TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Lasse Thor Lepik 223042IADB

PIKSELLEERITUD TEKSTI LUGEMINE OPTILISE MÄRGITUVASTUSE ABIL

Bakalaureusetöö

Juhendaja: Andres Käver
MSc

Tallinn 2025

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Lasse Thor Lepik

14.05.2025

Annotatsioon

See lõputöö keskendub pikselleeritud teksti optilisele märgituvastusele (OCR). Eesmärgiks on tuvastada sümboleid väga madala eraldusvõimega piltidelt, kus tähemärgid on vaid 1–6 pikslit kõrged. Selguskadu põhjuseks võib sageli olla tahtlik andmete varjamine või lihtsalt eraldusvõime kadu liigse töötlemise tõttu. See töö keskendub täpsemalt tahtlikult kahjustatud piltide lugemisele. Uuritavad pildid on peamiselt töötletud levinud algoritmi abil, mis jagab pildi plokkideks ja asendab iga ploki pikslid selle ploki keskmise värvitooniga.

Uuringu raames töötatakse välja erilised masinõppemudelid, et seda probleemi efektiivselt lahendada. Töö käigus uuritakse erinevaid närvivõrkude arhitektuure ja häälestatakse hüperparameetreid. Katsed näitavad, et kitsale määramispiirkonnale (*domain*), näiteks numbrilisele informatsioonile, spetsiaalselt disainitud mudelid saavutavad märkimisväärselt kõrge täpsuse tänu piiratud väljundklasside arvule. Lisaks uuritakse GAN-põhiseid superresolutsiooni tehnikaid, hinnates nende võimekust muuta kujutisi loetavateks. Tulemused näitavad, et GAN-mudelid võivad olla efektiivsed, kuid väga madalal eraldusvõimel pole need eriti täpsed. Tõestatakse automaatse mudelivaliku (*Mixture of Experts*) võimekust ja pakutakse välja konsensusmehhanismide süsteem. Lõputöö pakub avatud lähtekoodiga tarkvara ja eelnevalt treenitud mudeleid, mis generaliseerivad üle erinevate fontide ja plokisuuruste. Arendatud lahendus saavutab kõrge tuvastustäpsuse, jäädes samas arvutuslikult tõhusaks ja seega praktiliselt rakendatavaks reaalsetes olukordades.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 5 peatükki, 22 joonist, 4 tabelit.

Abstract

Reading Pixelated Text Using Optical Character Recognition

This thesis focuses on the optical character recognition (OCR) of highly pixelated text, aiming to accurately identify characters from images with low resolution, ranging from 1 to 6 pixels in height. Pixelated text recognition is particularly challenging due to the loss of clarity caused by intentional censoring or image degradation from repeated processing. This paper focuses on reading intentionally damaged images. The main algorithm used for processing is „block averaging“, which splits images into blocks and averages all colors in a block.

The research develops specialized machine learning models to address this problem effectively. The study investigates various neural network architectures and hyperparameter tuning. Experiments demonstrate that specialized models designed for narrow domains, such as numeric data, achieve remarkably high accuracy due to their limited output class set. Additionally, the research explores the effectiveness of Generative Adversarial Network based super-resolution techniques, assessing their potential to enhance pixelated images. Findings show potential benefits but highlight significant limitations when dealing with low-resolution images. The thesis provides open-source software and trained models designed to generalize across various fonts and block sizes. Work proves viability of automatic model selection (Mixture of Experts) and proposes consensus mechanisms. The developed solution achieves high accuracy while remaining computationally efficient, making it practically accessible for real-world applications.

The thesis is written in Estonian and is 29 pages long, including 5 chapters, 22 figures and 4 tables.

Lühendite ja mõistete sõnastik

Block averaging	Pilditöötlusmeetod, mis jagab pildi väikesteks plokkideks ja asendab iga ploki pikslid selle ploki keskmise värvitooniga.
Block shift	Plokinihe. Kui *block averaging* algoritm jaotab pildi ruudustikuks, siis see nihe kirjeldab, mitme piksli jagu on ruudustikku nihutatud. Pildil on nii X kui Y nihe. Kui nihe on sama suur kui plokk, siis rakenduslikult jõuab see ringiga tagasi nulli. Lõplik X nihe = X nihe % ploki X dimensioon.
Batch	Tegumipakk, töötlus- või käsitlusüksuste kogum. Kasutatakse sageli, et paralleelselt mitut üksust töötleda.
CAPTCHA	*Completely Automated Public Turing test to tell Computers and Humans Apart*. Test, mis eristab inimesi arvutiprogrammidest, tavaliselt eesmärgiga viimaseid tõrjuda.
CER	*Character Error Rate* kirjeldab kui suur osa sümbolitest on keskmises sõnes õigesti ennustatud. CER = 1 - CRR.
CRR	*Character Recognition Rate* on antud kontekstis sõne *Levenshtein distance* erinevus jagatud sõne pikkusega.
CTC *blank*	*Connectionist Temporal Classification blank* on masinõppes kasutatav tühimärk (sümbol), mida rakendavad mudelid sisemiselt edukuse tõstmiseks. Seda märki ennustuste lõpptulemustes ei kuvata. Tühimärk ei ole tühik.
De Bruijn Sequence	Tsükliline jada, mis on koostatud antud tähestiku elementidest nii, et iga fikseeritud pikkusega jadaosa esineb selles täpselt üks kord.
Dropout	Tähistab kui suur osa neuronitest närvivõrgus igal iteratsioonil välja lülitatakse. Peamine eesmärk on vähendada sõltuvust kindlatest neuronitest ja parandada generaliseerimist.

Epohh	Kasutatakse mudelite treenimismahu mõõtmiseks. Kui mudel käib kogu treeningandmestikust üle N korda, siis on mudelit treenitud N epohhi jagu.
GAN	*Generative Adversarial Network* ehk generatiivne võistlev võrk on närvivõrgu arhitektuur, kus generaator- ja diskriminaatorvõrk mõlemad õpivad ning võistlevad üksteisega, parandades seeläbi generaatormudelit.
HMM	*Hidden Markov Model*. OCR kontekstis aitab närvivõrgul teha täpsemaid ennustusi keelelise statistika põhjal. Kui sõnedes pole mustreid, siis HMM ei lisa mingit väärtust.
Levenshtein distance	Levenshteini kaugus võrdleb kahte sõne ja näitab, mitu korda tuleb tähti lisada, eemaldada või vahetada, selleks et esimesest sõnest saaks teine sõne.
LSTM	*Long Short-Term Memory* on rekurrentne närvivõrk, mis säilitab peale treeningandmete töötlemist siseolekuid.
MoE	*Mixture of Experts* on masinõppemeetod, mille puhul otsustav kontroller valib sisendi põhjal parima spetsialistmudeli või määrab mudelite kaale.
Närvivõrgud	Ajust inspireeritud masinõppemudelid, mis suudavad tuvastada keerukaid mustreid.
OCR	Optiline märgituvastus. Tehnoloogia, mis võimaldab pildil kujutatud teksti digitaalseks sõneks muuta.
Pikselleerimine	Pildi või selle sektsiooni tajutava eraldusvõime vähendamine, kasutades näiteks algoritmi „*block averaging*”.
ReLU	*Rectified Linear Unit* on lineaarne aktivatsioonifunktsioon, mille väljund võib olla nullist lõpmatuseni.
SRR	*String Recognition Rate* on õigesti ennustatud sõnede ja sõnede koguarvu suhe.
SER	*String Error Rate* kirjeldab ebatäiuslike ennustuste esinemise sagedust kõigi ennustuste seas, SER = 1 - SRR.

Sisukord

Jooniste loetelu

Tabelite loetelu

1. Sissejuhatus

Optilise märgituvastuse tehnoloogiat kasutatakse, et automatiseeritud viisil lugeda fotodelt või skaneeritud dokumentidelt teksti, tuvastada numbrimärke või digitaalselt talletada käsitsi kirjutatud infot. Optiline märgituvastus on samuti kasutusel pildipõhiste CAPTCHA-süsteemide petmiseks ning peale moonutatud teksti võimaldab ka lugeda pikselleeritud teksti [1]. Madala eraldusvõimega pildid ei võimalda neilt selgelt teksti lugeda. Selguskadu põhjuseks võib olla tahtlik andmete pikselleerimine või lihtsalt eraldusvõime kadu liigse töötlemise tõttu, mille üks põhjus võib olla faili korduv ringlemine sotsiaalmeedias. Teksti tuvastamine on eriti oluline valdkondades nagu turvalisus ja luure. Mitte ainult ei võimalda OCR siinkohal piltidelt info lugemist automatiseerida, vaid hea süsteemi puhul tuvastada inimese jaoks loetamatut teksti [1].

Turvakaalutlustel eemaldatakse sageli tundlikud andmed pildidelt pikselleerimise abil. Probleem seisneb selles, et neid pikselleeritud sõnesid ei saa pildilt välja lugeda. Nii andmete omanikul kui teistel osapooltel võib tekkida vajadus andmed pikselleeritud allika põhjal taastada. Kuidas lugeda originaalset sõne tahtlikult pikselleeritud pildilt? Optilise märgituvastuse (OCR) rakendamine muutub oluliselt keerukamaks, kui pildil kujutatud tekst on tugevalt pikselleeritud. Kaovad olulised visuaalsed detailid ja traditsioonilised meetodid [2] ei suuda enam usaldusväärselt eristada ega tuvastada tähemärke. Tahtlik pikselleerimine, mida kasutatakse sageli privaatsuse tagamiseks või turvalisuse tõstmise eesmärgil, muudab teksti lugemise keeruliseks, kuna algoritmid peavad arvestama erinevate variatsioonidega ja samas toime tulema väga piiratud lähteinfoga. Erinevad muutujad teevad üldistamise keeruliseks.

Eesmärk on arendada kättesaadav tarkvara, mis kasutab närvivõrgu mudelit/mudeleid, et tuvastada inimsilmale loetamatut pikselleeritud teksti. Töö skoobis on vaid digitaalsed sisendpildid, millel on teksti kõrgus 1–6 pikslit. Teksti piiritletakse väikseima nelinurgaga, millest väljaspool on kõik pikslid täpselt tausta värvi. *Monospace* fontides on kõik sümbolid sama laiad. Antud töö ei käsitle eraldi *monospace* fonte, sest see on vähelevinud

fontide alamgrupp [3]. Lisaks on *monospace* fonte palju lihtsam lugeda, sest ühiksuurus võimaldab segmenteerimist. Töö käigus analüüsitakse mudelite piiranguid ja efektiivsust.

Töö tulemusena loodav lahendus peaks olema:

- Võimeline tuvastama ja lugema äärmiselt pikselleeritud teksti kõrge täpsusega (> 90% CRR).
- Sobiv nii teksti- kui numbrijadade lugemiseks.
- Avatud lähtekoodiga, läbipaistev ning rakendatav reaalsetes olukordades, kus on vaja kiiret tekstituvastust.

2. Analüüs

Pikselleeritud teksti lugemiseks on erinevaid meetodeid. Mõned hõlmavad masinõppe-mudeleid, teised lihtsamaid võrdlevaid algoritme.

2.1 Lahtised küsimused

Järgnevalt on välja toodud erinevad tööga seotud küsimused. Nendele täpsete vastuste leidmine aitaks edendada antud tarkvara arendamist.

- Kuidas tuvastada teksti, mis on pikselleeritud 2-piksli kõrguseks?
- Kas on võimalik vähesel määral teksti tuvastada, kui kõrgus on vertikaalselt 1 piksel? Sellisel juhul esindaks sõne vaid rida piksleid.
- Milliseid masinõppepõhiseid meetodeid ja mudeliarhitektuure on tulus rakendada nii madala eraldusvõimega piltide puhul?
- Kuidas mõjutab monokroomsete piltide kasutamine mudeli jõudlust võrreldes värviliste piltidega?
- Kui hästi suudab üks mudel tuvastada erinevate fontide, renderdus-mootorite ja pikselleerimis-algoritmide abil loodud teksti (üldistada ehk generaliseerida)?
- Kas on kasulik teha kogu spetsialiseerunud mudeleid (*MoE – Mixture of Experts*)?
- Kas mudelite automaatseks valimiseks peaks looma otsuseid tegeva kontroller-mudeli (*gating network*)?
- Kui edukalt on võimalik lugeda täiesti suvalist andmejada, mille puhul ei saa rakendada keelelisel statistikal põhinevaid meetodeid?
- Teadustöid on tehtud, aga kust saab keskmine kasutaja kätte hästi töötava tarkvara, mis ta reaalse probleemi lahendab?

2.2 Kirjanduse analüüs

Antud teemal on kirjutatud mitmeid teadustöid, mis püüavad üldises võtmes lahendada sama probleemi. Töödes kasutati erinevaid lähenemisviise ning uuriti nende efektiivsust. Tipptulemusi saavutavad närvivõrgud, mis rakendavad *bidirectional LSTM (Long ShortTerm Memory)* ja *CTC (Connectionist Temporal Classifier)* komponente. [1], [4], [5]

HMM ehk *Hidden Markov Model* aitab närvivõrgul teha täpseid ennustusi keelelise statistika põhjal [6]. Näiteks kui on tegu udustatud sõnega „MOON“, siis närvivõrk võib pakkuda pikslite põhjal „M00N“ ja HMM lubab närvivõrgul eelistada sarnast, aga statistiliselt tõenäolisemat varianti „MOON“. Kui vaatluse all on suvaline andmejadaga, siis pole sellest lähenemisest kasu. LSTM on samuti võimeline sarnaseid mustreid leidma.

Üks lähenemisviis on kasutada toorest jõudu: proovimine ja võrdlemine. Tööriistad nagu *Depix* ja *Unredacter* püüavad genereerida sarnaseid vasteid pikselleeritud pildile. Need meetodid on aeglased ja ei toimi ideaalselt. *Depix* vajab *de Bruijn* jada ning *Unredacter'il* on hulk probleeme tähtede paiknemise ja segmenteerimisega. [3], [7]

Värvidel on tähtis roll, sest RGB kanalites peituv detailsus võimaldab paremaid ennustusi teha. Monokroomsete piltide kasutamine muudab teksti ääred mudeli jaoks selgemaks, aitab paremini generaliseerida, tõstab robustsust ja võtab vähem ressursse. Ometi võib paljude sarnaste klasside ja spetsiifiliste olukordade puhul värviinfo mudeli täpsust parandada. [8]

Superresolutsiooni-mudelid, mis on spetsiaalselt treenitud kitsal määramispiirkonnal (*narrow domain*) võimaldavad edukalt pikselleeritud teksti lugeda. Tegemist on universaalsema lahendusega, sest mudel ei väljasta sümboleid, vaid hoopis pildi. Seega ei ole meetod otseselt klasside arvuga piiratud. Seevastu pole see lähenemine eriti madala eraldusvõime puhul täpne. [9]

Eeltoodud tööd näitavad, et madala eraldusvõimega tekstide tuvastamise valdkonnas on tehtud edusamme, kuid pole ideaalset, lihtsat, kättesaadavat ja paindlikku tarkvara, mis loeks pildilt usaldusväärselt seosetut teksti. Monokroomsete piltide lugemine on kiirem.

2.3 Töö panus ning uudsus

Luuakse mitmed spetsiaalsete tingimuste jaoks loodud mudelid, näiteks erinevate tekstisuuruste, fontide ja märgijadade (tähed, numbrid) jaoks. Uudne on potentsiaalne kasu kontroller-mudelitest, mis aitavad automaatselt valida parima OCR mudeli kasutaja sisendpildi lugemiseks. Konsensusmehhanismide kasutamine mitme mudeli vahel võib tõsta teksti lugemise täpsust. Leitakse, milline närvivõrgu arhitektuur on kõige efektiivsem ekstreemse piksellatsiooni vastu. Töö käigus täpsustatakse hüperparameetreid. Valmib avatud lähtekoodiga lihtsalt kasutatav ja kiire tarkvara, mis ei toetu vaid toorele jõule ega keelelistele mustritele, et saavutada kõrge pikselleeritud sõnede tuvastustäpsus.

2.4 Metoodika valik ja põhjendus

Käesolev töö kasutab eksperimentaalset kvantitatiivset uurimisstrateegiat, mis keskendub uute masinõppe-mudelite väljatöötamisele ja hindamisele äärmiselt pikselleeritud teksti tuvastamisel. Eksperimentaalse lähenemise valik võimaldab katsetada erinevaid mudeliarhitektuure ja meetodeid, et leida optimaalseim lahendus antud probleemile. Sünteesitakse suur hulk testandmeid, mis hõlmavad erinevaid fonte ning plokisuuruseid. Andmeteks on paarid pikselleeritud piltidest ja sildid piltidel kujutatud tekstidest. Sünteetiliste andmete peal tehtud testide põhjal tehakse tarkvarale täiustusi.

2.5 Meetodite valik

Erinevaid meetodeid hinnati nii eelnevate tulemuste kui ka antud projekti sobivuse põhjal. Otsustati rakendada CTC + LSTM kihtidega närvivõrku ning testida ka kontrollermudelite ning spetsialistmudelite kombinatsiooni võimekust (*MoE*). Lähtepunktina kasutatakse spetsiaalselt kohandatud OCR mudeleid, mis kasutavad konvolutsiooniliste ja LSTM kihtide kombinatsiooni. Närvivõrgud on paindlikud ning võimelised leidma andmetes vaevumärgatavaid mustreid, mistõttu on need antud probleemi lahendamisel kasulikud. Testitakse ka spetsiaalselt treenitud GAN-põhiseid superresolutsioonimudeleid. Järgnevalt on välja toodud võrdlev ülevaade erinevatest meetoditest.

2.5.1 Ülevaade meetoditest

HMM: Kui tekstis puuduvad mustrid, on *hidden Markov model* kasutu, sest selle eesmärk on märkidevahelisi seoseid leida.

Toores jõud: Võrreldakse pikselleeritud pilte sildistatud piltidega, mis on varem ette renderdatud või sünteesitakse reaalajas. Võrreldakse pikslite heledust, et leida milline silt vastab sisendpildile. Paralleeli saab tuua samaräsirünnakuga: luuakse suvalisi andmeid kuni räsi tuleb sama. Võtab massiivselt aega, arvutusjõudu ja skaleerub väga halvasti. Lisaks ei pruugi olla võimalik sünteesida originaalile piisavalt ligilähedasi pilte, eriti kuna nii müra kui JPEG tihendusmoonutus on ennustamatud. [3]

Segmenteerimine: Pilt üritatakse jagada tähtedeks ja siis tehakse igale tähele eraldi otsing. Madala eraldusvõime puhul on raske tähti eraldada, erinevate tähtede pikslid on kokku sulanud. Kui on olemas teksti *de Bruijn* jada, võib see probleemi veidi lihtsustada. [3], [7]

CTC + LSTM närvivõrk: Jadade lugemiseks on väga efektiivne kasutada närvivõrkudel põhinevaid mudeleid, mis sisaldavad CTC ja LSTM kihte. Selliseid mudeleid kasutatakse ka tekstide [5] ning käekirja lugemiseks ja kõnetuvastuseks. Pikselleeritud piltide valkonnas on parimaid tulemusi saavutatud just selle meetodiga. [1], [4]

Superresolutsioonipõhised meetodid: Võivad potentsiaalselt parandada pildi kvaliteeti enne OCR-i rakendamist, kuid üldiste mudelite efektiivsus on nii madalal eraldusvõimel pea olematu. Spetsiaalselt treenitud mudelid võivad aga piiratud määramispiirkonna puhul originaalpildi pikselleeritud andmete põhjal uuesti konstrueerida [9].

Traditsioonilised pilditöötlusmeetodid: Kontrastsuse tõstmine, värvikanalite ning müramustrite analüüs ja teised meetodid ei anna lisainfot, mille põhjal saaks piksleid tekstiks tõlkida. Vaid inimjõul on väga raske või võimatu tugevalt pikselleeritud teksti lugeda.

***Mixture of Experts*:** Treenides kogu spetsialistmudeleid ning seejärel kontrollervõrgu, mis valib mudeli vastavalt olukorrale, on võimalik tõsta süsteemi täpsust. Üks hiljutisi näiteid *MoE* tehnika rakendamisest on DeepSeek. [10]

2.6 Seoseid otsiv masinõppemeetod ja reaalsus

Masinõppemudelid teevad otsuseid statistilisi mustreid leides. Reaalsetes olukordades ei ole kõik mustvalge ega perfektselt mustrina kirjeldatav. Leidub erandeid ning üksteisega kattuvaid olukordi. Päris andmete seas eksisteerivad ka mitmesed (mittedeterministlikud) seosed.

2.6.1 Probleem on absoluutset täpsust nõudes lahendamatu

Pikselleerimine definitsioonilt kitsendab andmeid. See vähendab määramispiirkonda, kui funktsiooniks on lugemine ja muutumispiirkond loetud sõne. Samal ajal kui määramispiirkond väheneb, püsib muutumispiirkond sama. Määramispiirkonna vähendamine tähendab, et on aina vähem infot, mille põhjal funktsioon saaks anda vastava väljundi. Lõpuks jõuab määramispiirkond piisavalt väikseks, et matemaatiliselt on mitmele samasugusele sisendile määratud kaks erinevat sõne. Sisendid on samad, kuid eeldatavad väljundid erinevad. Seos ei ole enam deterministlikult funktsionaalne, vaid mitmene. Seetõttu on pikselleerimist laialdaselt peetud andmeid osaliselt hävitavaks tegevuseks. Piisavalt madalal eraldusvõimel hakkavad erinevate sõnede pildid kattuma, mistõttu on 100% täpsusega nende lugemine mitte ainult ebarealistlik, vaid absoluutselt välistatud. Ometi on võimalik teha haritud pakkumisi. Masinõppe rakendamine pikselleerimis-probleemi lahendamiseks aitab teha just seda. Sageli on olemas teatud eksimisruum, mis võimaldab teha valesid pakkumisi.

Kokkuvõtvalt:

- Pikselleeritud teksti lugemine 100% täpsusega vaid toore piksli-informatsiooni põhjal on teatud plokisuurusest alates matemaatiliselt võimatu.
- Pakkumiste tegemine on ikka võimalik, olgu need siis 50% või 99.9% täpsed. Pakkumisi on võimalik andmepõhiselt teha täpsemalt kui suvaliselt proovides.
- Reaalses maailmas on paljudes olukordades teatud tolerants vigade vastu, mis võimaldab rakendada ka ebatäiuslikke ja eksivaid meetodeid.

2.6.2 Logogrammid ja keerulised tähestikud

Hiina keeles on kasutusel logogramm-tähestikud, mis sisaldavad kümneid tuhandeid erinevaid märke. Need on visuaalselt detailsemad ja keerulisemad kui ladina tähed. See teeb nende tähistike lugemise väga suureks väljakutseks. Kui sõnes on tuhanded erinevad kirjamärgid, siis nii madalal eraldusvõimel ei ole võimalik märkidel vahet teha. Seetõttu on teatud keeled naturaalselt pikselleerimisnõrkuste vastu väga robustselt immuunsed. Antud töös ei hakata selliste tähestike peal teste läbi viima, sest see oleks tulemusetu.

2.6.3 Keeleliste mustrite rakendamine

Tuues sisse keelelise statistika on võimalik pakkumiste täpsust parandada, kui tegemist on sõnega, mis sisaldab sõnu või mustreid. Kui pikselleeritud pildi ründaja aga teab, et peidetud sõne on seosetu märgijada, ei ole tal mõtet selliseid meetodeid rakendada. Kui ründaja ei ole kindel, mis liiki peidetud sõne on, siis võib keelelise statistika rakendamine loetud tulemusi kallutada ja anda hoopis vale tulemuse. Kompromiss oleks kasutada mõlemat meetodit korraga. Programm nii väljastab toore pakkumise kui ka teise pakkumise, mille puhul on keelemustreid rakendatud.

2.7 Tulemuste valideerimine

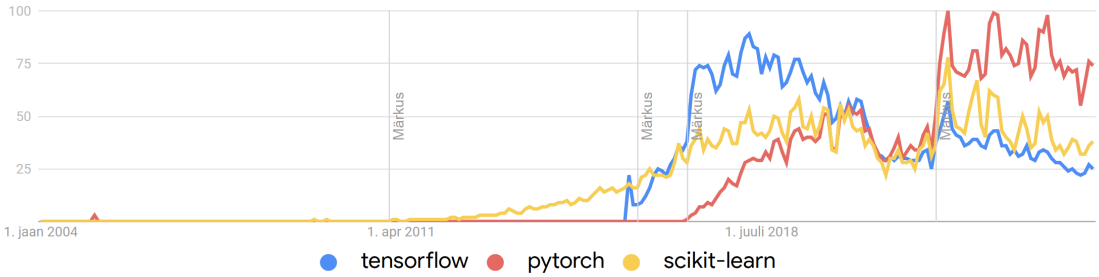
Mudeli efektiivsust hinnatakse järgmiselt:

- **Manuaalne valideerimine testandmestikul:** Kasutatakse sisendeid, mida mudel pole treeningu ajal näinud, et veenduda tarkvara võimes lahendada probleemi.
- **Automatiseeritud jõudlustest:** Mudel töötab läbi tuhandeid pilte ja programm koostab statistilise ülevaate, kus on antud protsentides keskmine *String Error Rate* (kui sageli teeb mudel sõnes vähemalt ühe vea) ja *Character Error Rate* (kui erinev on loetud sõne).

Need testid peegeldavad täpselt mudelite võimet pikselleeritud teksti lugeda. Mudelisse sisestatakse ka pilte, mis pole sünteetilised, vaid on kogutud reaalsetest olukordadest.

2.8 Masinõpperaamistiku valimine

Projekti elluviimiseks on tasuv rakendada masinõpperaamistikku. Populaarsed valikud on Scikit-learn, Pytorch ja TensorFlow. Scikit-learn on parem lihtsate mudelite jaoks, hea valik näiteks lineaarse regressioonanalüüsi jaoks. Pytorch on masinõppe valdkonnas väga levinud ning sellel on lai teekide ökosüsteem. Raamistik on sobivam sügavate närvivõrkude jaoks ja seda on lihtne kasutada. Tensorflow on mõistlik alternatiiv Pytorchile, kuid viimase paari aasta jooksul on see teisest raamistikust maha jäänud. Näiteks ei toetata versioonist 2.11 enam Windows-süsteemidel GPU-põhist riistavarakiirendust [11]. Graafikakaartidel treenimine on masinõppes, eriti süvamasinõppes, väga tähtis, sest see võimaldab tulemusteni jõuda palju kiiremini. Joonisel 1 on välja toodud raamistike relatiivne populaarsus. Antud andmete põhjal on sobivaim valik Pytorch.



Joonis 1. *Otsinguterminite populaarsus „Google Trends“ põhjal. [12]*

2.9 *ClearType* ja värvid

ClearType loodi eesmärgiga parandada monitoridel kuvatud tekstide loetavust. *ClearType* kasutab alampikslipõhist renderdamist, mis võib tekitada sümbolite äärtesse värvilisi moonutusi. Vaata joonist 2. [13]

Joonis 2. *Tekstide võrdlus. ClearType paremal. [13]*

Moonutused loovad värviinfo, mida võib olla võimalik masinõppes rakendada. Antud töö käigus valiti luua monokroomseid pilte töötlevad mudelid, sest see lihtsustab arendamist ning kiirendab mudelite treenimist. Lisaks võimaldab selline lähenemine üldistada ning lugeda tekste, mille renderamisel ei kasutatud *ClearType* tehnoloogiat. Tulevastes teadustöödes võib olla kasulik uurida, kas erinevate alampikslipõhiste renderdusmootorite puhul on võimalik lekkivate värvitoonide põhjal tuvastustäpsust märgatavalt parandada.

Värviinfo võib olla kasulik ka siis, kui tekstil on värviline piirjoon või tekst ise on täidetud mingisuguse mustri või materjal-efektiga. Antud töös käsitletakse lihtsalt renderdatud teksti ning ei uurita kõikvõimalikke eriefekte.

3. Mudelite treenimine

Treenimiseks on kasutatud alati sama riistvara: Ryzen 5900X ja RTX 3080 Ti. Teegiks on Pytorch, mis on konfigureeritud töötama graafikakaardil. Andmed laetakse kaardile treeningu käigus jooksvalt *batchide* kaupa. Analüüsi põhjal otsustati luua mudelid, mis loevad mustvalgeid pilte.

3.1 Treenimistingimused ja lähteandmed

Selles alampeatükkis kirjeldatakse esmase treeningprotsessi tingimusi. Samade andmete peal treeniti mitu mudelit, et näha, milline lähenemisviis on tulemuslikum. Esimene mudel on modifitseeritud variant tavalisest CAPTCHA OCR mudelist [14]. Muud mudelid on selle baasmudeli variatsioonid.

2016. aasta teadustöös „*Defeating Image Obfuscation with Deep Learning*“ [15] kasutati närvivõrgul põhinevat mudelit pikselleeritud käsikirja lugemiseks. Viidatud teadustöös kasutatakse piiratud klassidega andmestikku, mis koosneb vaid kümnest numbrist. Seega on suvaline pakkumine õige 10% täpsusega. Lisaks on tegu eraldi piltidega üksikutest sümbolitest, mitte jadadest.

Jadade lugemine on keerulisem, sest erinevad märgid sulanduvad pikselleerides kokku ja varjavad informatsiooni, mida saaks kasutada nende tuvastamiseks. Antud töö treeningandmestikus on klasse rohkem: 62 + 1. Sümboliteks on korraga „ABCDEFGHIJKLMNOPQRSTUVWXYZ“, „abcdefghijklmnopqrstuvwxyz“, „0123456789“ ja CTC tühimärk, mida kasutatakse sõnes tühjuse tähistamiseks ning tähekorduste vältimiseks. Sellisel juhul on suvaline pakkumine õige vaid 1.59% juhtudest, 20 sümboli korrektse ennustamise tõenäosus aga umbes üks undetsillionist (10^{-36}).

Iga sümbol tokeniseeritakse ehk asendatakse antud juhul täisarvuga. Tokeniseerimise faasis tekib ka tabel, mis kaardistab tokenid tagasi sümboliteks. Mudeli väljundiks on tokenid, mis enne kasutajale kuvamist tõlgitakse tabeli abil sümboliteks. Tühi token (CTC

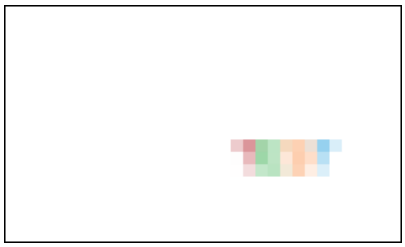
blank) saab indeksi 0, peale seda saab iga sümbol ühe võrra suurema indeksi. Eesmärk on vältida probleeme sisendandmetes esinevate sümbolitega. Näiteks kui tühja tokeni esindamiseks kasutatakse sümbolit ∅, siis treenides sõnel, mis sisaldab sedasama sümbolit, tekiks konflikt. See ajaks mudeli segadusse ja langetaks edukust. Seega on tähtis kõik sümbolid ümber tokeniseerida täisarvudeks.

Treenimine viiakse läbi piltidel, mis sünteesitakse eraldiseisva Püütoni programmi abil. Esialgseks testimiseks on sõned valitud mitmel viisil: Ingliskeelsed sõnad (466 550) [16], kõikvõimalikud sõned pikkusega 3 (238 328), suvalised jadad (200 000), populaarseimad paroolid (100 000) [17], populaarseimad kasutajanimed (50 000) [17]. Kokku on pilte liitandmestikus 1 054 878. Sellise andmestiku eesmärk on peegeldada reaalsetes olukordades esinevaid andmeid ja katta kõikvõimalikud üleminekud sümbolite vahel. Kuna andmestikud on suured, eraldatakse validatsiooni jaoks vaid 1% andmetest.

Suvaliselt genereeritud sõned luuakse kindla seemne alusel, mistõttu pole ühed andmestikud rohkem kallutatud kui teised. Suvaliselt genereeritud sõnede pikkuseks on 1 kuni 20 märki, sest see katab enamus pärisandmetes leitavatest sõnedest, jäädes samas lühikeseks, et kiirendada treenimist. Pikemate sõnede lugemiseks tuleb tekst mitmeks tükiks lõigata ning ükshaaval mudelisse sisestada. Piltide genereerimine toimus järgmiselt: Loodi valge taust dimensioonidega 512 x 64 pikslit. Fondi suurus valiti iga sõne puhul suvaliselt vahemikus 24 kuni 28pt. Font oli Arial. Pilt pikselleeriti *block average* algoritmiga, kus ploki suurus oli 8 pikslit. Plokinihe valiti suvaliselt vahemikus 0–8, et katta kõikvõimalikud paigutused. Sarnase efekti saavutamiseks kasutati ka pildi äärest polsterdamist (*padding*) 0–20 piksli ulatuses. Lisas 2 on välja toodud mudeli kood.

3.2 Sisendandmete eeltöötlus

Pildid standardiseeritakse, et vähendada varieeruvust. Programm laeb pikslite heledused normaliseeritud kujul vahemikus 0 kuni 1. Joonisel 3 on esitatud ülevaade esimestest eeltöötlusetappidest.

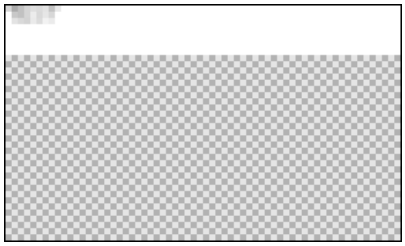


Originaalne sisendpilt. Näites on iga sümbol erinevat värvi.

Pilt konverditakse monokroomseks, et parandada jõudlust ja generaliseerimist. Nii on igal pikslil vaid heledusväärtus vahemikus 0–255.

Tuvastatakse andmeid sisaldav ala ja tühi ümbrus kärbitakse pildilt. Ruuduline muster esindab kärbitud ala, kus iga ruut vastab ühele pikslile.

Tuvastatakse plokisuurus ja pildi dimensioone vähendatakse, kuni üks plokk on üks piksel suur.



Pilt laiendatakse ühiksuurusele 64x8. Täiteks lisatakse pildist paremale ja alla valgeid piksleid. Standardiseeritud paigutus lihtsustab üldistamist.

Joonis 3. *Eeltöötlusprotsessi etapid.*

3.3 Esialgsed tulemused

Sama baasmudeli nullist treenimine kolmel eri katsel tõi varieeruvad tulemused (tabel 1). Mõõdetud SRR valim oli 10 000 ja CRR valim 100 000.

Tabel 1. *Mudelite täpsused.*

Mudel	SRR	CRR	SER	CER
Baasmudel katse 1	77.93%	94.39%	22.07%	5.61%
Baasmudel katse 2	78.59%	94.58%	21.41%	5.42%
Baasmudel katse 3	81.23%	95.25%	18.77%	4.75%
Leaky ReLU	79.32%	94.78%	20.68%	5.22%
3-kihiline LSTM 2 x param.	88.71%	96.97%	11.29%	3.03%
Dropout 0.04	78.69%	94.63%	21.31%	5.37%
Dropout 0.5	76.25%	93.90%	23.75%	6.10%

Tabel 2. *Mudelite täpsused vaid numbrijadadel treenides.*

Mudel	SRR	CRR	SER	CER
Baasmudel	98.66%	99.67%	1.34%	0.33%
3-kihiline LSTM 2 x param.	98.77%	99.68%	1.23%	0.32%

Kui treenida mudelid vaid numbrijadade peal, on täpsus palju kõrgem tänu klasside madalale arvule (tabel 2). Ainult numbrite tuvastamisel on väljund kitsas (vaid 10 klassi + CTC tühimärk). Tänu sellele suudab mudel saavutada kõrge täpsuse ning ületab märkimisväärselt üld-otstarbeliste mudelite tulemusi. Baasmudelite keskmine tulemus oli 95% usaldusvahemikuga järgnev:

SRR: 79.25% (veamäär 4.35%), **CRR:** 94.74% (veamäär 1.12%)

Veamäär näitab tulemuste kõikumist. Veamäär on välja toodud SSR ja CRR absoluutse väärtusena, mitte relatiivse protsendina. *Leaky ReLU* andis veidi paremaid tulemusi, kuid see jääb täielikult veapiiri sisse. Vajalik on suurem valimimaht. Topeltarvu parameetritega kolmekihiline LSTM-mudel toimis standardmudelist märkimisväärselt paremini, kuid lisakulu oli pikem treeninguaeg. SRR protsent tõusis 9.46 võrra, CRR protsent 2.19 võrra. Veatult ennustatud sõnede arv tõusis rohkem, sest mudel suutis täpsemalt ennustada viimaseid valesti määratud märke. Keskmine märkide ennustustäpsus aga nii oluliselt ei

kasvanud, sest eriti keerulised märgid nagu väike „l” ja suur „I” jäid mudelile jätkuvalt probleemseks omavahelise visuaalse sarnasuse tõttu.

Mudeleid treeniti 100 epohhi. Baasmudeli treeningaeg oli ligikaudu 350–410 minutit, samas kui keerukama kolmekihilise mudeli treenimine võttis umbes 950 minutit. Rohkem kui kahekordne treeningaja kasv peaaegu poole võrra väiksema sõnede veamäära (SER) saavutamiseks on suhteliselt hea kompromiss. Kolmekihiline mudel näitas aga palju väiksemat täpsuse kasvu numbrite andmestiku peal. Ilmselt olid andmed piisavalt eristatavad ka lihtsama mudeli abil. Rohkemate klasside puhul oli aga võimalik kasu lõigata selle keerulisemast arhitektuurist. Lisad 4, 5 ja 6 visualiseerivad treenimise kulgu.

Mis siis kui madaldada *dropout* väärtust? Muutes baasmudeli väärtuse 0.2 hoopis 0.04 peale toimus kerge ülesobitamine, validatsioonikadu tõusis. Kõrge *dropout* andis samuti halbu tulemusi. Täpsem hüperparameetrite seadistamine võib võrreldes baasmudeliga tulemusi parandada, testimine vajab hinnanguliselt 50 tundi treeningaega.

3.3.1 Mudelite hinnaefektiivsus

Antud süsteemis ühte mudelit korraga treenides on kogu energiakulu suurusjärgus 300W, 100 epohhi jaoks kulub umbes 400 minutit. Kokku kulub 2kWh. 2025. aasta keskmise turuhinnaga 0.14€/kWh on baasmudeli treenimise hind 0.28€ [18]. Mudel saavutab ka 5 epohhi treenimisega üle 85% keskmise sümboli ennustamise täpsuse. Sellisel juhul kulub 20 minutit ja 0.1kWh. Hinnaks kujuneb 0.014 eurot. Nii aja- kui hinnaefektiivsust annab tõsta samal riistvaral mitme mudeli paralleelse treenimisega. Keerulisem 3-kihiline LSTM mudel kasutab ära rohkem graafikakaardi CUDA tuumadest. Keerulisemat mudelit treenides suureneb kogu süsteemi voolutarve 450W peale.

Mudelite treenimine on relatiivselt odav, isegi kui arvesse võtta, et kõrgeima täpsuse saavutamiseks tuleb treenida palju erinevaid spetsiifilisi mudeleid. Kui luua põhjalik mudelikogu, kus on 10 fonti, 5 erinevat fondisuurust, 16 plokisuurust, 5 sõneliiki (numbrid, väiketähed, suurtähed, tähed koos, kõik koos), on kokku variatsioone 4000. Sellisel juhul on mudelite treenimise kulu vahemikus 56–1120 eurot sõltuvalt kasutatud epohhide arvust (5–100). Tuleb rõhutada, et tegu on pigem hinnalaega. Arvutus eeldab, et kasutatakse ebaefektiivset

tarbijariistvara, mudeleid ei treenita paralleelselt ning voolutarbe eest makstakse keskmise elektrihinnaga. Tegelik hind võib olla suurprojektides (näiteks luureagentuurides/sõjaväes) kümneid kordi odavam. Inferents on kiire ning seega samuti odav, testide põhjal kulub ühe pildi jaoks vähem kui 30 millisekundit.

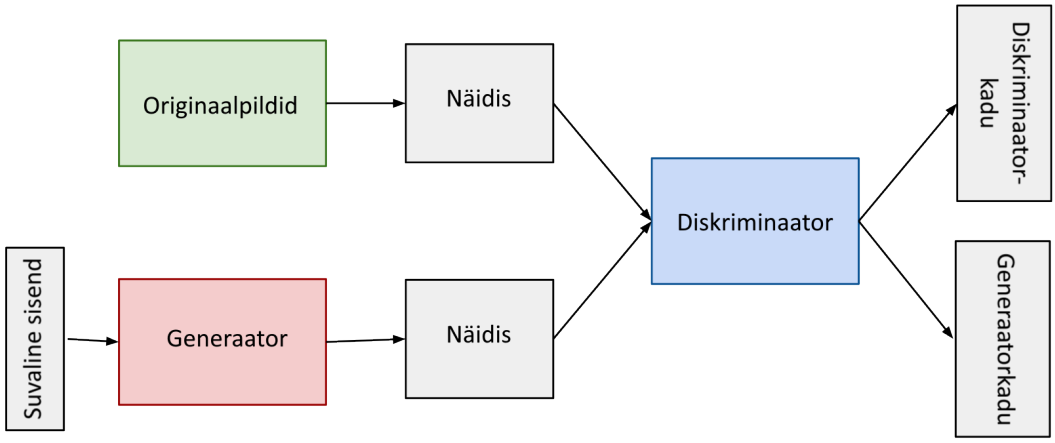
3.4 GAN-põhised superresolutsioonimudelid

GAN ehk *Generative Adversarial Network* on masinõppemeetod, mis loob kaks võistlevat närvivõrku:

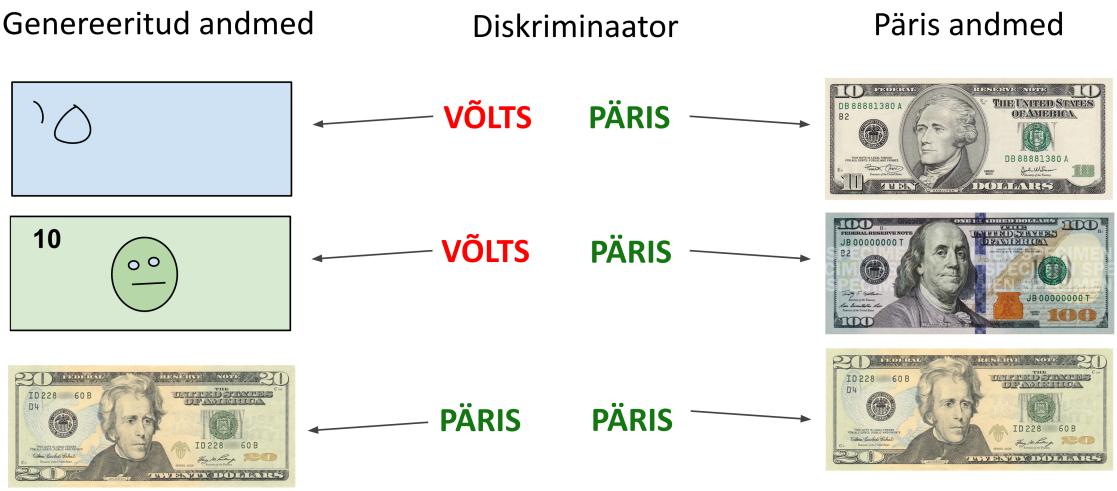
Generaator – Sünteesib väljundi, mis on võimalikult sarnane treeningandmetele.

Diskriminaator – Üritab teha vahet generaatori väljundil ja tõestel treeningandmetel.

Kahte närvivõrku treenitakse korraga nullsummamängus, kus mõlemad üritavad üksteist alistada. Generaator üritab luua tõelähedasi väljundeid ja diskriminaator püüab neid eristada. Kui treenitud diskriminaator ei suuda enam vahet teha päris andmetel ja generaatori väljundil, siis generaator on edukalt saavutanud masinõppe eesmärgi ja võidab. Joonised 4 ja 5 aitavad kontsepti visualiseerida. [19]



Joonis 4. *GAN struktuur. [19]*



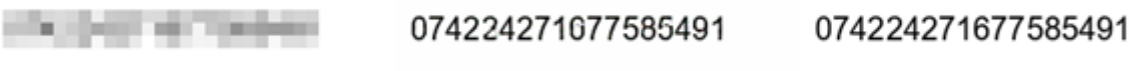
Joonis 5. *GAN-põhise pildigenereerija treenimisprotsessi illustratsioon. [19]*

3.4.1 Mudelite arhitektuur

Superresolutsioonimudelid üritavad madala eraldusvõimega pilte skaleerida suuremale eraldusvõimele, seejuures parandades pildi hoomatavat selgust. Diskriminaatoriks on viiekihiline konvolutsiooniline võrk (CNN). Generaator skaleerib 32x32px sisendpildi kolme sammuga dimensioonidele 256x256px. Kood on välja toodud lisas 3.

3.4.2 GAN-mudeli treenimise tulemused

Mudel püüdis pikselleeritud numbreid sisaldavatest piltidest konstrueerida originaalpilte. Sisend ja väljundpiltide eraldusvõimete vahe on kaheksakordne. Esmasel testimisel annab GAN mudel üllatavalt häid tulemusi. Tuleb meeles pidada, et andmestik sisaldab vaid kümmet klassi, tähestikul treenitud mudeli täpsus võib olla palju madalam. Antud joonistel 6 on sisendpilt vasakul, mudeli ennustus keskel ning tõene originaalpilt paremal.



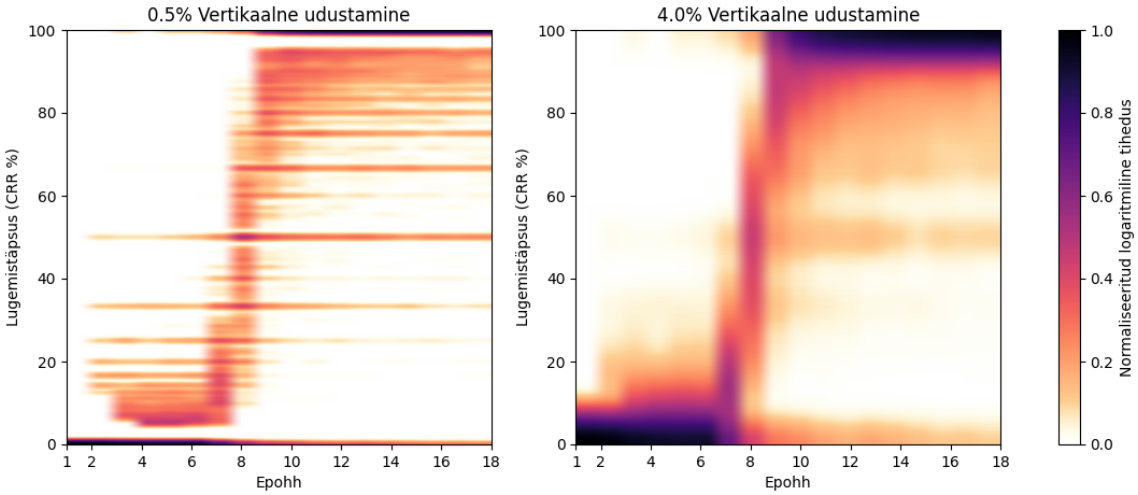
Joonis 6. *Pikselleeritud pildi, superresolutsioonimudeli väljundi ja lähtepildi võrdlus.*

3.5 Eksperimendid LSTM mudelitega

Järgnevates testides on mudelid treenitud inglise tähestiku suurtähtedel. See vähendab treenimisaega, sest klasse on 62 asemel 27. Paremate tulemuste jaoks kasutatakse originaalse baasmudeli asemel nüüdsest kolmekihilist LSTM mudelit, mille LSTM kihtide parameetrid on kahekordistatud. Antud mudeli SER ja CER olid teistest mudelitest palju madalamad. Erinevate plokisuuruste kohta on koostatud mudelite eksimismaatriksid. Eksimismaatriks (Lisa 7) näitab, mis tähtede osas mudel eksib. On näha, et mudel teeb 8-pikslise plokisuuruse puhul väga vähe eksimusi ning töötab hästi. Lisad 8 ja 9 näitavad, kuidas kõrgemal plokisuurusel treenitud mudelid eksivad tähtede osas rohkem.

3.5.1 Mudeli treenimist illustreeriv visuaal

Kuidas näeb välja mudeli õppimisprotsess? Selle visualiseerimiseks võib koguda mudeli ennustusi iga epohhi lõpus ning koostada joonise. Joonisel 7 näitavad tumedad värvid ennustuste tihedat paiknemist, heledad toonid hõredat paiknemist. Plokisuurus oli 8 pikslit, andmestiku suurus 50 000 pilti. Antud juhul sai mudel epohhide 7–9 jooksul lõpuks kaalud loogika järgi paika ning selle tulemusena kasvas mudeli täpsus hüppeliselt. Joonisel võib märgata domineerivaid horisontaaljooni täpsustel 50% ja 33%. Need tulenevad asjaolust, et paljude sõnede pikkused jaguvad kahe ja kolmega. Graafidel on kasutatud vertikaalset udustamist, et anda üldisem ülevaade muutustest.



Joonis 7. *Mudeli pakkumiste täpsuse jaotus üle epohhide.*

3.5.2 Programmide pikselleerimisalgoritmid erinevad

Erinevad programmid võivad pikselleerimis-efekte rakendada erinevate algoritmide abil. Photoshopis on tavaline *block average* efekt leitav menüüst: *Filter* > *Pixelate* > *Mosaic*. Programmis „Paint.NET“ on sarnane efekt leitav menüüst: *Effects* > *Distort* > *Pixelate*. Ometi ei kasuta see *block average* algoritmi, vaid mingit muud algoritmi. Seetõttu ei anna kindal algoritmil treenitud mudelid „Paint.NET“ abil pikselleeritud teksti kohta ühtegi õiget pakkumist. Visuaalse hinnangu põhjal tundub kasutatav algoritm olema agressiivsem, teisisõnu kontrastsem. Halltoonide arv väljundpiltidel on madalam. Sellisel algoritmil on mudelit raskem treenida, sest see on originaalandmete suhtes hävituslikum. Võimalik, et kõrge plokisuuruse puhul on üldistavate mudelite edukas treenimine isegi võimatu. Edukas lähenemisviis võib olla muutujaid kõrvaldades väga spetsiifiline mudel treenida. Näiteks treenides mudeleid andmestikul, millel on kindel font, fondisuurus, renderdus-stiil, plokinihe ja kitsas sümbolite valik. Andmete sünteesimine on aga keerulisem, kui ei ole teada, kuidas täpselt tarkvara pikselleerimist teostab.

„Paint.NET“ ei ole enam avatud lähtekoodiga, kuid vanasti avalikustati seda MIT litsentsi all. Versioon 3.36.7 on GitHubi keskkonnas saadaval. Failis *src/Effects/PixelateEffect.cs* on viide funktsioonile *ColorBgra.BlendColors4W16IP*. Funktsioon on deklareeritud failis *src/Core/ColorBgra.cs*. Pikselleerimisefekt ei leia mitte pikslite keskmist värvi, vaid teeb jõudluskaalutlustel sohki: valib vaid neli pikslit ploki nurkadest. Koodi lugemiseks vaata joonist 8. Koodi on muudetud loetavuse huvides. [20]

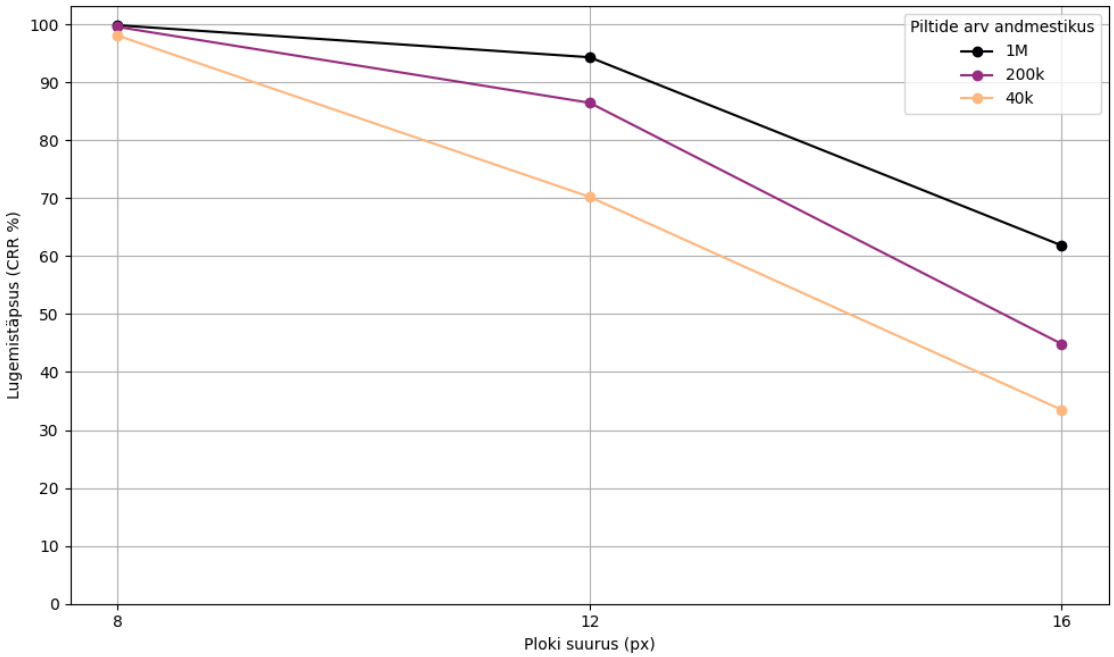
```
private ColorBgra ComputeCellColor(int x, int y,
    RenderArgs src, int cellSize) {
    Rectangle cell = GetCellBox(x, y, cellSize);
    cell.Intersect(src.Bounds);
    int left = cell.Left;
    int right = cell.Right - 1;
    int bottom = cell.Bottom - 1;
    int top = cell.Top;
    ColorBgra topLeft = src.Surface[left, top];
    ColorBgra topRight = src.Surface[right, top];
    ColorBgra bottomLeft = src.Surface[left, bottom];
    ColorBgra bottomRight = src.Surface[right, bottom];
    ColorBgra c = ColorBgra.BlendColors4W16IP(
        topLeft, 16384, topRight, 16384,
        bottomLeft, 16384, bottomRight, 16384);
    return c;
}

/// 4W16IP "4 colors, weights, 16-bit int precision"
public static ColorBgra BlendColors4W16IP(
    ColorBgra c1, uint w1, ColorBgra c2, uint w2,
    ColorBgra c3, uint w3, ColorBgra c4, uint w4) {
    const uint ww = 32768;
    uint af = (c1.A * w1) + (c2.A * w2)
        + (c3.A * w3) + (c4.A * w4);
    uint a = (af + ww) >> 16;
    uint b; uint g; uint r;
    if (a == 0) {
        b = 0; g = 0; r = 0;
    } else {
        b = (uint)((((long)c1.A * c1.B * w1)
            + ((long)c2.A * c2.B * w2)
            + ((long)c3.A * c3.B * w3)
            + ((long)c4.A * c4.B * w4)) / af);
        g = (uint)((((long)c1.A * c1.G * w1)
            + ((long)c2.A * c2.G * w2)
            + ((long)c3.A * c3.G * w3)
            + ((long)c4.A * c4.G * w4)) / af);
        r = (uint)((((long)c1.A * c1.R * w1)
            + ((long)c2.A * c2.R * w2)
            + ((long)c3.A * c3.R * w3)
            + ((long)c4.A * c4.R * w4)) / af);
    }
    return ColorBgra.FromBgra((byte)b, (byte)g,
        (byte)r, (byte)a);
}
```

Joonis 8. *Paint.NET lähtekoodis arvutatakse vaid nurkades asuvate värvide keskmine.*

3.5.3 Plokisuuruse ja andmestiku suuruse mõju lugemistäpsusele

Jooniselt 9 on näha, et suurenev plokisuurus vajab täpsuseks aina suuremat andmestikku. Miljonite piltide genereerimine kindla rakenduse abil on palju aeganõudvam ja keerulisem ülesanne kui algoritmiline sünteesimine.



Joonis 9. *Plokisuuruse ja andmestiku suuruse mõju täpsusele.*

Joonisel 9 kujutatud mudelite kumulatiivne treenimisaeg oli 88 tundi. Mudeleid treeniti 100 epohhi. Spearmani korrelatsioonikordaja on andmestiku suuruse ja CRR vahel 0.316, plokisuuruse ja CRR vahel -0.949. Andmestiku suuruse ja CRR vahel on nõrk positiivne korrelatsioon. Plokisuuruse ja CRR vahel on väga tugev negatiivne korrelatsioon. Mudeleid treenitakse jagades andmestiku treening- ja validatsiooniandmestikuks. Suvaliselt andmeid sünteesides ei ole ühtegi reeglit, mis keelaks duplikaatsõnede tekkimist. Duplikaatsõnede teke on tahtlik, sest pildil on peale sõne sisu ka teised muutujad. Sünteesprogramm muudab näiteks fondi suurust ja plokinihet. Kui andmestiku suurus kasvab, siis kasvab ka kahe andmestiku kattuvus. See rikub võimet hinnata mudeli üldistusvõimet, kuid ei takista mudeli jõudluse hindamist. Lõppkasutaja jaoks ei oma tähtsust, mis sõnedel mudelit treeniti, vaid kas mudel väljastab tema sisendi põhjal õige vastuse. On tõenäoline, et CRR kasvab jätkuvalt, kui suurendada andmestike suurust veelgi, kuid mõju väheneb.

3.5.4 *Batchi* suuruse mõju

Batchi suurus võib mõjutada mudeli jõudlust ja täpsust. Suuremad *batchid* võimaldavad mudelit treenida jõudlusefektiivsemalt, kuid seda täpsuse arvelt. Korraga vaadeldes vähem pilte, õpivad mudelid paremini detaile tundma. Mudeli kaalud muutuvad iga *batchi* lõpus. Väiksemate *batchide* puhul muutuvad kaalud sagedamini. Pole teada kui tugev selle mõju antud mudeli täpsusele on. Seetõttu viiakse läbi testid, et leida kompromiss treeningaja ja mudeli täpsuse vahel. Kasutatakse väärtuseid 1, 4, 8, 32, 128, 512. Vaata tabelit 3.

Tabel 3. *Batchi suuruse mõju mudeli täpsusele ja treeningajale.*

Batchi suurus	SRR	CRR	Treenimisaeg
1	0%	0%	5390 minutit
4	0.20%	3.37%	1623 minutit
8	61.25%	87.17%	890 minutit
32	48.45%	82.89%	364 minutit
128	47.25%	82.02%	220 minutit
512	43.00%	80.39%	216 minutit

Tulemused näitavad, et mõju mudelite täpsusele oli märkimisväärne. Kõige ajaefektiivsem on treenida 128 pildi kaupa, misjuhul CRR tõusis 100 epohhi jooksul keskmiselt 0.3728% minutis. Kõige täpsem mudel tuli treenides 8 pildi kaupa, kuid CRR tõusis vaid 0.0979% minutis. Seega leppides 5.15% võrra madalama CRR tulemusega on võimalik kiirendada treenimisprotsessi 3,8 korda. Kui *batchi* suurus oli 4 või 1, siis ei suutnud mudel kaale vastavalt nähtud mustritele korrastada. *Learning rate* on hüperparameeter, mis defineerib kui palju mudel võib korraga oma kaale muuta. Aja jooksul vähenes *learning rate* automaatselt ning mudel ei saanud enam piisavalt kaale kohandada. On spekuleeritav, et teistsugune *learning rate* võib selle probleemi lahendada ja viia paremate tulemusteni. Tulevikus võib rakendada etapilist treenimist, kus sama mudelit treenitakse näiteks esimesed 40 epohhi 128-pildiste pakkidega, seejärel 40 epohhi 8-pildiste pakkidega ja lõpuks 20 epohhi 4-pildiste pakkidega. See tõstaks tõenäoliselt nii treenimisprotsessi kiirust kui ka mudeli täpsust võrreldes katsetes saadud tulemustega.

4. Kontrollermudelid ja spetsialistmudelite valimine

Kontrollermudelite ülesanne on valida iga olukorra jaoks parim masinõppemudel. Spetsifilistel andmetel treenitud mudelid on täpsemad kui üldistavad mudelid, seda kinnitasid ka tulemused peatükis 3.3.

4.1 Kontrollermudelite rakendamine ja konsensusmehhanismid

Kui iga sisendpildi puhul on võimalik kindlaks teha selle liik (näiteks numbrijada/tähejada), siis tõuseks lugemistäpsus märkimisväärselt. Peamine probleem selle strateegia rakendamisel on aga kontrollermudeli enda eksimisrisk. Kui kontrollermudel annab tähti sisaldava sõne numbritega tegelevale lugemismudelile, siis pole lootustki saada korrektne väljund.

Kontrollermudelite kasutamist võib vaadelda kui kõrge riskiga ja kõrge tasuvusega lähenemist. Kui kontrolleril on õigus, on väljundiks väga täpne tulemus; kui kontroller eksib, siis rikume tulemuse võrreldes üldistava mudeliga täielikult. Siinkohal võib olla kasu konsensusmehhanismidest eri mudelite vahel, mis võivad vähendada eksimisriski. Näiteks antakse sisendpilt neljale erinevale mudelile: kontroller, numbrilugeja, tähelugeja, üldistaja (tunneb mõlemaid jadasid). Üldistajamudeli kindlusmaatriks korrutatakse läbi numbri- ja tähelugeja maatriksitega, et saada täpsem tulemus. Kui kontrollermudeli ennustus kattub saadud maatriksiga, siis on väga tõenäoline, et valitakse õige mudel.

Kolme mudeli konsensust kontrollermudelita võib samuti kasutada, kuid kontroller lisab kindlust tulemuse täpsuses ning aitab paremini otsuseid langetada olukordades, kus vaatleme ka sõneklassi, milles sisalduvad numbrid ja tähed segamini. Ainuke reaalne puudus on vajaliku arvutusjõu kasv nii treenimisel kui ka lugemisfaasis. Lugemisfaasis ei tohiks see probleeme tekitada, arvestades et ühe pildi lugemise jaoks kulub mudelil sekundeid või millisekundeid. Treenimisfaasis on vaja treenida üldistaja ning kontrollermudelid, mis on lisakulu. Ometi on võimalik tarbijariistvaral praktiliselt kasulik mudelikogu treenida vaid ühe ööpäevaga. Seega on konsensussüsteemi kasutamine mõistlik ja hea kompromiss.

4.2 Kontrollermudeli teooria valideerimine

Treeniti binaarselt klassifitseeriv kontrollermudel, mille andmestikuks oli:

- 1 miljon pilti suurtähtedest ja numbritest eraldi sõnedes

Treeniti kolm eraldi mudelit, mille andmestikeks olid:

- 1 miljon pilti suurtähtedest (tähtedele spetsialiseerunud mudel)
- 1 miljon pilti numbritest (numbritele spetsialiseerunud mudel)
- 1 miljon pilti suurtähtedest ja numbritest eraldi sõnedes (üldistav mudel)

Kõikide piltide puhul kasutati plokisuurust 16 pikslit. Peale treenimist arvutati, kas arvestades sisse kontrollermudeli mõju paranes lugemistäpsus võrreldes üldistava mudeliga. Kontrollermudeli täpsus on 81.02%. Eeldades numbriliste ja tähti sisaldavate sisendsõnede võrdset esinemist on kontrollersüsteemi täpsus 56.29%. See on 6.34% võrra kõrgem kui üldistaval mudelil. Seega tõestab kontrollermudeliga süsteem võimekust anda täpsemaid tulemusi ja seda isegi karmides tingimustes (plokisuurus 16px). Vaata tabelit 4.

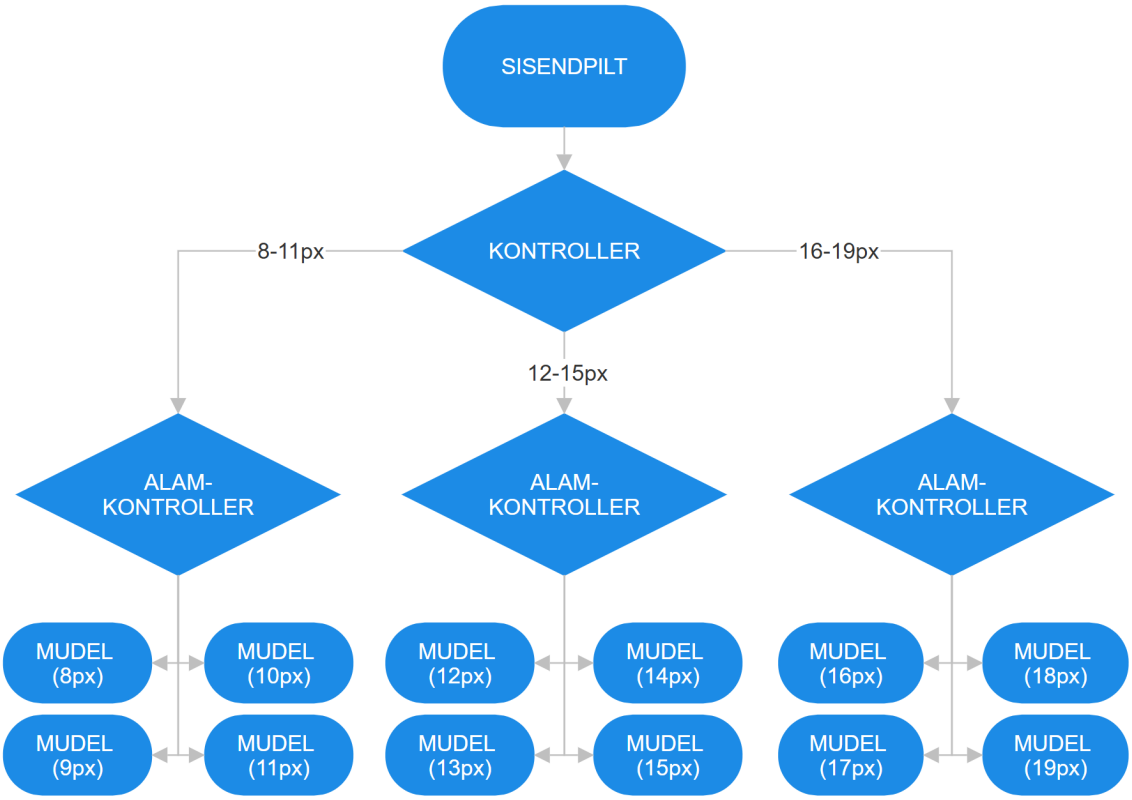
Tabel 4. *Lugemistäpsuste võrdlus.*

Mudel/Süsteem	**CRR**
Tähejadadel treenitud mudel	61.70%
Numbrijadadel treenitud mudel	77.26%
Tähejadadel ja numbrijadadel treenitud mudel	49.95%
Kontrollersüsteem + 2 eraldi treenitud mudelit	56.29%

4.3 Kontroller-arhitektuuri edasine arendus

Kontrollermudelid tõestasid eelmises peatükis võimekust lugemistäpsust parandada. Põhjaliku kontrollersüsteemi arendamiseks pakutakse selles töös välja uurida otsuspuu-põhiseid lahendusi. Iga mudel teeb ühe otsuse, näiteks fondimudel valib fondi, plokisuurusmudel valib plokisuuruse. Otsuseid on võimalik teha paralleelselt, kuid efektiivne võib olla ka mitmeastmeline jadaprotsess, kus eelmiste valikute põhjal valitakse rohkem spetsialiseerunud kontroller järgmise valiku jaoks.

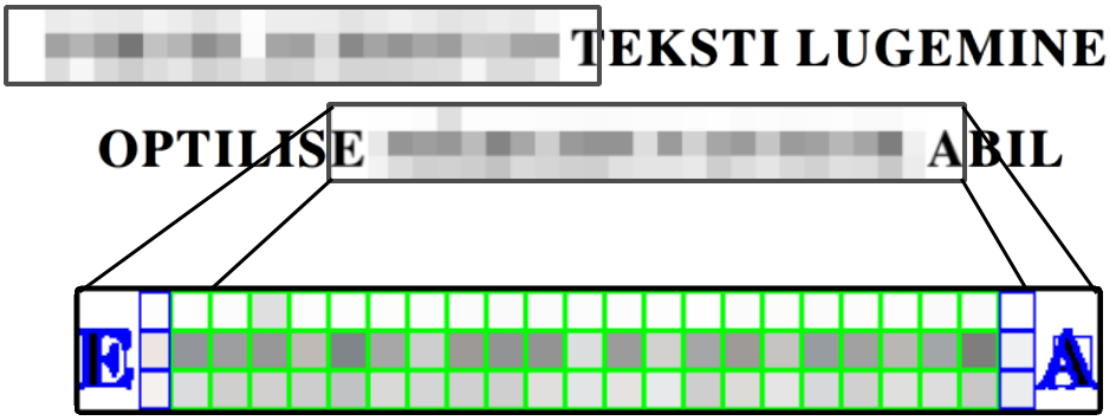
Plokisuurust valiv mudel teeb sisendpildi põhjal otsuse: 8–11px, 12–15px või 16–19px. Näiteks tehakse otsus 12–15px. Seejärel valib sellele vahemikule spetsialiseerunud mudel täpse plokisuuruse, näiteks 13px. Selline mitmekihiline protsess võib anda täpsemaid tulemusi kui ühetasandiline klassifikatsioon. Iga alamkontroller on treenitud enda kitsa vastutusala jaoks ning kokku tehakse seetõttu tõenäoliselt täpsem otsus. Vaata joonist 10.



Joonis 10. *Kahekihiline kontrollersüsteem valib plokisuuruse.*

4.4 Otsast-otsani automatiseeritud töövoog

Joonisel 11 on rakendatud väga algelist eeltöötlussüsteemi, et programm suudaks otse ekraanitõmmiseid lugeda. Sellise süsteemi järjekindlaks ning mugavaks muutmine on ajamahukas. Seetõttu seda töös enam peale kontsepti tõestava prototüübi ei käsitleta.



Joonis 11. *Algoritm otsib pikselleeritud alad, eraldab need ja analüüsib plokke.*

Edasiarenduse käigus on võimalik luua tarkvara, mis saab sisendiks videovoo ja kuvab tulemusi reaalajas. Programm tuvastaks pikselleeritud alad, teeks eeltöötluse, inferentsi ning kirjutaks pikselleeritud tekstid väljundvoos üle loetud tulemustega.

5. Tulenevad privaatsustagajärjed

Hiljutises infoühiskonna ajaloos hakati piksellatsiooni kasutama tähtsa privaatsustööriistana, mis eemaldab tundliku sisu piltidelt ja videomeedialt. Seda kasutatakse, et kaitsta indiviidide indentiteete ja tagada, et tundlik info on konfidentsiaalne ja loetamatu. Digitaalkommunikatsiooni ja sotsiaalmeedia kontekstis on pikselleerimine kasutusel, et vältida kõrvalistel osapooltel ligi pääseda isikuandmetele, seeläbi säilitades inimeste anonüümsust olukordades, kus privaatsus on esmatähtis. [21]

Edusammud masinõppes on aga viinud keeruliste optiliste märgituvastusmudelite tõusuni. Need suudavad tõlgendada ja isegi taasluua teksti tugevalt pikselleeritud piltidelt. Sellised süsteemid kasutavad närvivõrgu arhitektuure, mis sisaldavad konvolutsioonilisi kihte, rekurrentseid võrke (näiteks LSTM) või generatiivseid võistlevaid võrke (GAN). Generatiivsed võrgud võivad kasutada superresolutsiooni meetodeid, et taasluua detaile, mida kunagi peeti täiesti kadunuks. Kuigi sellised tehnoloogilised innovatsioonid pakuvad märkimisväärset kasu andmetele juurdepääsetavuse ja taastamise valdkondades, avavad need ka ukse potentsiaalsele väärkasutamisele. [6], [9]

Üks põhimure on volitamata andmete taastamine ja isikute taas-tuvastamise risk. Tundlik teave, sealhulgas unikaalsed isikuandmed, konfidentsiaalsed sõnumid või omandiõigusega kaitstud andmed, mis on pikselleerimisega tahtlikult varjatud, võidakse nüüd vältimatult paljastada. Pahatahtlikud osapooled võivad neid pikselleerimisvastaseid tehnikaid ära kasutada varjatud andmete eraldamiseks, kahjustades sellega üksikisikute ja organisatsioonide privaatsust. [21]

Võimekus lugeda pikselleeritud pilte kutsub välja avalikkuses laialtlevinud eelduse, et piksellatsioon on robustne ja turvaline viis andmete anonümiseerimiseks. Ajastul, mil masinõppemudelid jätkuvalt paranevad, tuleb pikselleerimise „ohutu“ lävi uuesti ümber hinnata. See areng tõstatab küsimuse: Kas kehtivad privaatsusseadused ja andmekaitsestandardid on piisavad, et seista vastu pikselleerimisvastaste tööriistade võimekusele? Paljudes asutustes ei pruugi kehtida reeglid, mis takistaksid pikselleerimist. [21]

Lisaks andmete taaslugemise ohule on depikseleerimistehnoloogia väärkasutamise potentsiaal suurem. Näiteks võivad valitsused või eraühingud kasutada selliseid vahendeid massiliseks jälgimiseks, mis võib rikkuda kodanikuvabadusi. Andmelekete korral võidakse varastatud pikselleeritud pilte töödelda tundliku sisu paljastamiseks, laiendades sellega andmeleket. Veelgi enam, kui lai avalikkus on teadlik, et piksellatsiooni saab ka tagurpidi töötleda, võib juhtuda, et usaldus digitaalse meedia vastu väheneb, mistõttu inimesed kõhklevad enne teabe jagamist rohkem, isegi kui see on tegelikult turvaliselt anonüümseks muudetud.

Nende privaatsusprobleemide lahendamiseks peavad arenema seaduslikult reguleerivad raamistikud ja head tavad. Vaja on rangemaid andmete anonüümseks muutmise standardeid, tagamaks et andme-eemaldusmeetodid jäävad masinõppe-tehnikate vastu tõhusaks. Läbipaistvus mängib riskide maandamisel olulist rolli. Eraisikuid tuleks teavitada pikselleerimise kui privaatsustööriista võimekusest ja piirangutest. Soovitatav on kasutada alternatiivseid meetodeid, näiteks teksti katmine läbipaistmatu ristkülikuga. Tuleb märkida, et dokumentide käsitlemisel ei tohiks sellist töötlemist rakendada. Levinud viga on Wordi failides teksti katmine mustade kastidega või teksti taustavärvi mustaks muutmine. Need on ainult visuaalsed abinõud ega eemalda allolevat teavet, võimaldades kolmandatel osapooltel originaalsisule juurde pääseda. [22]

On lootust, et lihtsalt pikselleeritud pilte murdva tarkvara avalikustamine edendab teadlikkust ühiskonnas turvakäitumise osas ja vähendab andmelekkeid, mis tekivad tulevikus pikselleerimise kasutamise tõttu. Töö tulemusena valminud programm on saadaval GitHub keskkonnas [23].

6. Kokkuvõte

Käesoleva lõputöö raames uuriti optilise märgituvastuse (OCR) võimalusi tugevalt pikselleeritud tekstide lugemiseks. Töö skoobis olid sõned, mille kõrgus piltidel oli kuni 6 pikslit. Pikselleeritud teksti lugemine on keeruline, kuna pikselleerimine põhjustab olulist selguskadu ja hävitab osa algandmetest.

Uurimistöö jooksul töötati välja spetsiaalsed masinõppemudelid, mis suudavad efektiivselt tuvastada tähemärke ja numbreid väga madala eraldusvõimega piltidelt. Analüüsiti erinevate närvivõrguarhitektuuride mõju täpsusele, sealhulgas hinnati hüperparameetrite ja mudeli keerukuse mõju lõpptulemustele.

Eksperimentide tulemusena selgus, et kitsalt spetsialiseerunud mudelid, näiteks numbriliste andmete tuvastamiseks, saavutavad märkimisväärselt kõrgema täpsuse tänu nende piiratud väljundklasside arvule. Uuriti ka generatiivsete võistlevate võrkude (GAN) rakendamist. Tulemused näitasid, et GAN-põhised superresolutsioonimeetodid võivad teatud olukordades pildi loetavust parandada, kuid nende kasutamisel esinevad olulised piirangud äärmiselt madala eraldusvõimega kujutiste korral. Töö käigus tõestati kontrollermudelite süsteemi efektiivsust võrreldes üldistavate mudelitega.

Lõputöö praktiliseks tulemuseks on avatud lähtekoodiga tarkvara ja treenitud mudelid, mis suudavad generaliseerida üle mitmesuguste fontide ning plokisuuruste. Välja pakuti automaatne mudelivalik kontroller-mudelite abil ning konsensusmehhanismide süsteem. Võib öelda, et arendatud lahendus saavutab kõrge tuvastustäpsuse ning on arvutuslikult tõhus, mis teeb selle sobivaks ja praktiliselt rakendatavaks reaalsetes olukordades.

Kasutatud kirjandus

[1] V. Garske and A. Noack. *Recovering information from pixelized credentials*. GI SICHERHEIT 2022. 2022. DOI: 10.18420/sicherheit2022_08 (vaadatud 02.10.2024).

[2] *Tesseract OCR*. 2025. URL: https://github.com/tesseract-ocr/tesseract (vaadatud 14.05.2025).

[3] D. Petro. *Never, Ever, Ever Use Pixelation for Redacting Text*. 2022. URL: https://bishopfox.com/blog/unredacter-tool-never-pixelation (vaadatud 02.10.2024).

[4] J. D. Gilbey and C.-B. Schönlieb. *An end-to-end Optical Character Recognition approach for ultra-low-resolution printed text images*. 2021. arXiv: 2105.04515 [cs.CV]. URL: https://arxiv.org/abs/2105.04515 (vaadatud 02.10.2024).

[5] X. Liu et al. *FOTS: Fast Oriented Text Spotting with a Unified Network*. 2018. arXiv: 1801.01671 [cs.CV]. URL: https://arxiv.org/abs/1801.01671 (vaadatud 13.04.2025).

[6] S. Hill et al. "On the (In)effectiveness of Mosaicing and Blurring as Tools for Document Redaction". In: *Proceedings on Privacy Enhancing Technologies* 2016 (Feb. 2016). DOI: 10.1515/popets-2016-0047 (vaadatud 02.10.2024).

[7] S. Mellema. *Recovering passwords from pixelized screenshots*. 2020. URL: https://spipm.nl/2030.html (vaadatud 02.10.2024).

[8] V. Buhrmester et al. "Evaluating the Impact of Color Information in Deep Neural Networks". In: *Pattern Recognition and Image Analysis*. Ed. by A. Morales et al. Cham: Springer International Publishing, 2019, pp. 302–316. ISBN: 978-3-030-31332-6 (vaadatud 14.01.2025).

[9] X. Xu et al. “Learning to Super-Resolve Blurry Face and Text Images”. In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 251–260. DOI: 10.1109/ICCV.2017.36 (vaadatud 06.03.2025).

[10] D. Dai et al. *DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models*. 2024. arXiv: 2401.06066 [cs.CL]. URL: https://arxiv.org/abs/2401.06066.

[11] Tensorflow. *Install TensorFlow with pip*. 2025. URL: https://tensorflow.org/install/pip%5C#windows-native (vaadatud 27.03.2025).

[12] Google. *Google Trends*. 2025. URL: https://trends.google.com (vaadatud 27.03.2025).

[13] Microsoft. *Microsoft ClearType overview*. 2022. URL: https://learn.microsoft.com/en-us/typography/cleartype/ (vaadatud 13.04.2025).

[14] A. K. Nain. *OCR model for reading Captchas*. 2020. URL: https://keras.io/examples/vision/captcha_ocr/ (vaadatud 14.05.2025).

[15] R. McPherson, R. Shokri, and V. Shmatikov. *Defeating Image Obfuscation with Deep Learning*. 2016. arXiv: 1609.00408 [cs.CR]. URL: https://arxiv.org/abs/1609.00408 (vaadatud 08.10.2024).

[16] Dwyl. *List Of English Words*. 2025. URL: https://github.com/dwyl/english-words (vaadatud 06.04.2025).

[17] D. Miessler et al. *SecLists: The pentester's companion*. 2025. URL: https://github.com/danielmiessler/SecLists (vaadatud 06.04.2025).

[18] Macdronic. *Elektri börsihind täna*. Andmete algallikas: Nord Pool. 2025. URL: https://home.macdronic.com/et/elekter/tana (vaadatud 17.03.2025).

[19] Google. *Overview of GAN Structure*. Litsentseeritud CC BY 4.0 all: https://creativecommons.org/licenses/by/4.0/. Tekstid tõlgitud. 2025. URL: https://developers.google.com/machine-learning/gan/gan_structure (vaadatud 07.03.2025).

[20] R. Brewster and dotPDN LLC. *Paint.NET source (v3.36.7; prior to non-MIT license change)*. 2009. URL: https://github.com/rivy/OpenPDN (vaadatud 08.04.2025).

[21] Api4ai. *Why Image Anonymization is Vital for GDPR Compliance*. URL: https://api4.ai/blog/why-image-anonymization-is-vital-for-gdpr-compliance (vaadatud 14.05.2025).

[22] Redactable. *Most embarrassing redaction failures in history, and how they can be avoided*. 2025. URL: https://redactable.com/blog/most-embarrassing-redaction-failures-in-history-and-how-they-can-be-avoided (vaadatud 13.04.2025).

[23] L. T. Lepik. *Depixelation software*. 2025. URL: https://github.com/lassethorlepik/Depixelation-software (vaadatud 14.05.2025).

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

I Lasse Thor Lepik

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Pikselleeritud teksti lugemine optilise märgituvastuse abil", mille juhendaja on Andres Käver
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

14.05.2025

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

Lisa 2 – Baasmudeli kihtide kood

```
class OCRModel(nn.Module):
    def __init__(self, img_width, img_height, num_chars):
        super(OCRModel, self).__init__()
        self.img_width = img_width
        self.img_height = img_height
        self.num_chars = num_chars
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)
        self.bn = nn.BatchNorm2d(128)
        self.relu = nn.ReLU(inplace=True)
        self.lstm_input_size = img_height * 128
        # RNN Layers: Two bidirectional LSTM layers with dropout.
        self.lstm1 = nn.LSTM(
            input_size=self.lstm_input_size,
            hidden_size=128,
            num_layers=2,
            bidirectional=True,
            batch_first=True,
            dropout=0.2,
            )
        self.lstm2 = nn.LSTM(
            input_size=2 * 128,
            hidden_size=64,
            num_layers=2,
            bidirectional=True,
            batch_first=True,
            dropout=0.2,
            )
        # Fully connected (Dense) layer
```

```
        self.fc = nn.Linear(2 * 64, num_chars)
        self._initialize_weights()

    def forward(self, x, y_true=None):
        x = self.conv1(x)  # [B, 32, H, W]
        x = self.relu(x)
        x = self.conv2(x)  # [B, 64, H, W]
        x = self.relu(x)
        x = self.conv3(x)  # [B, 128, H, W]
        x = self.bn(x)
        x = self.relu(x)
        x = x.permute(0, 3, 2, 1)
        (B, W, H, C) = x.size()
        x = x.contiguous().view(B, W, H * C)  # [B, W, H*128]
        (x, _) = self.lstm1(x)
        (x, _) = self.lstm2(x)
        x = self.fc(x)  # [B, W, num_chars]
        x = F.log_softmax(x, dim=2)  # For CTCLoss
        # Rearrange dimensions for CTCLoss: (T, B, num_chars)
        x = x.permute(1, 0, 2)  # [T, B, num_chars]
        T = x.size(0)  # Time steps
        B = x.size(1)  # Batch size
        input_length = torch.full((B, ), T, dtype=torch.long,
            device=x.device)
        label_length = torch.full((B, ), y_true.size(1),
            dtype=torch.long, device=x.device)
        if y_true is None:
            loss = 0
        else:
            loss = ctc_batch_cost(y_true, x,
                input_length,label_length)
            return (x, loss)
```

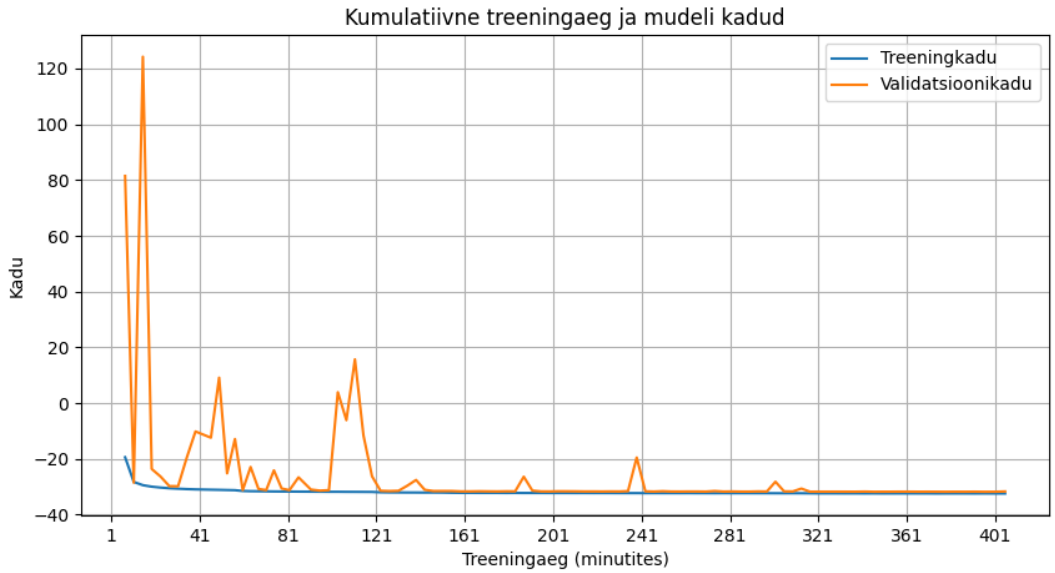
Joonis 12. *Baasmudeli kihtide kood.*

Lisa 3 – GAN generaatori kood

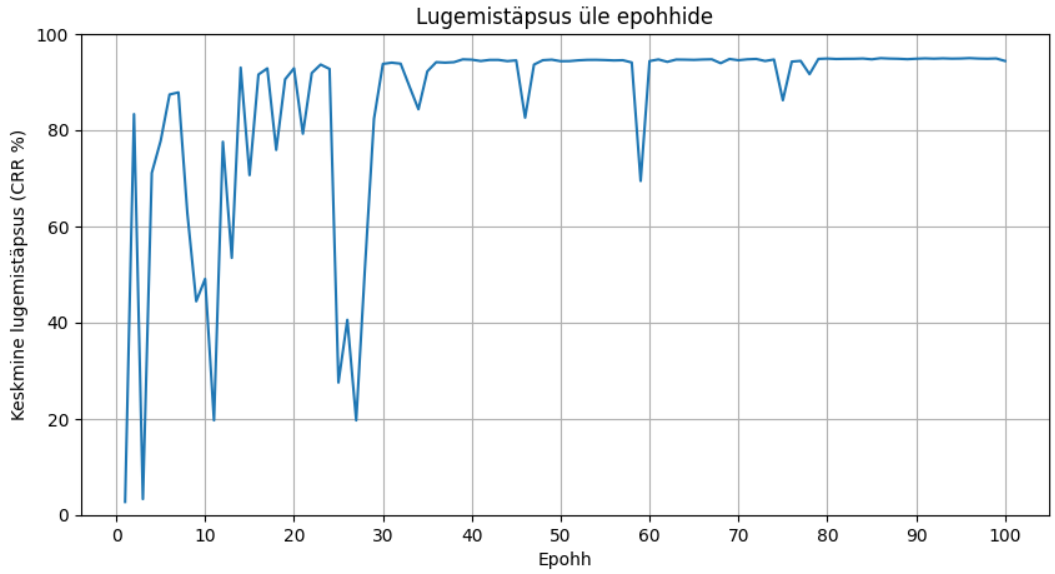
```
def build_generator(self):
    layers = []
    layers.append(nn.ConvTranspose2d(3, 64, kernel_size=6,
        stride=2, padding=2))  # 32x32 to 64x64
    layers.append(nn.BatchNorm2d(64))
    layers.append(nn.ReLU(inplace=True))
    layers.append(nn.Conv2d(64, 64, kernel_size=5,
        stride=1, padding=2))
    layers.append(nn.BatchNorm2d(64))
    layers.append(nn.ReLU(inplace=True))  # 64x64 to 128x128.
    layers.append(nn.ConvTranspose2d(64, 64, kernel_size=6,
        stride=2, padding=2))
    layers.append(nn.BatchNorm2d(64))
    layers.append(nn.ReLU(inplace=True))
    for _ in range(8): # 8 x conv to refine at 128x128
        layers.append(nn.Conv2d(64, 64, kernel_size=5,
            stride=1, padding=2))
        layers.append(nn.BatchNorm2d(64))
        layers.append(nn.ReLU(inplace=True))
    layers.append(nn.ConvTranspose2d(64, 64, kernel_size=6,
        stride=2, padding=2))  # 128x128 to 256x256
    layers.append(nn.BatchNorm2d(64))
    layers.append(nn.ReLU(inplace=True))
    # Final layer: convert features to 3 channels.
    layers.append(nn.Conv2d(64, 3, kernel_size=5,
        stride=1, padding=2))
    layers.append(nn.Tanh())
    return nn.Sequential(*layers)
```

Joonis 13. *GAN generaatori kood.*

Lisa 4 – Baasmudeli treenimise kulg

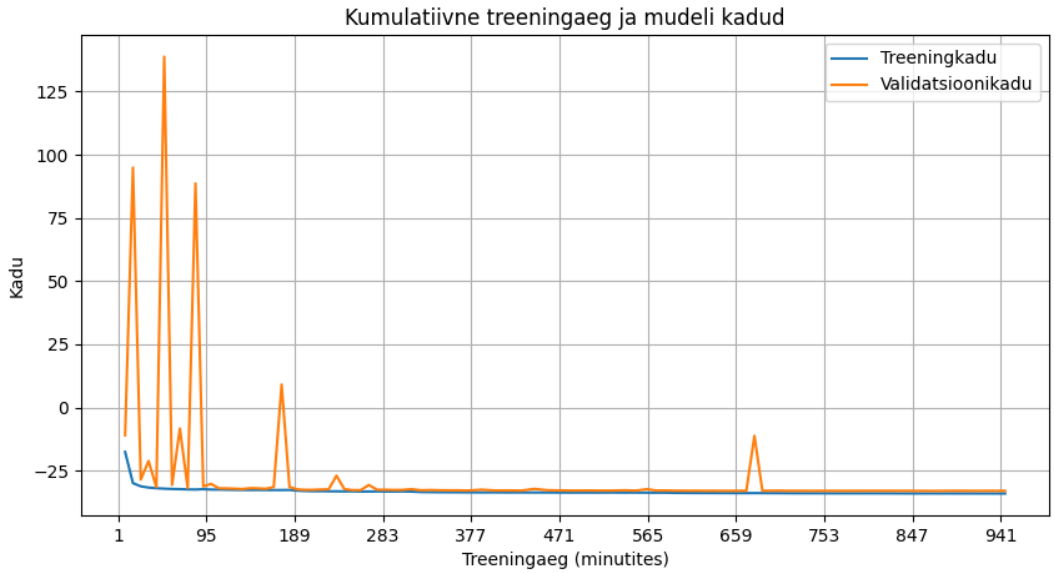


Joonis 14. *Baasmudeli kadud üle minutite.*

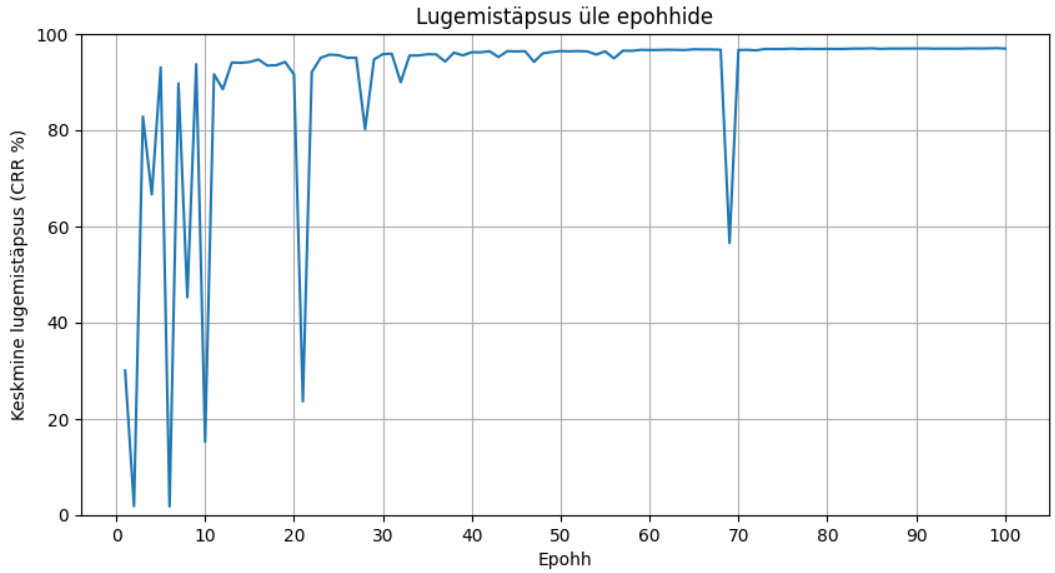


Joonis 15. *Baasmudeli loetud sõne täpsus üle epohhide.*

Lisa 5 – Kolme-kihilise LSTM mudeli treenimise kulg

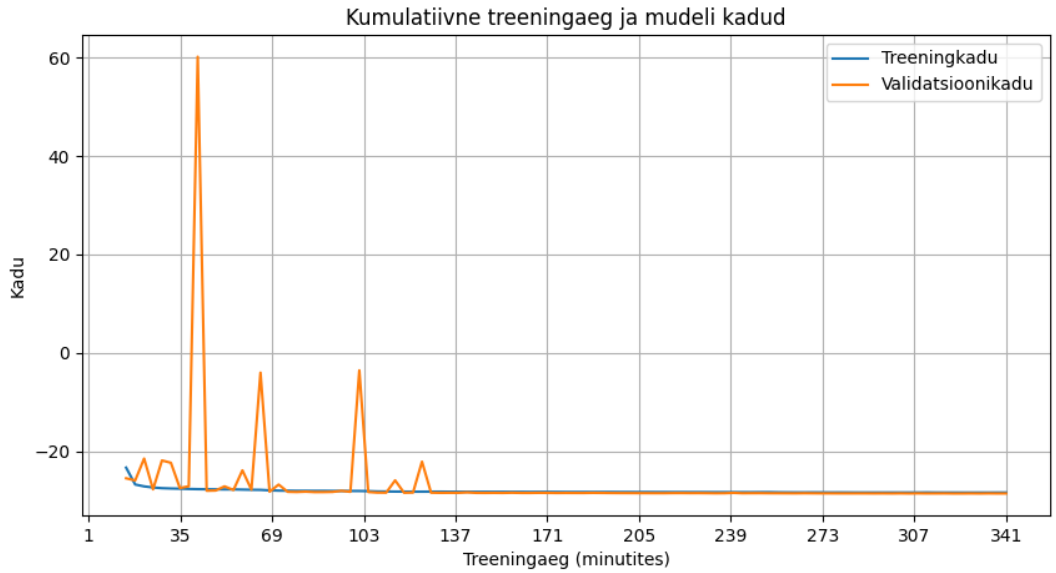


Joonis 16. *Kolmekihilise mudeli kadud üle minutite.*

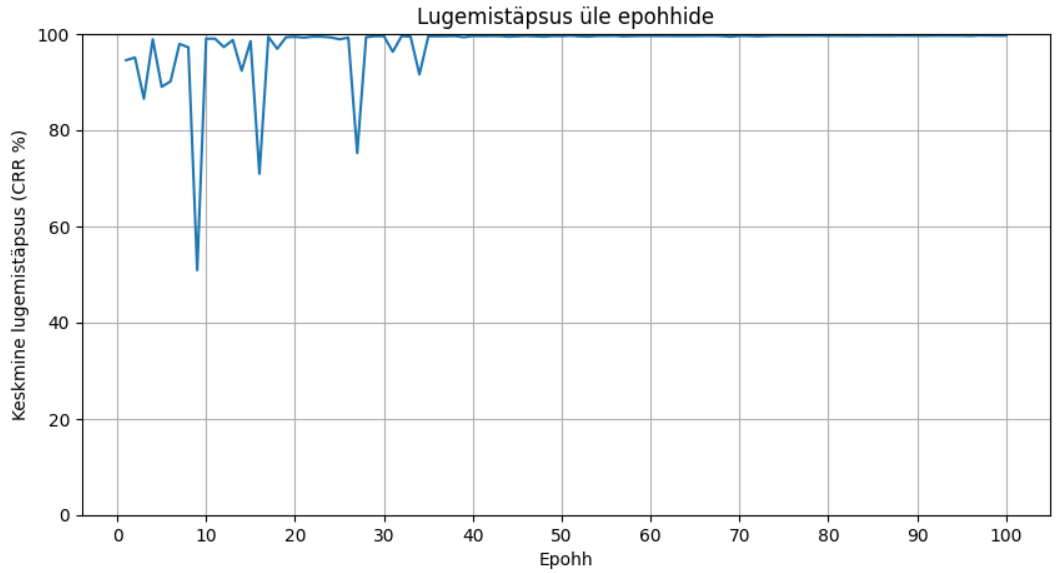


Joonis 17. *Kolmekihilise mudeli loetud sõne täpsus üle epohhide.*

Lisa 6 – Baasmudeli numbritel treenimise kulg

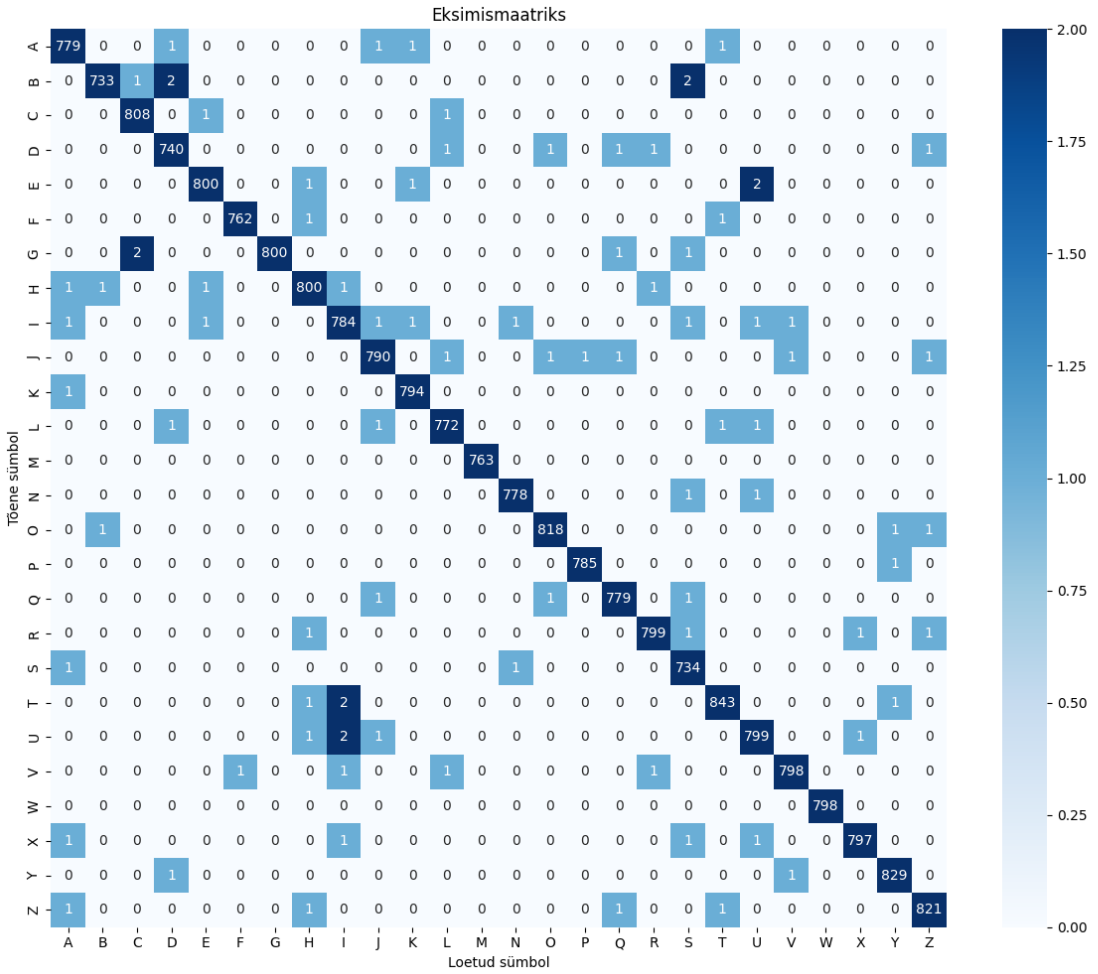


Joonis 18. *Numbritel treenitud baasmudeli kadud üle minutite.*



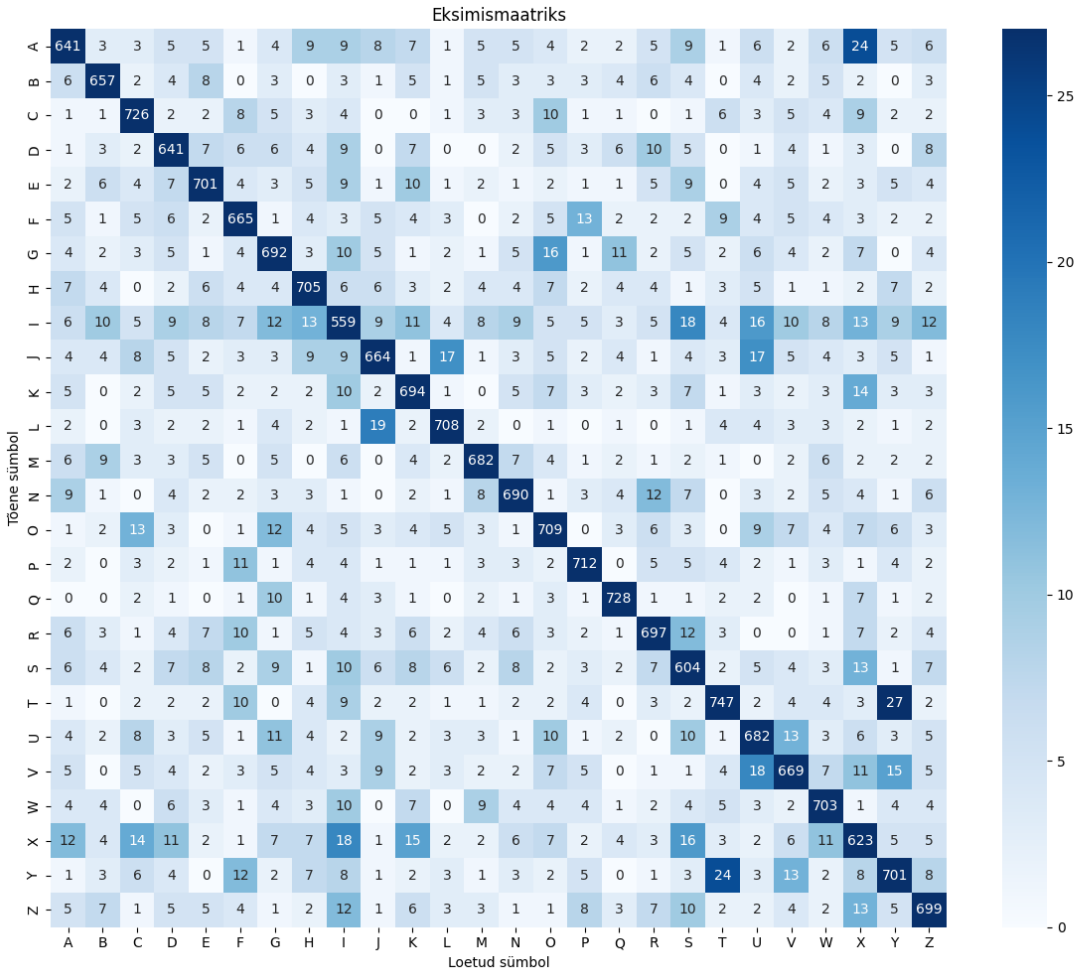
Joonis 19. *Numbritel treenitud baasmudeli loetud sõne täpsus üle epohhide.*

Lisa 7 – Eksimismaatriks plokisuurusel 8px



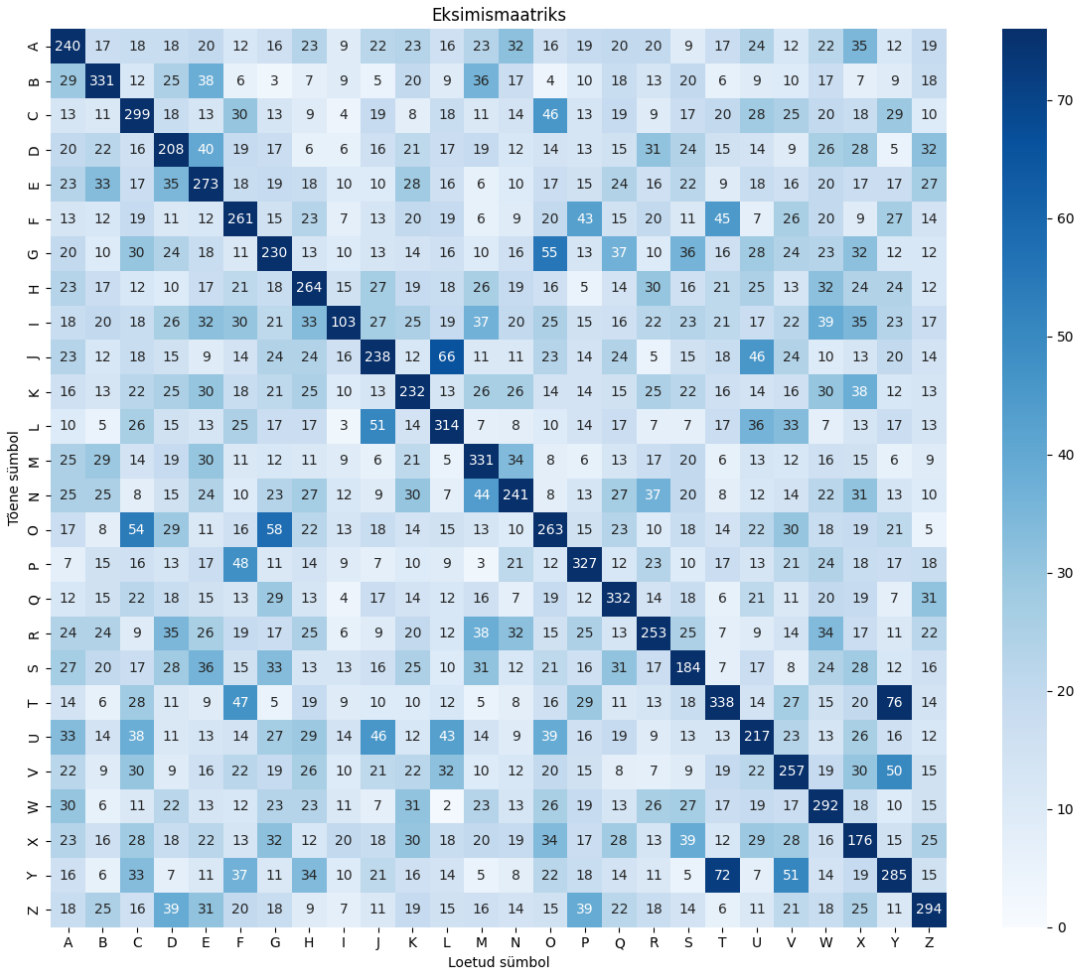
Joonis 20. *Eksimismaatriks, plokisuurus 8px, andmestik 200 000, N ≈ 40 000.*

Lisa 8 – Eksimismaatriks plokisuurusel 12px



Joonis 21. *Eksimismaatriks, plokisuurus 12px, andmestik 200 000, N ≈ 40 000.*

Lisa 9 – Eksimismaatriks plokisuurusel 16px



Joonis 22. *Eksimismaatriks, plokisuurus 16px, andmestik 200 000, N ≈ 40 000.*