeldud piltidega. Positiivsed ja testpildid on kohustuslikud, negatiivsed pildid võib ära jätta.

Kaustas peab kindlasti olema võrdselt palju pilte ja *.xml* faile ehk pildi markeerimise tulemuse faile.

8.3.1.2 Klasside faili genereerimine

Teine samm on klasside faili genereerimine. Vastava nupule vajutades kasutaja näeb eraldi akna, kus saab sisestada soovitud klasside nimed. Klassile vastav numbriline identifikaator genereeritakse automaatselt ning on kasutajale nähtav.

Sisestatud klasse saab ära kustutada koostatud nimekirjast. Selleks on vaja panna linnuke vastava klassi juurde ning vajutada “Remove”. Peale klassi ära kustutamist kõik numbrilised identifikaatorid arvutatakse ümber (Lisa 13).

Vajadusel saab juba genereeritud klasside nimekirja muuta sessiooni jooksul, aga sellisel juhul tuleb teha ka need sammud uuesti, mis vajavad klasside nimekirja (need on taustal jooksvad tegevused, mida pannakse käima treeningu andmete ettevalmistamise nupuga).

8.3.1.3 Mudeli kausta üleslaadimine

Kolmandaks, on vaja laadida mudeli kausta üles. Sellele sammule samamoodi kehtib valideerimine - programm kontrollib, et kaust sisaldaks kõik vajalikud mudeli failid (*pipeline.config*, *model.ckpt* failid).

8.3.1.4 Piltide tüübi valik

Neljanda sammuna kasutajal tuleb valida, milliste piltidega ta soovib treenida mudelit - kas ainult positiivsete piltidega või nii positiivsete, kui ka negatiivsetega. Esimesel juhul negatiivsete piltide kaust ei ole nõutud, teisel juhul aga valiku tegemine ei õnnestu, kui negatiivsete piltide kaust puudub.

8.3.1.5 Piltide arvu valik

Viienda sammuna tuleb valida kogu piltide arvust see osa, mis läheb treenimisele. Kasutajale pakutakse kolm varianti: kõik pildid ehk 1, pool piltide hulgast ehk ½ või kolm neljandikku kõikidest piltidest ehk ¾ . Peale valiku tegemist programm arvutab taustal arvu, mitu pilti on vaja võtta esialgsest piltide arvust. Lisaks, arvutatakse ka negatiivsete piltide arv positiivsete piltide suhtes - negatiivseid peab olema ½ vähem positiivsetest.

8.3.1.6 Treenimise ajaperioodi valik

Viimase ehk kuuenda sammuna kasutaja peab valima treenimise aega. Treenimise protsessi pikkusena saab valida kas 6, 12 või 24 tundi.

8.3.2 Programmi tausta tegevused

Kui kasutaja on teinud kõik vajalikud sammud ning andmed on programmis olemas, siis tuleb vajutada “Prepare training” nupu peale selleks, et käivitada kõik ülejäänud ettevalmistamise käigud. Selle protsessi automatiseerimine ongi rakenduse idee - kõikide andmete olemasolul jooksutada kõik muud protsessid ühe klikiga.

Selle protsessi käigus genereeritakse *.csv* failid, TFRecord failid, kantakse vajalik arv pilte treenimise kausta sisse, muudetakse mudeli *pipeline.config* fail.

Kasutaja saab kõik andmed üle kontrollida. Samal ajal programm vaatab kõik genereeritud ja üleslaetud andmed üle ning valideerimise õnnestumise korral teeb treenimise alustamise nuppu aktiivseks.

Treenimise protsessi käivitamisel ilmub ekraani peale aken koos edukäigu ribaga ja informatsiooniga, mille kausta sisse hakatakse jooksvalt treenimise sammud salvestama (Lisa 16).

8.4 Kasutatud tehnoloogiad

Kogu rakenduse loogika on kirjutatud Python programmeerimiskeeles. Graafilise liidese osa on tehtud kasutades PyQt5 teeki, mis võimaldab luua graafilisi objekte ja määrata nende funktsionaalsus. Koodi põhistruktuur jälgib OOP printsiipe ning mõned otsese loogikaga seotud osad (nt hetke teekonda leidmine, erinevad arvutused) on kirjutatud tavaliste meetoditena.

Kommentaarid

Käesoleva projekti töö oli põnev ning teadmisi ja oskusi arendav, kuid samal ajal oli raske ja osutus tõeliseks väljakutseks meie jaoks. Kuigi meil on olemas tehisintellekti teemast põhiteadmised ülikooli programmist, nimelt masinõppe, sügavõppe ja närvivõrkude teemad olid meie jaoks täiesti uued ja varem käsitlemata. Aga vaatamata takistustele ja probleemidele, saime kõikide püstitatud eesmärkideni edukalt jõuda ja kõik tellija nõuded kaasa arvatud jooksvalt tekkinud ära täita.

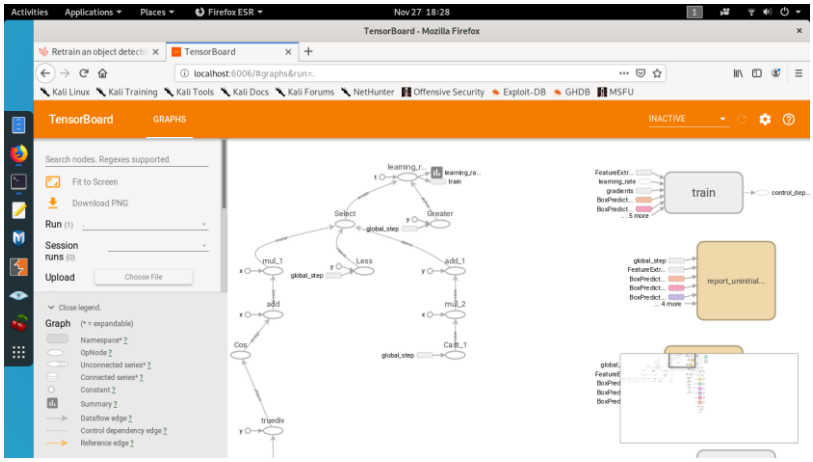
Mudeli treenimine kasutades Docker

Coral'i ametliku veebilehe juhendis oli pakutud teostada mudeli treenimist kasutades Docker'it.

Mudeli treenimine võiks toimuda ka Coral'i seadme abil, vaiaga lisaks kasutaja poolt loodud Docker image’is, mis kasutab arvuti mälu ja protsessori. Sellist variandi kasutades tuleb arvestada sellega, et milline on praegune arvuti seisund ehk kui palju vaba mälu on, milline CPU ja kui palju HDD on.

Kasutasime Dell Latitude E6320, 3 core, Intel core i5, 8GB mälu, HDD 512GB.

Pärast käivitamist, arvuti töö protsess sai oluliselt aeglasemaks. Sellele vaatamata suutsime avada TensorFlow töötahvli (Joonis 45), kus olid näidatud treenimise algoritmid, pildid, klassifikaatorid.



Joonis 45. TrensorFlow arhitektuur ja TensorBoard sisu

Võib öelda, et selline treenimise viis on mugav käivitamiseks, aga selleks on vaja piisavalt võimsamat arvutit, mille peal on vähemalt 64GB mälu ja minimaalselt 5 tuumat CPU. See meie arvuti, mille peal oli võimalik kasutada Coral rakendust, kahjuks ei sobinud madalate tehniliste andmete tõttu.

Tehnoloogiate valiku probleemid

Mudelite treenimine ja testimine

Vaatamata sellele, et TensorFlow tarkvarateek ja selle Object Detection API on hästi levinud masinõppe ja närvivõrkude arendajate seas, nende tehnoloogiatega on tekkinud mõned probleemid. Need takistused olid päris tüüpilised ning sarnased küsimused on tulnud ka teistest kasutajatest, aga TensorFlow ja Object Detection API arendajad kahjuks ei suutnud vastata ega vihjet anda. Seetõttu mõni probleem jäi lahendamata (näiteks, uuesti treenimata mudelite konverteerimine *.tflite* formaadiks)

Treenimise GUI rakendus

Treenimise rakenduse juurde tahtsime lisada ka kohe piltide markeerimise võimalust, aga kahjuks LabelImg lähtekoodi omanikud ei luba nende koodi kasutada teiste rakendustega koos. Seega lisasime meie rakenduse sisse juhendi, kust saab markeerimise rakenduse alla laadida ja kuidas seda peab kasutama.

Rakenduse sees treenimise koodi käivitamine on keskmise võimsusega arvuti jaoks natukene ülekoormav. Programm ei jookse kokku ja treenimise protsess käivitub, aga graafiline liides ei jõua korralikult kujuneda.

Edasised sammud

Meie projekti raames said leitud parimad treenimiseks vajalikud andmekoguse ja mudeli arhitektuuri parameetrite kombinatsioonid. Tänu tehtud tööle nüüd on võimalik põhjalikum arendada objekti tuvastamise mudelid - kasutada rohkem aega treenimise protsessis ning laiendada treenimiseks vajaliku andmekogust ehk tekitada rohkem pilte, mille peal mudelit treenida. Antud töö jooksul kogutud teadmised ja tulemused võivad osutuda väga kasulikuks järgmiste masinõppega seotud projektide tegijatele ning ilmtingimata aitavad Marduk Technologies OÜ kiirendada uurimisprotsessi ja rakendada saadud teadmised oma toote edasi ehitamiseks.

Treenimiseks mõeldud kasutajasõbralik rakendus on suurepärane võimalus ka mitte tehnilistele inimestele luua endale objekti tuvastamise mudeli. Näiteks, sellist rakendust võiks kasutada koolides informaatika tundides ning võimaldada õpetada masinõppe ja närvivõrkude teemasid ka kooli tasemel. Meie treenimise GUI rakendust saab tulevikus oluliselt täiendada ja laiendada selle funktsionaalsust. Praeguseks hetkeks saime panna hästi tugeva aluse selle idee edasi arendamiseks.

Kasutatud allikad

[1] "https://en.wikipedia.org/wiki/TensorFlow", [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/TensorFlow. [Kasutatud 3 Veebruar 2020]

[2] YOLO: Real-Time Object Detection. [Võrgumaterjal]. Available: https://pjreddie.com/darknet/yolo/. [Kasutatud 3 Veebruar 2020].

[3] GitHub. Repository "raccoon_dataset", Dat Tran, 2017. [Võrgumaterjal]. Available: https://github.com/datitran/raccoon_dataset. [Kasutatud 4 Veebruar 2020]

[4] Coral. Get started with the USB Accelerator. [Võrgumaterjal] Available: https://coral.ai/docs/accelerator/get-started/ [Kasutatud 4 Veebruar 2020]

[5] Cornell University, Computer Vision and Pattern Recognition," SSD: Single Shot MultiBox Detector", Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed,Cheng-Yang Fu, Alexander C. Berg, 2017 [Võrgumaterjal]. Available: https://www.researchgate.net/profile/Dumitru_Erhan/publication/308278279_SSD_Single_Shot_MultiBox_Detector/links/586e874c08ae8fce491c8acd/SSD-Single-Shot-MultiBox-Detector.pdf. [Kasutatud 2 Aprill 2020]

[6] Medium. Octavian-AI, "How to pick up the best learning rate for your machine learning project", David Mack, 2018. [Võrgumaterjal]. Available: https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2. [Kasutatud 28 Aprill 2020]

[7] Towards Data Science. "Understanding Learning Rates and How It Improves Performance In Deep Learning", Hafidz Zulkifli, 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10. [Kasutatud 28 Aprill 2020]

[8] Towards Data Science. "Stochastic Gradient Descent with momentum", Vitaly Bushaev, 2017. [Võrgumaterjal]. Available: https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d. [Kasutatud 12 Mai 2020]

[9] TensorFlow. Tutorials, "TFRecord and tf.Example", [Võrgumaterjal]. Available: https://www.tensorflow.org/tutorials/load_data/tfrecord. [Kasutatud 12 Mai 2020]

[10] Machine Learning Crash Course, "Classification: True vs. False and Positive vs. Negative", [Võrgumaterjal]. Available: https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative. [Kasutatud 12 Mai 2020]

[11] PaperspaceBlog. "Intro to optimization in Deep Learning: Momentum, RMSprop and Adam", Ayoosh Kathuria, 2018. [Võrgumaterjal]. Available: https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/. [Kasutatud 12 Mai 2020]

[12] Towards Data Science. “Adam - latest trends in deep learning optimization.”, Vitaly Bushaev, 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c. [Kasutatud 12 Mai 2020]

[13] Towards Data Science. “The 3 Best Optimization Methods in Neural Networks”, Marco Peixeiro, 2019. [Võrgumaterjal]. Available: https://towardsdatascience.com/the-3-best-optimization-methods-in-neural-networks-40879c887873. [Kasutatud 12 Mai 2020]

[14] Towards Data Science. “Understanding RMSprop - faster neural network learning”, Vitaly Bushaev, 2018. [Võrgumaterjal]. Available: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a. [Kasutatud 12 Mai 2020]

[15] Machine Learning Mastery. “A Gentle Introduction to Tensors for Machine Learning with NumPy”, Jason Brownlee, 2018. [Võrgumaterjal]. Available: https://machinelearningmastery.com/introduction-to-tensors-for-machine-learning/. [Kasutatud 12 Mai 2020]

[16] Pathmind. “A Beginner's Guide to Neural Networks and Deep Learning”, [Võrgumaterjal]. Available: https://pathmind.com/wiki/neural-network. [Kasutatud 13 Mai 2020]

[17] Github. “LabelImg”, tzutalin, 2020.[Võrgumaterjal]. Available:

https://github.com/tzutalin/labelImg [Kasutatud 13 Mai 2020]

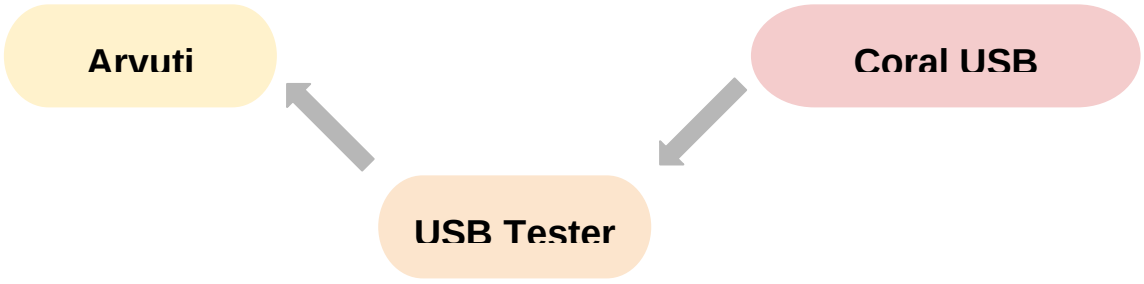
[18] Github, “Satellite Aerial Image Retrieval”, llgeek, 2018.[Võrgumaterjal]. Available:

https://github.com/llgeek/Satellite-Aerial-Image-Retrieval [Kasutatud 13 Mai 2020]

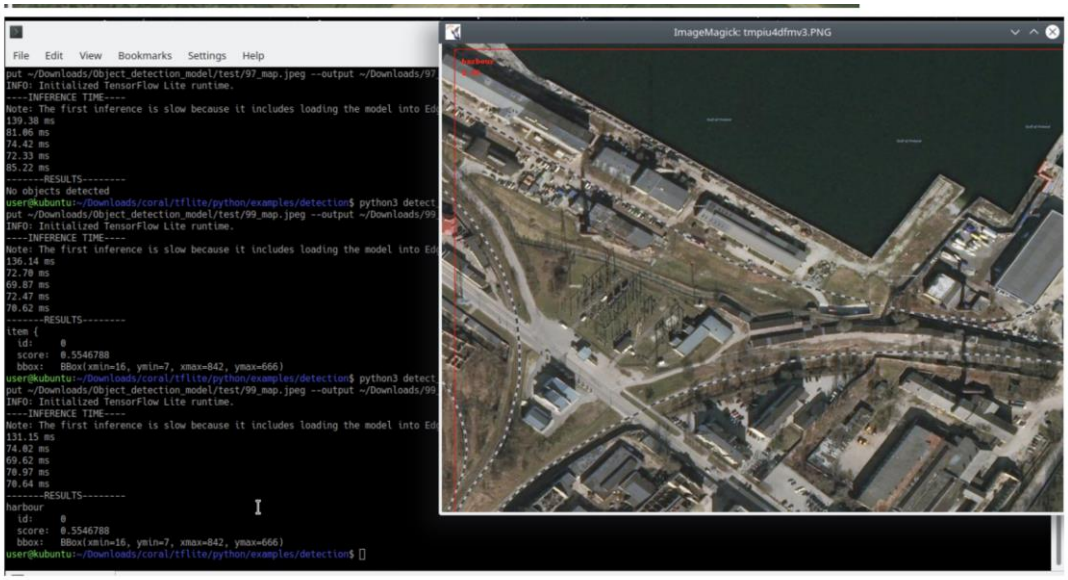
[19] Github, “TensorFlow Object Detection API”, myproblemchild, 2017.[Võrgumaterjal]. Available:

https://github.com/tensorflow/models/tree/master/research/object_detection?fbclid=IwAR1vv3Edginz6hLdYgoZyiFVl5SQZXCpsQsU1R0_waoWsQ23sIUNJgP-A2w[Kasutatud 18 Mai 2020]

Lisa 1 - “Coral USB ja arvuti vahele energiakulu mõõteriista ühendamine”



Lisa 2 - “Objekti tuvastamine pildi peal kasutades Coral USB akseleraatorit”



Lisa 3 - “XML failidest CSV failide genereerimise kood”

Järgnev kood pärineb allikast [3] *xml_to_csv.py*:

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET


def xml_to_csv(path):
   xml_list = []
   for xml_file in glob.glob(path + '/*.xml'):
       tree = ET.parse(xml_file)
       root = tree.getroot()
       for member in root.findall('object'):
           value = (root.find('filename').text,
                    int(root.find('size')[0].text),
                    int(root.find('size')[1].text),
                    member[0].text,
                    int(member[4][0].text),
                    int(member[4][1].text),
                    int(member[4][2].text),
                    int(member[4][3].text)
                    )
           xml_list.append(value)
   column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin',
'xmax', 'ymax']
   xml_df = pd.DataFrame(xml_list, columns=column_name)
   return xml_df



def main():
```

```
    image_path = os.path.join(os.getcwd(), 'annotations')
    xml_df = xml_to_csv(image_path)
    xml_df.to_csv('raccoon_labels.csv', index=None)
    print('Successfully converted xml to csv.')


main()
```

Lisa 4 - “TFRecord failide genereerimise kood”

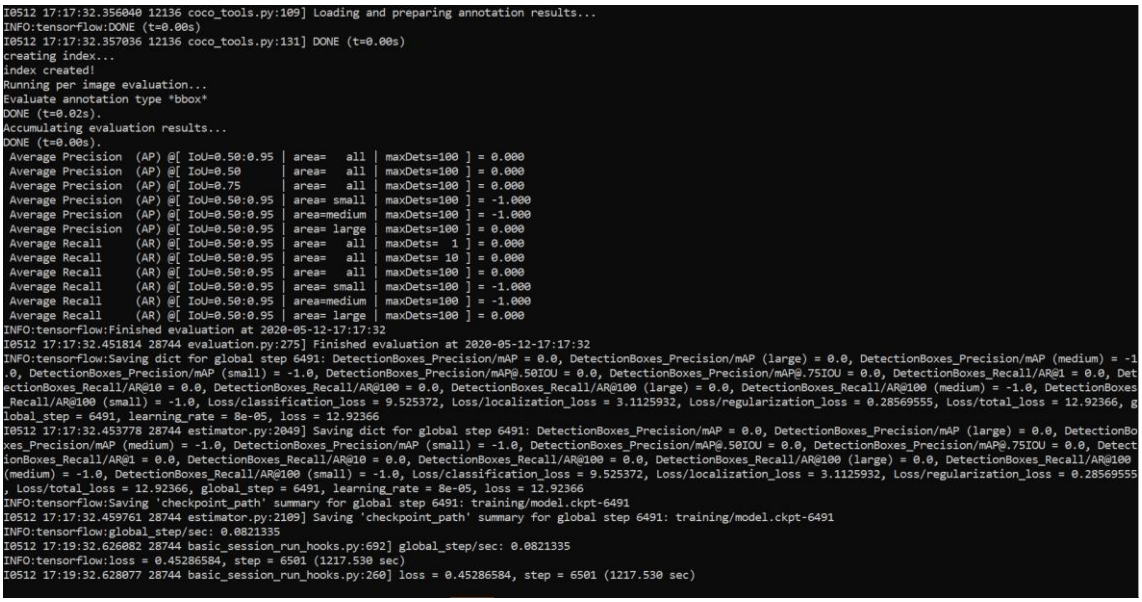
Järgnev kood pärineb allikast [3] *generate_tfrecord.py*:

```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd
import tensorflow as tf
from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.compat.v1.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('output_path', '', 'Path to output TFRecord')
flags.DEFINE_string('image_dir', '', 'Path to images')
FLAGS = flags.FLAGS

# REPLACE WITH OUR LABELS
def class_text_to_int(row_label):
   if row_label == 'plane':
       return 1
   elif row_label == 'airport':
       return 2
   else:
       return 0
```

```
def split(df, group):
   data = namedtuple('data', ['filename', 'object'])
   gb = df.groupby(group)
   return [data(filename, gb.get_group(x)) for filename, x in
zip(gb.groups.keys(), gb.groups)]


def create_tf_example(group, path):
   with tf.io.gfile.GFile(os.path.join(path, '{}'.format(group.filename)),
'rb') as fid:
       encoded_jpg = fid.read()
   encoded_jpg_io = io.BytesIO(encoded_jpg)
   image = Image.open(encoded_jpg_io)
   width, height = image.size


   filename = group.filename.encode('utf8')
   image_format = b'jpg'
   xmins = []
   xmaxs = []
   ymins = []
   ymaxs = []
   classes_text = []
   classes = []
   for index, row in group.object.iterrows():
       xmins.append(row['xmin'] / width)
       xmaxs.append(row['xmax'] / width)
       ymins.append(row['ymin'] / height)
       ymaxs.append(row['ymax'] / height)
       classes_text.append(row['class'].encode('utf8'))
       classes.append(class_text_to_int(row['class']))


   tf_example = tf.train.Example(features=tf.train.Features(feature={
```

```
        'image/height': dataset_util.int64_feature(height),
        'image/width': dataset_util.int64_feature(width),
        'image/filename': dataset_util.bytes_feature(filename),
        'image/source_id': dataset_util.bytes_feature(filename),
        'image/encoded': dataset_util.bytes_feature(encoded_jpg),
        'image/format': dataset_util.bytes_feature(image_format),
        'image/object/bbox/xmin': dataset_util.float_list_feature(xmins),
        'image/object/bbox/xmax': dataset_util.float_list_feature(xmaxs),
        'image/object/bbox/ymin': dataset_util.float_list_feature(ymins),
        'image/object/bbox/ymax': dataset_util.float_list_feature(ymaxs),
        'image/object/class/text':
dataset_util.bytes_list_feature(classes_text),
        'image/object/class/label': dataset_util.int64_list_feature(classes),
    }))
    return tf_example


def main(_):
    writer = tf.io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())


    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords: {}'.format(output_path))


if __name__ == '__main__':
    tf.compat.v1.app.run()
```

Lisa 6 - “Treenimise protsessi jooksev aruanne konsoolis”



Lisa 7 - “Loop.py fail”

```
from detect_image import main

def test():
      model_full_path = '/home/user/Downloads/models_without_training/'
      img_fullpath = '/home/user/Downloads/test_for_not_trained_models/'

      models = ['ssd_mobilenet_v1_0.75_depth/',
            'ssd_mobilenet_v1_0.75_depth_quantized/',
            'ssd_mobilenet_v1_quantized/',
            'ssd_mobilenet_v2_quantized/']
      classes =
'/home/user/Downloads/Object_detection_model/classes_test.txt'
      pics = ['family.jpg', 'park.jpg', 'street.jpg', 'pets.jpg', 'food.jpg']

      for m in models:
      model = model_full_path + m + 'converted_model.tflite'
      for p in pics:
            print(p)
            image_full = img_fullpath + p
            main(model, image_full, classes)


if __name__ == '__main__':
  test()
```

Lisa 8 - “Detect_image.py faili modifitseeritud meetod main()”

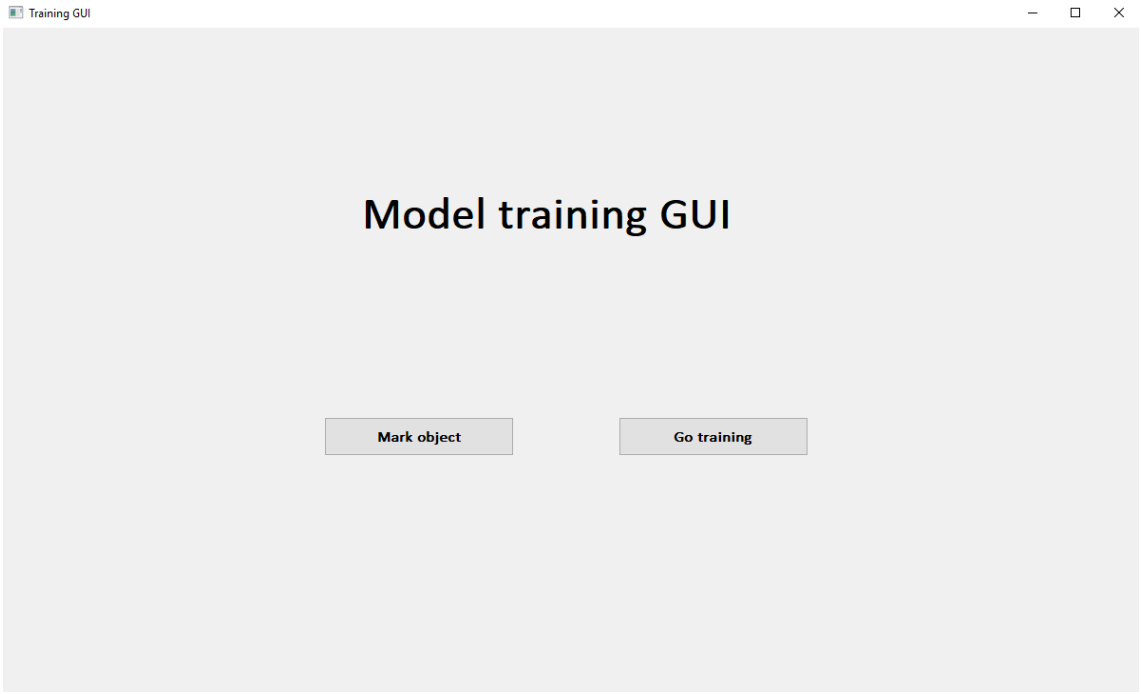
```
def main(m,i, l):
  parser = argparse.ArgumentParser(
      formatter_class=argparse.ArgumentDefaultsHelpFormatter)
  #parser.add_argument('-m', '--model', required=True,help='File path of
.tflite file.')
  #parser.add_argument('-i', '--input', required=True,help='File path of
image to process.')
  parser.add_argument('-l', '--labels',
                      help='File path of labels file.')
  parser.add_argument('-t', '--threshold', type=float, default=0.4,
                      help='Score threshold for detected objects.')
  parser.add_argument('-o', '--output',
                      help='File path for the result image with
annotations')
  parser.add_argument('-c', '--count', type=int, default=5,
                      help='Number of times to run inference')
  args = parser.parse_args()

  labels = load_labels(l)
  interpreter = make_interpreter(m)
  interpreter.allocate_tensors()

  image = Image.open(i).convert('RGB')
  scale = detect.set_input(interpreter, image.size,
                      lambda size: image.resize(size, Image.ANTIALIAS))

  print('----INFERENCE TIME----')
  print('Note: The first inference is slow because it includes',
      'loading the model into Edge TPU memory.')
  f = open("results.txt", "a")
  t = open("time.txt", "a")
  counter = 0
  for itr in range(args.count):
      start = time.monotonic()
      interpreter.invoke()
      inference_time = time.monotonic() - start
      objs = detect.get_output(interpreter, args.threshold, scale)
      if itr != 0:
      counter += (inference_time * 1000)
      print('%.2f ms' % (inference_time * 1000))

  counter = counter / 4000
  print('AVERAGE %.3f s' % counter)
  t.write('%.3f\n' % counter)

  print('-------RESULTS--------')
  if not objs:
      f.write(m + ' ,'+ i + ', '+ 'No objects detected \n')
      print('No objects detected')
```

```
  for obj in objs:
      print(labels.get(obj.id, obj.id))
      print('  id:      ', obj.id)
      print('  score: ', obj.score)
      print('  bbox:  ', obj.bbox)
      f.write(m+ ' ,'+ i + ', '+ labels.get(obj.id, obj.id) + ' : ' +
str(obj.score) + '\n')

  if args.output:
      draw_objects(ImageDraw.Draw(image), objs, labels)
      image.save(img_out)
      image.show()
```

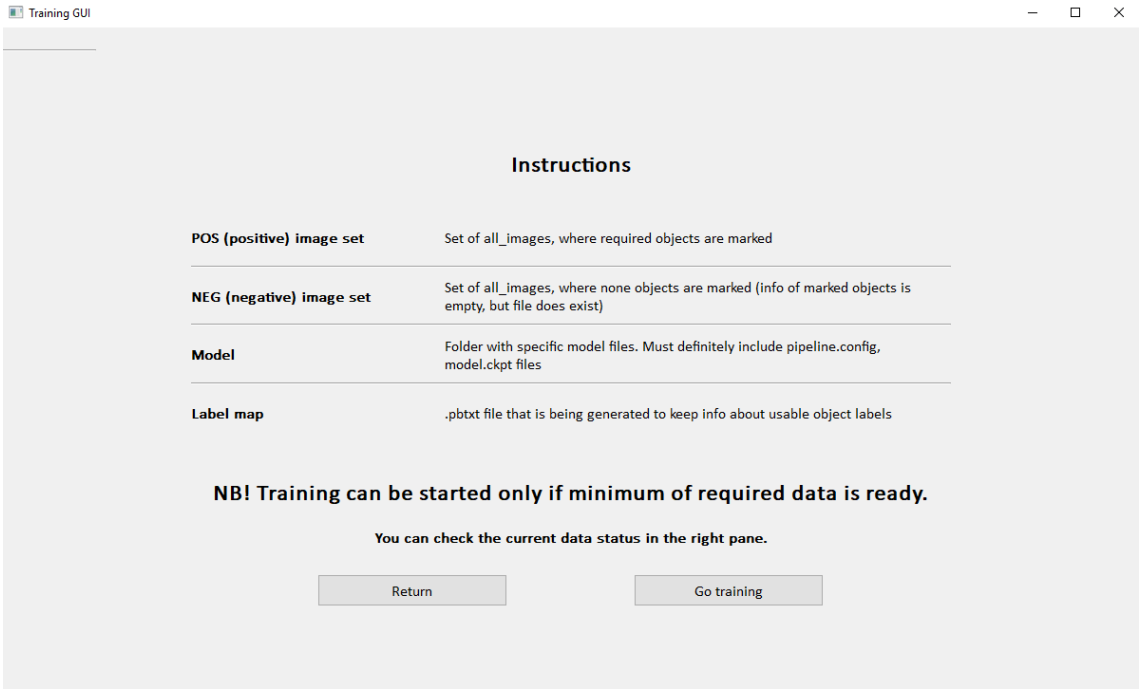
Lisa 9 - “Converter.py fail”

```
from functools import reduce

def average(arr):
      return reduce(lambda a, b: a + b, arr)/len(arr)

def convert():
      f = open("results.txt", "r")
      wr = open("airports.txt", "a")
      lines = f.readlines()

      res = {}

      for line in lines:
      el = line.split(',')
      e = el[-1].split(':')
      if e[0].strip() == 'airport':
            if int(el[0]) in res:
                  new = res.get(int(el[0]))
                  new.append(float(e[1].strip().rstrip()))
                  res[int(el[0])]= new
            else:
                  res[int(el[0])] = [float(e[1].strip().rstrip())]

      for i in range(1, 151):
      if i in res:
            wr.write(str(average(res.get(i))).replace('.',',') + '\n')
      if i not in res:
            wr.write('\n')



if __name__ == '__main__':
  convert()
```

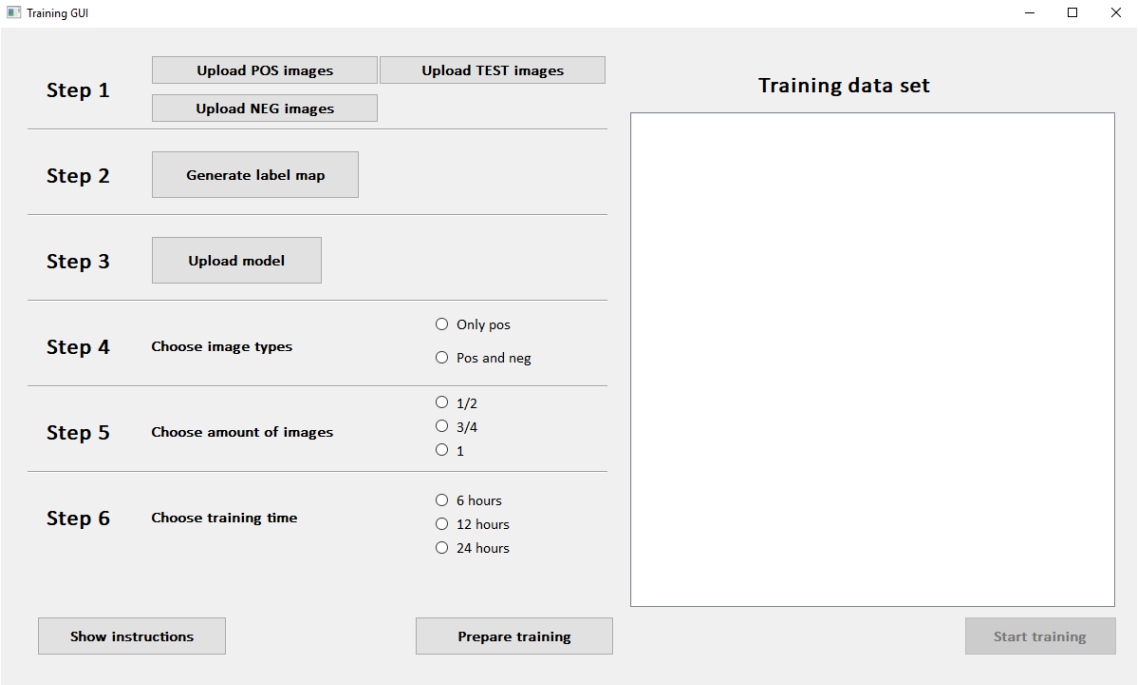
Lisa 10 - “Treenimise rakenduse pealeht”



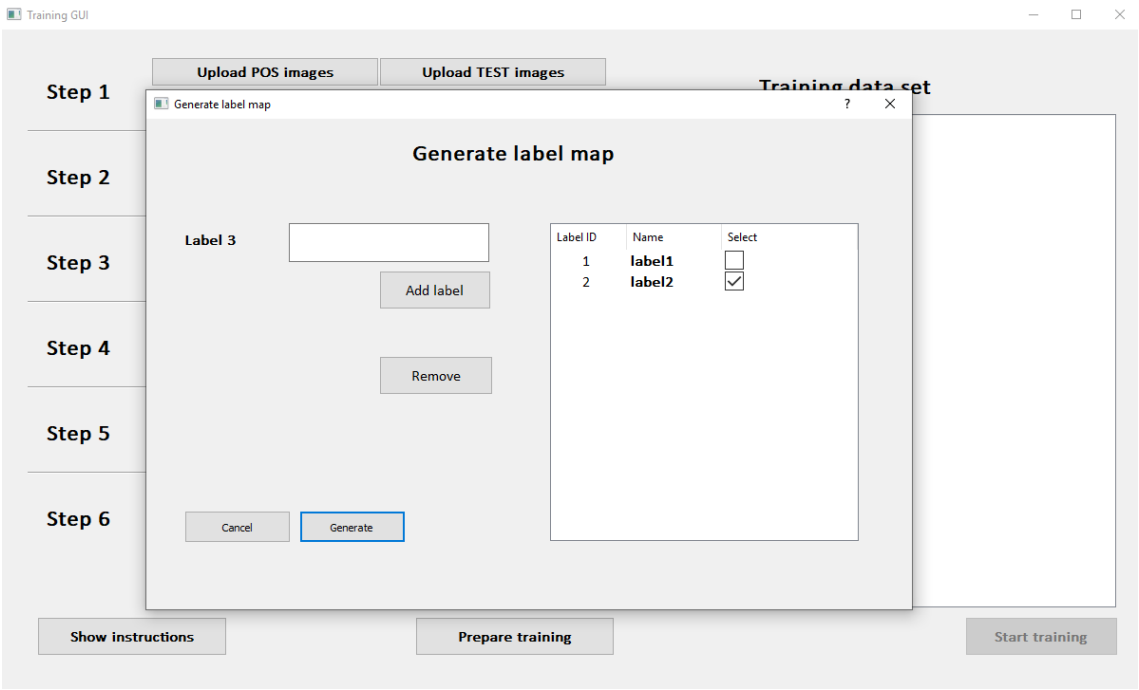
Lisa 11 - “Treenimise rakenduse juhend”



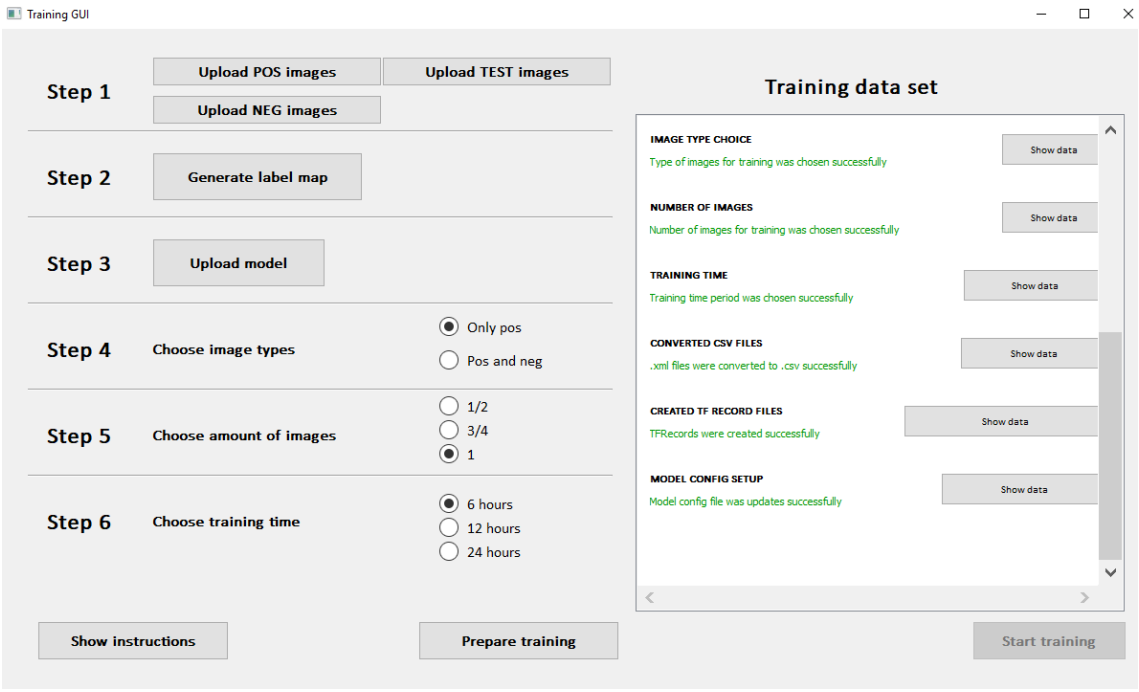
Lisa 12 - “Treenimise rakenduse põhileht”



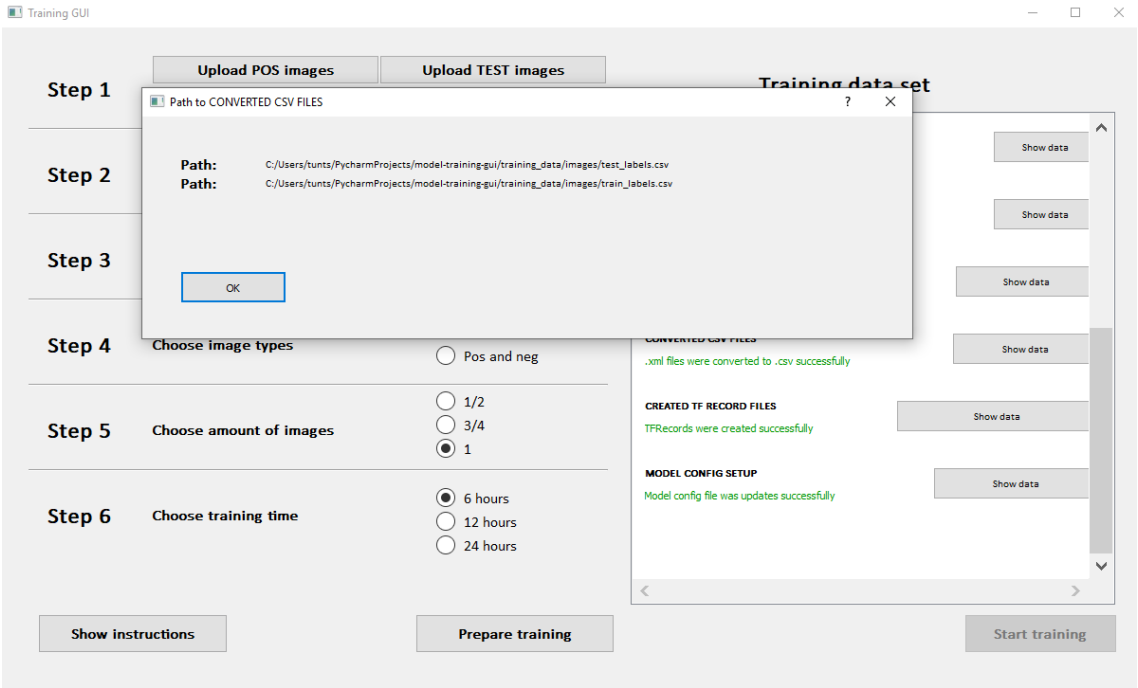
Lisa 13 - “Klasside nimekirja genereerimine



Lisa 14 - “Treenimise leht läbitud sammude infoga”



Lisa 15 - “Treenimise sammu täiendav info”



Lisa 16 - “Treenimise protsessi käivitamine rakenduses”