TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Triin Rüütli 179628IABB

SARNASTE TOOTEPILTIDE SOOVITUSSÜSTEEM SIIRDEÕPPE JA OMADUSVEKTORITE ABIL

Bakalaureusetöö

Juhendaja Martin Rebane

MSc

Tallinn 2020

Autorideklaratsioon

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

Autor: Triin Rüütli

18.05.2020

Annotatsioon

Käesoleva töö eesmärgiks on leida sarnased tootepildid kasutades konvolutsioonilist närvivõrku omadusvektorite eraldamiseks. Töö eesmärk on ajendatud e-kaubanduse personaliseerimise olulisusest. Sarnaste tootepiltide leidmine aitab pakkuda e-poes personaalset tootevalikut, mis muudab kaupmehe kliendi silmis atraktiivseks.

Omadusvektorite eraldamiseks on kasutatud siirdeõppe meetodit mitme erineva konvolutsioonilise närvivõrgu alusel. Sarnasuste leidmiseks arvutatakse omadusvektorite vaheline kaugus ning mida väiksem on kaugus, seda sarnasemad on omavahel kaks pilti. Sarnaste piltide hoiustamiseks ja kiireks leidmiseks on analüüsitud kahte lähenemist – asukohatundlikku paisksalvestamist *(locality sensitive hashing)* ning graafiandmebaasi. Mõlema meetodi puhul on sarnased pildid leitud 3 erineva konvolutsioonilise närvivõrgu eraldatud omadusvektoritega. Töö tulemusena aktsepteeriti sarnasuste leidmiseks mõlemat lahendust.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 4 peatükki, 19 joonist, 8 tabelit.

Abstract

Recommentation system of similar product images using transfer learning and feature vectors

The purpose of this thesis is to find similar product images using convolutional neural networks for extracting feature vectors. The purpose is driven by the importance of personalization in e-commerce, where finding similar product images offers personalized sortiment of products. That makes the merchant attractive to the customers.

Feature vectors were extracted using transfer learning with multiple different pre-trained convolutional neural network architectures. In order to find the network with maximum accuracy, it is needed to compare different convolutional neural networks. The similarity between images can be found when calculating their feature vectors' distance. Smaller the distance, more similar are the images. In order to store and quickly find similar images, there has been analyzed 2 different options – locality sensitive hashing and graph database. Both methods were analyzed with types of feature vectors extracted with different neural networks. As a result of the thesis, locality sensitive hashing and graph database are both acceptable solutions for finding similar images. The author suggests to choose from these approaches depending on the number of products in the store.

The thesis is in Estonian and contains 28 pages of text, 4 chapters, 19 figures, 8 tables.

Lühendite ja mõistete sõnastik

Ahenduskiht	*Pooling layer*. Konvolutsioonilise närvivõrgu kiht, kus vähendatakse sisendi dimensioone
Asukohatundlik paisksalvestamine	*Locality sensitive hashing*. Ruumiliste vektorite paisksalvestamine, kus sarnased vektorid on lähestikku
Epohh	*Epoch.* Üks treeningtsükkel kogu treeningandmestikuga
Graafiandmebaas	Andmebaas, mis koosneb sõlmedest ja nendevahelistest seostest
Konvolutsiooniline kiht	Närvivõrgu kiht, kus lokaliseeritakse pildi omadused väikeste mõõtmetega filtrite abil
Konvolutsiooniline närvivõrk	Närvivõrk, mis koosneb konvolutsioonilisest kihist, ahenduskihist ning täissidusast kihist
Närvivõrk	*Neural network.* Arvutuslik arhitektuur, mis võimaldab masinõppe algoritmide abil töödelda keerulisi andmestikke
Omadusvektor	*Feature vector*. Pildist eraldatud omadused numbrilisel kujul
Siiami närvivõrk	*Siamese network.* Kaks identset konvolutsioonilist närvivõrku, mis jagavad omavahel parameetreid ja kaale.
Siirdeõpe	*Transfer learning.* Ühe ülesande käigus kogutud informatsiooni kasutamine teise ülesande lahendamiseks
Täissidus kiht	*Fully connected layer.* Närvivõrgu kiht, kus toimub pildi klassifitseerimine
Õpisamm	*Learning rate.* Parameeter, mis närvivõrgu treenimisel optimeerib kaale selliselt, et viga oleks minimaalne

Sisukord

Joonised

Tabelid

1 Sissejuhatus

Personaliseerimine on üks viis, kuidas muuta veebipoodi kliendile atraktiivseks. E-kaubanduse personaliseerimine seisneb tänapäeval veel suureski klientide segmenteerimises ja nende kliendisegmentide käitumise ennustamises. Kliendi ostukogemuse parandamiseks ja müüginumbrite suurendamiseks on üks variant kategooriate ja toodete põhine lähenemine, kus süsteem peaks pakkuma iga individuaalse kliendi jaoks relevantseid tooteid tema teekonna põhjal ehk süsteem peaks täitma nö konsultandi rolli.

Käesoleva töö eesmärk on luua soovitussüsteem, mis keskenduks piltide sarnasuste leidmisele, et kaardistada piltide omavahelised sarnasussuhted ning nende põhjal leida etteantud pildile kindel hulk sarnaseid vasteid. Piltide sarnasuste leidmiseks on kõige perspektiivikam kasutada kaht peamist lähenemist: siiami närvivõrkude kasutamine või madaladimensiooniliste omadusvektorite eraldamine ja neil sarnasusalgoritmide rakendamine. Käesoleva töö tulemuse hindamiseks seadsin kvaliteedikriteeriumi, mille kohaselt kõik tagastatud sarnased pildid peavad asuma otsingupildiga samas kategoorias.

Töö on jagatud mõtteliselt kolmeks suureks osaks. Esimeses osas rakendan siirdeõppe meetodit erinevate konvolutsiooniliste närvivõrkudega ning võrdlen saadud tulemusi. Teises osas kasutan asukohatundlikku paisksalvestamist *(locality sensitive hashing)* kolme erineva mudeli omadusvektorite sarnasuste leidmiseks. Kolmandas osas kasutan piltide sarnasuste leidmiseks graafiandmebaasi ning võrdlen samuti kolme erineva mudeli omadusvektorite tulemusi.

Töö protsessist on loodud GitHub repositoorium[1].

[1] https://github.com/triinruutli/Image-similarity

2 Äriline ja tehniline taustinformatsioon

Selles peatükis tutvustan oma töö eesmärki nii ärilistest kui ka tehnilistest aspektidest.

2.1 Äriline taust

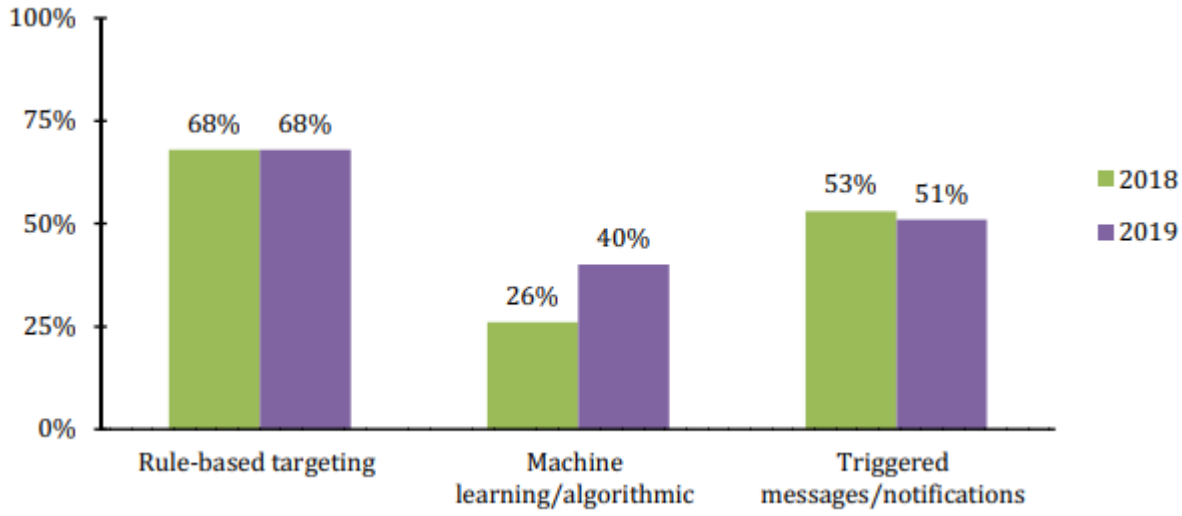
2.1.1 Personaliseerimine e-kaubanduses

E-kaubanduses on edu võtmeks personaliseerimine. Suur osa klientide ostudest toimuvad just veebipoodides ning seeläbi kasvab kaupmeestevaheline konkurents ka internetis. Klient on valmis rohkem maksma personaliseeritud ostukogemuse eest ning see omakorda tõstab kaupmehe müüki. 2017. aastal uuriti USAs veebipoodide kasutajatelt nende ostukogemuste kohta, 49% neist tõdeb, et tänu heale soovitussüsteemile on nad ostnud kaupu, mida algselt osta ei plaaninud [1].

Kaupmehe müüginumbrite kasv on tugevalt seotud e-poes kasutatavate personaliseerimise meetmetega, seda on tõestatud 2019. aastal USA-s tarkvarafirma Evergage poolt läbi viidud uuring [2]. 90% kaupmeestest, kes rakendasid vähemal või rohkemal määral personaliseerimise meetmeid, nägid oma müüginumbrites kasvu. Enam kui pooltel neist (58%) kasvas müük 10%, umbes 15% kaupmeestel kasvas müük enam kui 30%. Seega võib väita, et müügi kasvatamise ja personaliseerimise vahel on tugev seos.

Toodete soovitussüsteem on üks personaliseerimise tõhusamaid meetodeid. Tihti kasutatakse toodete soovitamise meetodit emaili turunduse ühe osana. Mina keskendun pigem eraldiseisvale soovitussüsteemile, mida rakendadakse kliendi ostuteekonna jooksul. Soovitussüsteem võib olenevalt tootevaliku suurusest olla algoritmilise või manuaalse lahendusega. Manuaalne lahendus tähendab seda, et kaupmees seob kaks toodet omavahel käsitsi. Selline lähenemine töötab väikse tootevalikuga poe puhul, üle 100 tootega ettevõtte puhul muutub see juba probleemseks ajalises ja füüsilise võimekuse mõttes.

Masinõppel põhinevate lahenduste kasutamine on viimase aasta jooksul jõudsalt kasvanud [2, p. 20] (joonis 1) ning kaupmehed plaanivad aina enam kasutusele võtta masinõppe või algoritmilisi lahendusi [2, p. 24].



Joonis 1. Algoritmiliste meetodite kasutusele võtmise kasv aastatel 2018-2019 [2, p. 24]

Hetketurul on palju tooteid, mis pakuvad e-kaubanduse personaliseerimist nii algoritmilisel kui mittealgoritmilisel viisil. Kuigi eesmärk on neil suuresti sama, erinevad need kõik funktsioonide võimekuse ja hinna poolest. Võtsin oma valimisse 4 erinevat lahendust – Prestashop Related Products module, Nosto, Klevu, STACC.

2.1.2 Turulolevad lahendused

Related Products Module on üks näide Prestashop *addons* keskkonna moodulist, mis võimaldab kaupmehel eelnevalt tooteid omavahel siduda. Tegemist ei ole tehisintellektil põhineva lahendusega, sest kaks toodet seotakse omavahel manuaalselt, küll aga vastab see moodul osaliselt minu kriteeriumile näidata kliendile sarnaseid tooteid. Kõnealune moodul eeldab kaupmehelt väga palju manuaalset tööd ja seega pole sobiv suurkaupmeestele. Moodul maksab 49.99 eurot.

Nosto on tehisintellektil põhinev toode, mis võimaldab personaliseerimist läbi viia väga paljudel tasanditel. Nosto analüüsib iga individuaalse kliendi käitumist nii veebipoes kui ka mujal internetis, et teada saada just temale relevantseid tooteid. Personaalne toodete soovitussüsteem on vaid üks osa Nosto tootest, seega tuleks kaupmehel osta kogu pakett. Paketi hind on 1000 eurot kuus ning maksmine toimub kord aastas ehk kaupmehe jaoks tähendaks see 12 000 eurost ühekordset väljaminekut aastas [3].

Klevu on tehisintellektil põhinev nutikas otsing, mis võimaldab personaliseerimist läbi viia kasutades e-poe otsingu funktsiooni. Klevu pakub väga palju funktsionaalsusi personaliseerimiseks, kuid võttes luubi alla personaalse soovitussüsteemi, siis toodete reklaamimine toimub ainult otsingu lahtrile klikkides. Klevu on suunatud just nendele klientidele, kes kasutavad oma ostuteekonnal valdavalt otsingut. Küll aga on võimalik lisana juurde osta Klevu Smart Category Navigation, mis seab kategoorialehel tooted paremusjärjestuse alusel ehk kõige relevantsemad tooted kuvatakse kliendile esimesena [4]. Pakettide hinnad algavad 499 eurot kuus.

STACC on Eestis tegutsev ettevõte, kelle peamisteks valdkondadeks on masinõpe ja andmeteadus. Ettevõte pakub ühe oma tootena toodete soovitussüsteemi, mis võtab arvesse ka kaupmehe ärilisi eesmärke [5]. Stacci tehtud tööna võin välja tuua näiteks Selveri Offline soovitussüsteemi personaliseeritud e-maili kampaaniate läbiviimiseks.

2.2 Tehniline taust

Käesolevas töös keskendun piltide sarnasuste leidmisele, et kaardistada piltide omavahelised sarnasussuhted ning nende põhjal leida etteantud pildile kindel hulk sarnaseid vasteid. Piltide sarnasuste leidmiseks on masinõppes kaks peamist lähenemist: siiami närvivõrkude kasutamine või madaladimensiooniliste omadusvektorite eraldamine ja neil sarnasusalgoritmide rakendamine. Selles peatükis kirjeldan eelmainitud lähenemisviise detailsemalt.

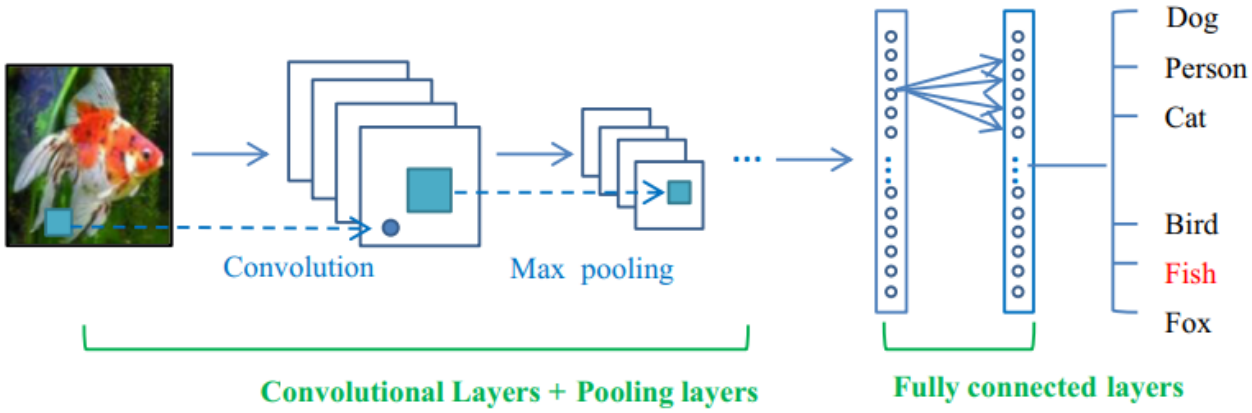
2.2.1 Omadusvektorid

Piltide sarnasuste leidmiseks teisendatakse kõrgemõõtmelised pildid madalamõõtmelisteks omadusvektoriteks. See on numbriliste väärtuste jada, mis iseloomustab pildi omadusi. Omaduseks võib olla värv, nurk, halltoonide intensiivsus, jms. Omadusvektorite analüüsimiseks ja sarnasuste leidmiseks tehakse nendega matemaatilisi arvutusi [6].

2.2.2 Konvolutsiooniline närvivõrk ja selle mudelid

Süvaõpe *(deep learning)* on masinõppe *(machine learning)* alamvaldkond, mis põhineb mitmetasandilisel informatsiooni töötlemisel, et luua seoseid andmete vahel. Süvaõppe arhitektuurid jagunevad järgmiselt: konvolutsioonilised võrgud, rekurrentsed võrgud, rekursiivsed võrgud, juhendamata eeltreenitud võrgud [7]. Kasutan oma töös just konvolutsioonilist närvivõrku, seega keskendun siinkohal ainult nimetatud arhitektuurile.

Konvolutsioonilise närvivõrgu *(convolutional neural network-CNN)* arhitektuuri kasutatakse peamiselt piltide ja visuaalide analüüsimiseks ja äratundmiseks. Arhitektuur koosneb kolmest närvikihist – konvolutsiooniline kiht *(convolutional layer)*, ahenduskiht *(pooling layer)* ja täissidus kiht *(fully connected layer)* (joonis 2) [8]. Konvolutsioonilistes kihtides toimub omaduste järkjärguline lokaliseerimine pildil, need kaardistatakse ja tulemuseks on omaduste maatriks. Ahenduskiht järgneb kovolutsioonilisele kihile ning seal toimub tunnuste maatriksi dimensioonide vähendamine, tulemuseks on pildi põhitunnustega ühedimensiooniline maatriks ehk omadusvektor. Täissidus kiht on kõige viimane ning seal konverteeritakse 2D omaduste maatriks 1D omadusvektoriks, toimub klassifitseerimine [8].



Joonis 2. Konvolutsioonilise närvivõrgu arhitektuur [8]

Konvolutsioonilised närvivõrgud on aja jooksul palju arenenud. Toon välja mõned populaarsemad mudelid - AlexNet, VGG, Inception, ResNet.

AlexNet mudel on üks esimestest arhitektuuridest, millega saavutati väga head tulemused ImageNet klassifikatsiooni määramisel. Arhitektuuris on 5 konvolutsioonilist kihti ning 3

täissidusat kihti [9]. AlexNet mudelis võeti kasutusele ReLu ehk mittenegatiivne lineaarfunktsioon, mis võimaldab mudelil treenimist kiirendada. Samuti lisati AlexNet mudelisse peale igat täissidusat kihti väljajätu kihtid *(dropout layers),* mis aitasid vähendada ülesobitamist *(overfitting)* [9].

VGG mudeliga asendati AlexNet mudelis kasutatavad suured kernelid väiksemate, 3x3 kernelitega. Mitu üksteise järel väikse suurusega kerneli kihti võimaldavad närvivõrgul õppida keerulisemaid tunnuseid [9]. Selline lahendus leidis laiemat kasutust edasistes närvivõrkudes peale VGG mudelit.

GoogLeNet närvivõrgus on 22 kihti – seega kiirem ja täpsem kui eelnevad mudelid. GoogLeNet mudeli idee seisneb selles, et enamike sõlmede aktivatsioon ei ole vajalik omavaheliste korrelatsioonide tõttu ehk kõik 512 sissetulevat vektorit ei ole ühenduses 512 väljamineva vektoriga [10]. Seega arvutuste vähendamiseks on loodud Inception moodul, mis paneb hõredate ühendustega närvivõrgu vastama tavalisele tihedate ühendustega närvivõrgule. Kuna enamik sõlmedest ei ole efektiivsed, siis on ka kernelite suurused väiksed (3x3, 1x1). Väiksed 1x1 konvolutsioonikihid mooduli alguses aitavad vähendada sissetulevate vektorite dimensioone enne kui need liiguvad edasi suuremate kernelitega konvolutsioonikihtidesse. Esimesed kihid aitavad vähendada arvutuste mahtu ning teevad närvivõrgu sügavamaks ja täpsemaks. GoogLeNet mudelis on asendatud viimased täissidusad kihid ahenduskihiga, mis arvutab 2D tunnustekaardilt välja keskmised väärtused. See võimaldab GoogLeNetil saavutada väga head täpsust ning on kiirem kui VGG mudel [10].

ResNet on arhitektuurilt sarnane VGG mudelile, koosnedes peamiselt 3x3 suuruste kernelitega konvolutsiooni kihtidest [11]. ResNet mudelis võeti kasutusele residuaalne plokk, kus kasutatakse kihtide vahele jätmist selleks, et luua otseühendus sisendi ja väljundi vahel. Üle hüppamisel lisatakse viimase kihi väljund järgneva kihi väljundile, seega saadakse uus väärtus, mis on sisendiks veel järgmisele kihile. Sarnaselt GoogLeNet mudelile kasutab ka ResNet viimaste täissidus kihtide asemel ahenduskihti [11].

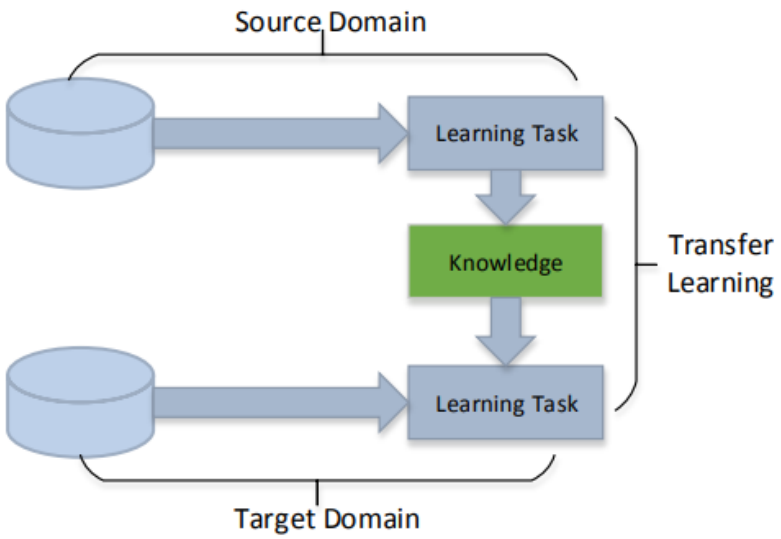
Eelnevatest mudelitest võib järeldada, et mida sügavam ehk mida rohkem konvolutsioonilisi kihte on närvivõrgul, seda täpsemad on küll tulemused, kuid aina keerulisem on närvivõrku treenida.

2.2.3 Siiami närvivõrgud

Siiami närvivõrk on arhitektuur, milles on kaks identset konvolutsioonilist närvivõrku, mis jagavad omavahel sisendite parameetreid ja kaale. Mõlemad võrgud koosnevad konvolutsiooni kihtidest, ahenduskihtidest ja täissidusatest kihtidest. Eesmärk on saada sisendpiltidele optimaalsed omadusvektorid, mis sarnasuse korral on vektorruumis lähestikku ja mittesarnasuse korral üksteisest kaugemal [12]. Siiami närvivõrke kasutatakse näotuvastussüsteemides, sellest lähtuvalt on võimalik siiami närvivõrke rakendada ka käesoleva töö eesmärgi saavutamiseks.

2.2.4 Siirdeõpe

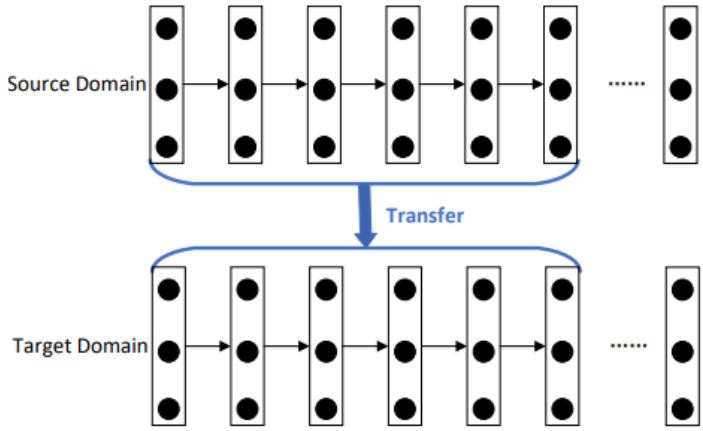
Siirdeõpe masinõppes *(deep transfer learning)* on tehnika, kus lähteülesandes kogutud informatsiooni antakse närvivõrgu kaudu edasi ning kasutatakse sihtülesande lahendamiseks [13]. Joonisel 3 on kirjeldatud siirdeõppe protsessi, kus *source domain* koosneb lähteandmete kogumikust ja lähteülesandest ning *target domain* sihtandmete kogumikust ja sihtülesandest. Siirdeõpe toimub lähteülesande teadmiste jagamisel sihtülesande lahendamiseks.



Joonis 3. Siirdeõppe protsess [13]

Tulenevalt siirdeõppe kasutamise tehnikatest on tekkinud järgnevad kategooriad: juhtumipõhine *(instance-based)*, tunnusepõhine *(feature-based)*, mudelipõhine *(model-based)* ja suhtepõhine *(relation-based)* siirdeõpe [14]. Kuna käesolevas töös kasutan mudelipõhist siirdeõppe tehnikat, siis keskendun ainult sellele kategooriale.

Mudelipõhine siirdeõpe on tehnika, kus eelnevalt suurte andmekogude peal treenitud masinõppe mudelit kasutatakse uue mudeli loomisel. Eeltreenitud mudeli esimesed kihid kantakse üle ning algab uue mudeli treenimine (joonis 4) [13].



Joonis 4. Mudelipõhise siirdeõppe protsess [13]

Lähtedomeenilt kantakse üle treenitud mudeli esimesed konvolutsioonikihid koos kaaludega, viimane kiht jäetakse välja. Konvolutsioonilise masinõppe mudeli esimestes kihtides toimub pildi tunnuste eraldamine, viimases kihis aga pildi tunnuste põhjal klassifikatsiooni määramine. Sihtdomeeni mudelit on vaja kohandada uuele andmestikule vastavaks, ühe levinuima viisina külmutatakse kovolutsioonikihid ning mudeli viimane kiht treenitakse uue andmestikuga läbi [13].

Siirdeõppe kasutamisel on mitmeid eeliseid. Kuna kogu mudeli treenimine on väga ajamahukas protsess, siis eeltreenitud mudeli kasutamine ja ainult viimase kihi modifitseerimine on vähem aeganõudvam lahendus. Samuti on siirdeõppe kasutamisel lõpptulemused väga täpsed, sest mudelit on treenitud suure andmekogu peal ning seetõttu on mudelil eraldatud väga palju erinevaid omadusvektoreid. Sellest tulenevalt on võimalik

siirdeõppe tehnikat kasutada ka väikeste andmekogude peal ning saada väga täpsed tulemused.

2.2.5 Sarnasuse leidmine

Kahe pildi sarnasuse leidmiseks rakendatakse sarnasusalgoritme, mis põhinevad omadusvektorite omavahelise kauguse arvutamisel. Pildiotsingus kasutatakse peamiselt omadusvektorite kauguste leidmiseks kas Eukleidilist mõõdet, mis leiab lühima tee kahe punkti vahel (valem 1) [15] või koosinussarnasuse mõõdet, mis leiab kahe n-dimensionaalse vektori vahelise nurga koosinuse (valem 2) [16].

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Valem 1. Eukleidilise kauguse leidmise valem

$$d(p,q) = \frac{A \times B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\ \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Valem 2. Koosinussarnasuse leidmise valem

Sarnasuste leidmiseks võib kasutada k-lähima naabri algoritmi *(k-nearest neighbour algorithm*) ja ligikaudset lähima naabri algoritmi (*approximate nearest neighbour algorithm*). K-lähima naabri algoritmi idee seisneb etteantud vektori võrdlemises kõikide teiste vektoritega andmebaasis ning tagastatakse k kõige lühema kaugusega vektorit. See lahendus on suurte andmekogude korral ebamõistlik valik, kuna sarnaste piltide leidmine võtab väga kaua aega [17]. Ligikaudne lähima naabri algoritm seevastu ei otsi seoseid kõikide vektorite seast, vaid tagastab kõige lähemate vektoritega peaaegu samal kaugusel olevad ehk ligikaudsed vektorid. Kuigi tagastatud vektorid ei ole kõige täpsemad, on kahe algoritmi tagastatud vektorite kauguste erinevused väikesed. See lähenemine vähendab arvutuste mahtu ning võtab tulemuste tagastamiseks tunduvalt vähem aega kui k-lähima naabri algoritm [18].

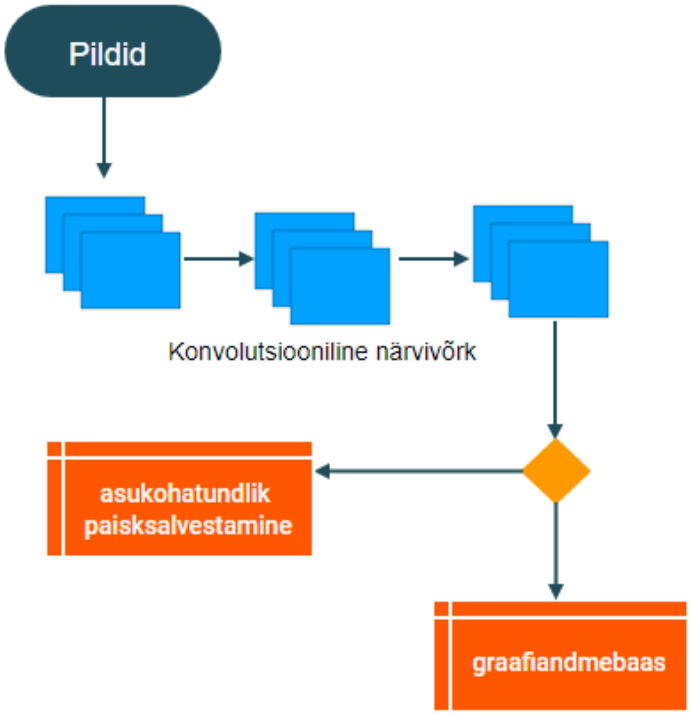
2.2.6 Töövahendid

Käesoleva töö raames kasutasin sarnasuste leidmiseks *locality sensitive hashing* meetodit, mis on ligikaudse lähima naabri algoritm. See on ruumiliste vektorite paisksalvestamine, kus kõrgedimensiooniliselt sarnased pildid paigutatakse massiivis sama indeksi alla ning indeksi alla tekib uus massiiv, mis hoiab sarnaseid pilte lähestikku. Uue pildi lisamisel luuakse talle räsi *(hash)*, mis määrab vektori asukoha massiivis ning võrreldakse uue vektori kaugust kõikide samas asukohas olevate vektoritega [18].

Sarnasuste leidmiseks kasutasin teise meetodina graafiandmebaasi. Graafiandmebaas võimaldab säilitada lisaks piltide omadusvektoritele ka nende omavahelisi seoseid, mis kirjeldavad sarnasust. Graafiandmebaasi sisendiks on eelnevalt eraldatud omadusvektori massiiv.

3 Realisatsioon

Käesoleva töö algusetapis kaalusin piltide sarnasuse analüüsimiseks ja tuvastamiseks kaht võimalust - siiami närvivõrkude kasutamine või erinevate konvolutsiooniliste mudelite abil omadusvektorite eraldamine ja sarnasuse tuvastamiseks sarnasusalgoritmide rakendamine. Närvivõrgu treenimine on väga ajakulukas protsess, seega otsustasin alustada esmalt teise variandiga ehk kasutada eeltreenitud mudeleid omadusvektorite eraldamiseks. Joonisel 5 olen kirjeldanud välja töötatud lahenduse arhitektuuri.



Joonis 5. Töö käigus välja töötatud lahenduse arhitektuur

Esmalt katsetasin omadusvektorite eraldamist ja sarnasuse leidmist Img2Vec teegi abil. Seejärel kasutasin siirdeõppe meetodit erinevate eeltreenitud mudelitega ning sarnasuste leidmiseks kasutasin nii asukohatundliku paisksalvestamise *(locality sensitive hashing)* meetodit kui ka graafiandmebaasi. Järgnevalt olen alapeatükkidena välja toonud kirjeldatud tegevused detailsemalt.

3.1 Eeltreenitud mudeli kasutamine Img2Vec teegi abil

Esmalt kasutasin Pytorchi Img2Vec[2] teeki, mis võtab aluseks nii ResNet-18 kui ka AlexNet eeltreenitud mudelid [19]. ResNet-18 eeldab, et sisendandmestiku pildid oleksid mõõtmetes 224x224. Img2Vec teegi puhul teisendatakse sisendandmestik vastavatesse mõõtmetesse ja dimensioonidesse enne kui lastakse pilt mudelist läbi. Väljundiks on multidimensionaalne massiiv omadusvektoritest. Sarnasus arvutatakse koosinussarnasuse mõõtena, mis leiab mitmetasandilises ruumis kahe punkti vahelise nurga koosinuse.

Rakendasin Img2Vec teeki kahe pildi sarnasuse leidmiseks nii ResNet-18 kui ka AlexNet mudeliga. Sisendpildid on nähtavad joonisel 6 ning tulemused on kirjeldatud järgnevas tabelis (tabel 1).

Joonis 6. Sisendpildid Img2Vec teegi abil sarnasuse arvutamiseks

ResNet-18	**AlexNet**
Sarnasus 62.44%	Sarnasus 31.71%

Tabel 1. Img2Vec teegi abil sarnasuse arvutamise tulemused kahe erineva mudeliga

Saadud tulemused on liiga erinevad ja ei ole usaldusväärsed.

[2] https://github.com/christiansafka/img2vec

3.2 Eeltreenitud mudeli kasutamine siirdeõppe meetodil

Üheks suurimaks siirdeõppe eeliseks süvaõppe ees on aja kokkuhoid. Siirdeõppe puhul ei treenita tervet mudelit algusest peale, vaid uue mudeli loomisel võetakse aluseks juba eelnevalt suurte andmekogudega treenitud mudel ning treenitakse see uue andmestikuga üle. Võtsin aluseks valmis algoritmi github repositooriumist, mis kasutab siirdeõpet [20]. Siirdeõppe meetodi jaoks kasutasin FastAI[3] teeki, mis põhineb PyTorch[4] masinõppe teegil. Nimetatud teegid aitavad lihtsustada närvivõrkude treenimist kaasaegsete praktikatega.

Treenimiseks võtsin andmekogumiku Caltech-101 [21]. Andmestik koosneb 9144 erinevast pildist, mis on jagatud 101 kategooriasse. Enne treenimist jagasin andmekogu kaheks osaks – treeningandmestik, mida kasutatakse mudeli treenimisel ning valideerimise andmestik, mida kasutatakse mudeli tulemuste hindamiseks. Transformeerisin andmekogu pildid kindlatesse mõõtmetesse, et saaksin neid mudeli treenimisel kasutada.

FastAI teegi funktsioon *cnn_learner* loob andmekoguga mudeli, mille konvolutsioonikihid on külmutatud ning viimased täissidusad kihid on eemaldatud. Uute täissidusate kihtide treenimiseks leian esmalt optimaalse õpisammu *(learning rate)*. Õpisammu valimisel ei tohi valida liiga suurt ega liiga väikest väärtust. Liiga suure õpisammuga treenimisel on kaalude muutused liiga suured ja optimiseerimisel ei leitagi minimaalset kaofunktsiooni. Liiga väikse õpisammuga on küll treenimine täpsem, kuid väikeste kaalude muudatustega on minimaalse kaoni jõudmine pikem ja võtab väga palju aega. FastAI teek võimaldab optimaalse õpisammu leidmiseks käivitada mudelit väikse alamhulga andmete peal paljude erinevate õpisammudega, et tekiks graafik.

Viimaste kihtide treenimisel kasutan leitud optimaalset õpisammu, kõik eelnevad konvolutsioonilised kihid koos kaaludega on külmutatud. Salvestasin olemasoleva mudeli koos õpitud täissidusate kihtide kaaludega, et saaksin edaspidi mudelit paremaks häälestada.

[3] https://docs.fast.ai/

[4] https://pytorch.org/

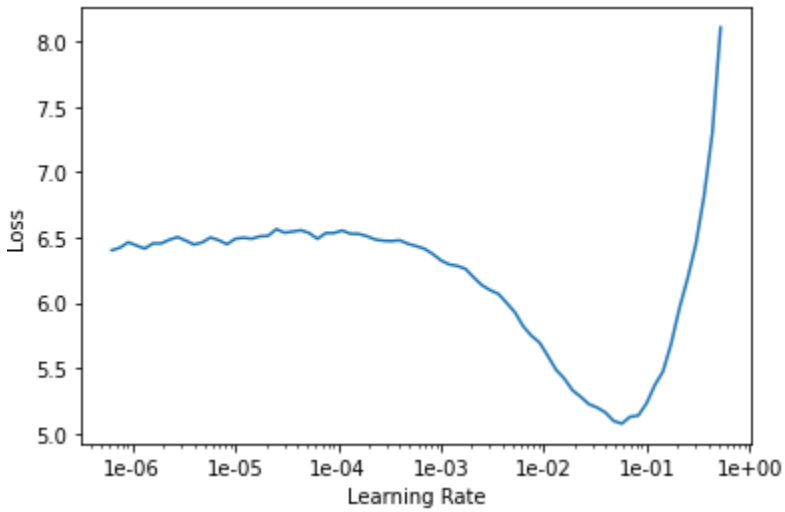
Seejärel sulatasin mudeli konvolutsioonilised kihid ning korrates eelnevalt kirjeldatud protsessi treenisin kogu mudeli.

Järgmistes alapeatükkides toon siirdeõppe protsessi tulemused erinevate eeltreenitud mudelitega.

3.2.1 ResNet mudelite kasutamine

Kuna ResNet mudelid varieeruvad konvolutsioonikihtide arvult, siis võrdluseks teen kogu protsessi läbi ResNet-18, ResNet-34 ja ResNet-101 mudelitega.

Võtsin ResNet-18 mudeli, mis koosneb 18 konvolutsioonilisest kihist. Joonisel 7 on välja toodud mudeli ResNet-18 täissidusate kihtide õpisammu graafik, kust võib näha, et minimaalne kadu saavutatakse õpisammude 1e-02 ja 1e-01 vahel.



Joonis 7. ResNet-18 täissidusate kihtide treenimiseks optimaalse õpisammu graafik.

Täissidusate kihtide treenimisel kasutan eelnevalt leitud optimaalset õpisammu 1e-02, kõik eelnevad konvolutsioonilised kihid koos kaaludega on külmutatud. 5 epohhi tulemusel sain täpsuseks 91,5%, salvestasin täissidusate kihtide treenimisel saadud tulemused. Seejärel sulatasin mudeli konvolutsioonilised kihid ning korrates eelnevalt kirjeldatud protsessi treenisin kogu mudeli. Kogu mudeli treenimise tulemusi on näha tabelis 2.

epoch	train_loss	valid_loss	accuracy	time
0	0.335155	0.289074	0.915208	01:02
1	0.278094	0.284138	0.918490	01:02
2	0.311860	0.272056	0.919584	01:01
3	0.268254	0.271067	0.920131	01:02
4	0.282987	0.272487	0.918490	01:01

Tabel 2. ResNet-18 mudeli treenimisel 5 epohhi tulemused.

Sarnaselt ResNet-18 mudelile kasutasin sama protsessi ka ResNet-34 ja ResNet-101 mudelite puhul. Treenimiste protsess on täpsemalt näha github repositooriumis, alltoodud tabelites olen välja toonud vaid lõplikud tulemused (tabelid 3 ja 4).

epoch	train_loss	valid_loss	accuracy	time
0	0.324506	0.291054	0.917943	01:12
1	0.263094	0.286264	0.920131	01:13
2	0.209477	0.277733	0.923961	01:13
3	0.253106	0.271091	0.923414	01:13
4	0.239918	0.272013	0.922319	01:13

Tabel 3. ResNet-34 mudeli treenimisel 5 epohhi tulemused.

epoch	train_loss	valid_loss	accuracy	time
0	0.329249	0.312117	0.909190	02:58
1	0.359676	0.316841	0.909737	02:58
2	0.271106	0.254013	0.923414	02:58
3	0.152593	0.218876	0.931619	02:58
4	0.119171	0.215369	0.933260	02:58

Tabel 4. ResNet-101 mudeli treenimisel 5 epohhi tulemused.

Eelnevatest tabelitest saab järeldada, et kuigi kõige täpsemad tulemused andis ResNet-101 93.3%, siis ajakulu oli selle mudeli puhul kõige suurem. Kuna ResNet-101 on kõige sügavama arhitektuuriga mudel, siis suur ajakulu on loogiline. Kõige vähem aega kulus ResNet-18, kuid see-eest täpsus on mudelitest madalaim 91.8%. ResNet-34 oli ajakulu poolest üsna võrdne ResNet-18 mudeliga, kuid täpsus oli suurem 92.2%. Närvivõrgu treenimine toimub ainult üks kord, seega ajakulu ei ole tegelikult mudeli valiku seisukohast oluline. Oluline valiku kriteerium on täpsus, eelnevatest mudelitest näitas kõige paremat tulemust ResNet-101.

3.2.2 Inception mudeli kasutamine

Lisaks ResNet mudelitele võtsin võrdlusesse ka Inception eeltreenitud mudeli, sest mainitud mudeli arhitektuur erineb suuresti ResNet arhitektuurist. Kuna Inception ei ole FastAI teegi mudel, siis kasutasin Cadene[5] eeltreenitud mudeleid. Kui ResNet mudelite puhul valmistas *cnn_learn*er funktsioon mudeli siirdeõppeks ette, siis Inception mudeli puhul pean käsitsi määrama lõikekoha. Kasutasin selleks joonisel 8 olevat käsku [22].

```
from fastai.vision.models.cadene_models import *
learn = cnn_learner(data, inceptionv4,pretrained=True,cut=-
2, split_on=lambda m: (m[0][11], m[1]), metrics=accuracy)
```

Joonis 8. Inception mudeli loomine.

Sarnaselt ResNet mudelite tööprotsessile, leidsin ka Inception mudelile optimaalse õpisammu, treenisin täissidusad kihid ning kogu mudeli tervikuna. Kogu mudeli treenimine võttis aega 25 minutit. Tulemusi on näha allolevas tabelis (tabel 5)

[5] https://github.com/fastai/fastai/blob/master/fastai/vision/models/cadene_models.py

epoch	train_loss	valid_loss	accuracy	time
0	0.873075	0.737094	0.819475	04:58
1	0.856284	0.520119	0.875274	04:59
2	0.483637	0.366525	0.911926	04:59
3	0.219065	0.245001	0.939825	05:00
4	0.069466	0.206115	0.948031	05:00

Tabel 5. Inception mudeli treenimise tulemused.

Inception treenitud mudeli täpsus on 94,8%, mis on suurem kui ühelgi ResNet-34 mudelil, kuid ajakulu on see-eest peaaegu kahekordne.

3.2.3 DenseNet mudeli kasutamine

Otsustasin lisaks Inception ja ResNet arhitektuuridele võtta juurde ka kolmanda arhitektuuri, milleks valisin DenseNet mudeli. DenseNet arhitektuur koosneb tihedatest plokkidest, kus konvolutsioonikihid on omavahel ühenduses järgmiselt: iga konvolutsioonilise kihi sisend on eelmiste kihtide väljund. See lähenemine muudab kaofunktsiooni minimeerimise efektiivsemaks, sest iga uus kiht on tihedas seoses eelmiste kihtidega [23].

Sarnaselt eelmiste mudelite tööprotsessile, leidsin ka DenseNet-121 mudelile optimaalse õpisammu, treenisin täissidusad kihid ning kogu mudeli tervikuna. Kogu mudeli treenimine võttis aega 15 minutit 30 sekundit. Tulemusi on näha allolevas tabelis (tabel 6):

epoch	train_loss	valid_loss	accuracy	time
0	0.290231	0.362467	0.897155	02:10
1	0.288892	0.335733	0.900985	02:10
2	0.232204	0.329725	0.903173	02:11
3	0.222284	0.314195	0.908643	02:11
4	0.193381	0.312665	0.910832	02:11

Tabel 6. DenseNet-121 mudeli treenimise tulemused.

Tulemustest lähtuvalt on DenseNet121 mudeli ajakulu kiirem kui Inceptionil, kuid aeglasem kui ResNet-34. Kui võrrelda mudeleid täpsuses, siis DenseNet-121 täpsus on kõikidest treenitud mudelitest kõige madalam, kõigest 91.08%.

3.3 Omadusvektorite salvestamine Pytorch Hook meetodil

Treenitud mudeli viimase täissidusa kihi väljund on multidimensiooniline omadusvektor. Omaduste eraldamiseks ja salvestamiseks kasutasin Pytorch Hook tehnikat, mille idee seisneb mudelisse funktsiooni paigutamises, mis kogub ja salvestab informatsiooni. Haagid jagunevad kaheks vastavalt info liikumise suunale. Kuna eesmärk on salvestada viimasest konvolutsiooni kihist tulev väljund, siis kasutasin haaki, mis käivitub info pärisuunas liikumisel.

Edaspidiseks analüüsiks ja sarnasuse kvaliteedi hindamiseks on mõistlik luua pildi nime ja asukoha ning omadusvektori vahel seos. Selleks salvestasin andmed sõnastiku andmestruktuurina (*dictionary*), kus võtmeks on pildi asukoht ning väärtuseks omadusvektor. Omadusvektorite põhjal sarnasuse leidmiseks kasutan kaht erinevat lähenemist, seetõttu on mõistlik omadusvektorite sõnastik salvestada faili, et vajadusel saaks andmeid hõlpsasti kasutada.

Salvestasin sõnastiku kahel erineval viisil. Esmalt salvestasin andmed kasutades Python pickle moodulit (joonis 9) ning seejärel salvestasin andmed ümber CSV failiformaati (joonis 10). CSV fail on vajalik graafiandmebaasi andmete laadimisel.

```
pickle.dump(feature_dict, open('101_ObjectCategories/feature_dict.p',
'wb'))
```

Joonis 9. Käsk sõnastiku salvestamiseks pickle mooduliga

```
feature_dict = pickle.load(open('101_ObjectCategories/feature_dict.p',
'rb'))
with open('output.csv', mode='w') as csv_file:
    writer = csv.writer(csv_file)
    for key, val in feature_dict.items():
        writer.writerow([[list(val)]+[key]])
    csv_file.close()
```

Joonis 10. Käsk sõnastiku salvestamiseks CSV faili.

3.4 Sarnasuse leidmine asukohatundliku paisksalvestamisega

Asukohatundlik paisksalvestamine *(locality sensitive hashing)* on ligikaudse lähima naabri leidmise algoritm, kus sarnased pildid on paigutatud massiivis lähestikku. Päringu kiirus on suur tänu limiteeritud arvutustele, mis tehakse ainult sama räsiga piltide puhul. Selle meetodi puhul võtsin samuti aluseks eelmainitud GitHub repositooriumi [20]. Kasutasin paisksalvestamiseks teeki lshash[6], mis võimaldab Pythoni programmeerimiskeelega kasutada asukohatundlikku paisksalvestamise meetodit.

Funktsioon võtab parameetriteks räsi suuruse, tabelite arvu ning sisendi dimensionaalsuse numbrina. Võtan aluseks eelnevalt salvestatud sõnastiku piltide asukohtade ja omadusvektoritega, salvestan pildid ükshaaval räsitabelitesse. Kuna sarnasust teostatakse ja asukohta massiivis määratakse omadusvektorite põhjal, siis määran sisendi võtmeks esmalt omadusvektori ning väärtuseks pildi asukoha (joonis 11).

```
k = 10
L = 5
d = 512
lsh = lshash.LSHash(hash_size=k, input_dim=d, num_hashtables=L)
for img_path, vec in tqdm_notebook(feature_dict.items()):
    lsh.index(vec.flatten(), extra_data=img_path)
```

Joonis 11. Asukohatundliku paisksalvestamise loomine ja pildide salvestamine

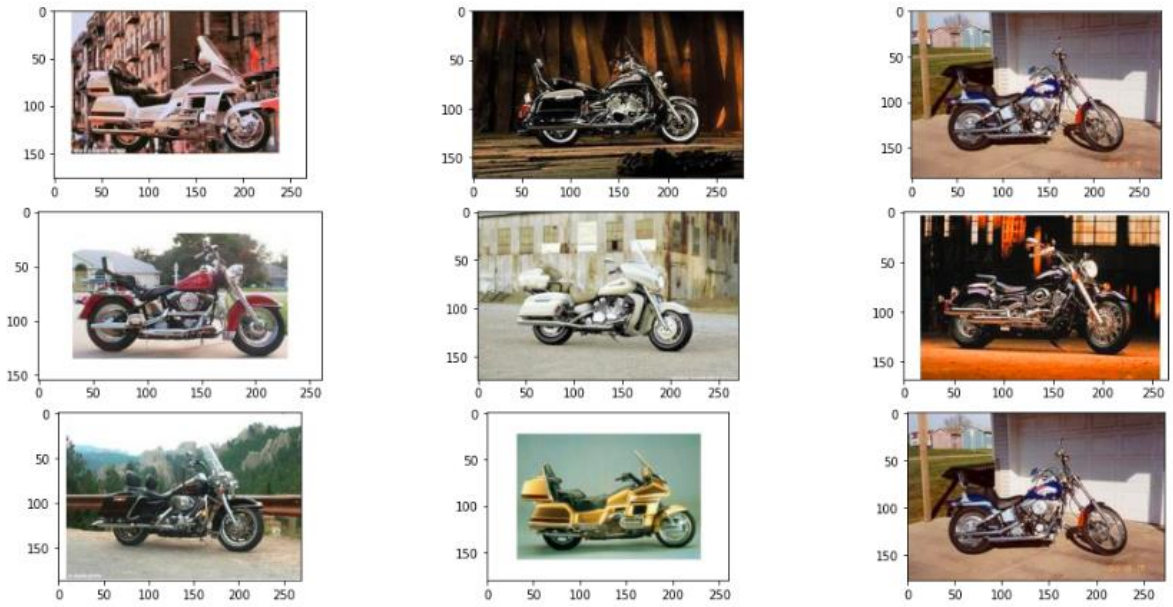
[6] https://pypi.org/project/lshash/

Ette antud pildile sarnaste piltide leidmiseks paisksalvestusest kasutan funktsiooni (joonis 12), mis võtab parameetriteks ette antud pildi indeksi sõnastikus, omadusvektorite sõnastiku, paisksalvestuse ning näidatavate tulemuste arvu [20].

```
def get_similar_item(idx, feature_dict, lsh_variable, n_items=5):
    response = lsh_variable.query(feature_dict[list(feature_dict.keys(
))[idx]].flatten(),num_results=n_items+1, distance_func='hamming')
    columns = 3
    rows = int(np.ceil(n_items+1/columns))
    fig=plt.figure(figsize=(2*rows, 3*rows))
    for i in range(1, columns*rows +1):
        if i<n_items+2:
            img = Image.open(response[i-1][0][1])
            fig.add_subplot(rows, columns, i)
            plt.imshow(img)
    return plt.show()
```

Joonis 12. Sarnaste piltide tagastamine paisksalvestusest

Sama räsiga piltide vahel teostatakse Hammingu sarnasuse mõõte alusel arvutused ning tagastatakse kindel arv kõige sarnasemat tulemust. Tagastatud tulemusi hindasin kvalitatiivselt. Järgnevatel joonistel (joonis 13, joonis 14, joonis 15) on kirjeldatud asukohatundliku paisksalvestamise tulemused erinevate siirdeõppe mudelitega. Esimene pilt tähistab päringut, millele tagastati sarnased pildid.

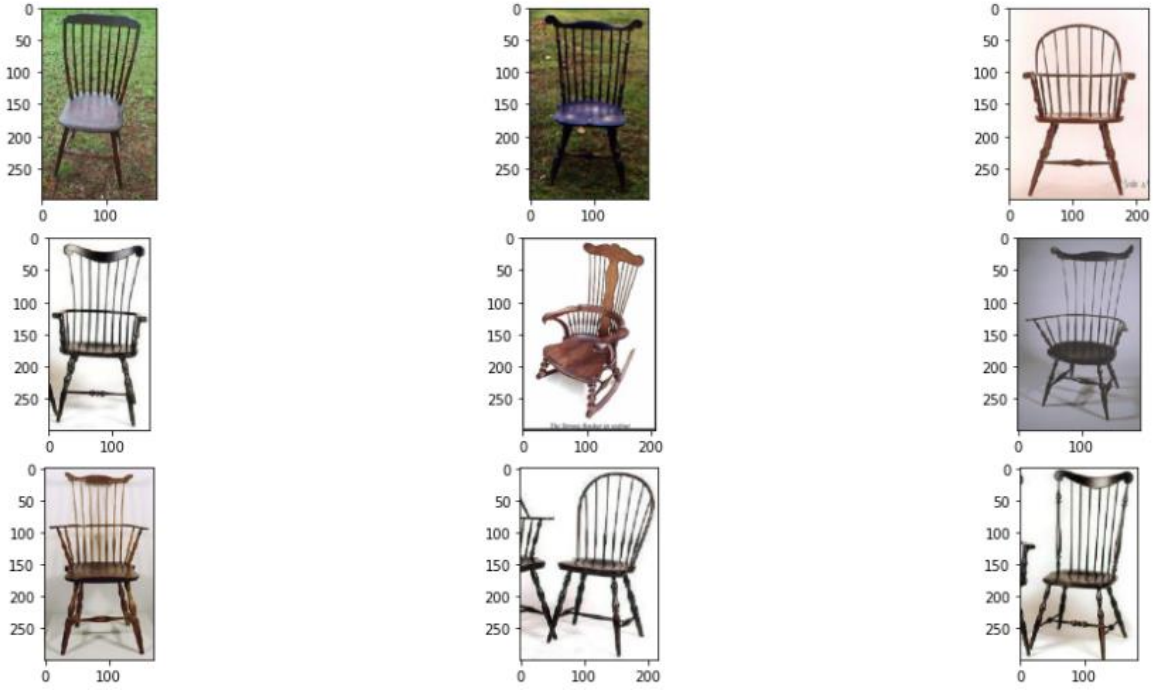


```
elapsed: 1.3005158159999155
```

Joonis 13. ResNet-101 mudeliga eraldatud omadusvektorite sarnasus asukohatundliku paisksalvestamise kasutamisel.

```
elapsed: 1.142730044000018
```

Joonis 14. Inceptionv4 mudeliga eraldatud omadusvektorite sarnasus asukohatundliku paisksalvestamise kasutamisel.



```
elapsed: 1.1904267540001001
```

Joonis 15. DenseNet-121 mudeliga eraldatud omadusvektorite sarnasus asukohatundliku paisksalvestamise kasutamisel.

Visuaalsel hinnangul on sarnasuse leidmine kõikide mudelite puhul edukalt õnnestunud. Inceptionv4 ja DenseNet121 mudelite omadusvektorite alusel tagastati täpselt samad vasted, DenseNet121 puhul oli ajakulu suurem.

3.5 Sarnasuse leidmine graafiandmebaasiga

Teine lähenemine, mida kasutan, on graafiandmebaasis piltide ja nendevaheliste sarnasuse seoste säilitamine. Graafiandmebaasi platvormina kasutasin Neo4J, sest lihtsasti kasutatav töölauarakendus muutis graafiandmebaasidega töötamise ja õppimise mugavaks. Neo4J päringukeel on Cypher, mis võimaldab andmebaasiga läbi viia erinevaid toiminguid. Cypher on inspireeritud SQL päringukeelest ning võimaldab andmetest ja seostest luua visuaalseid mustreid.

Graafiandmebaasi annan sisendiks siirdeõppe käigus eraldatud piltide omadusvektorid ja asukohad, mis on alapeatükis 2.3 salvestatud CSV faili. Pildi kohta luuakse üks sõlm *(node)*, mille omadusteks *(property)* määran omadusvektori ja pildi asukoha, süsteemi poolt antakse igale sõlmele unikaalne ID. CSV failiformaadist andmete laadimisel määratakse alati iga väärtus tekstitüüpi andmetüübiks. Kuna omadusvektorid ei saa sarnasuse arvutamisel olla tekstitüüpi, siis konverteerisin kõik väärtused omadusvektorite massiivides ujukomaarvudeks (*float*). Päring andmete sisse laadimiseks andmebaasi on välja toodud lisas 1. Andmete laadimise tulemusena salvestati andmebaasi 90359 millisekundi jooksul 9119 sõlme.

3.5.1 Sarnasusalgoritmide võrdlemine

Teek *Neo4J Graph Data Science Library* võimaldab kasutada erinevaid sarnasusalgoritme, mis põhinevad vektoripõhistel mõõdikutel. Parima algoritmi valimiseks viisin läbi eksperimendi, kus kasutan kolme sarnasusalgoritmi erinevate sisendpiltide sarnasuse mõõtmiseks. Esimesena viisin läbi eksperimendi samade piltidega (joonis 16), tulemused on kujutatud tabelis 7.

Joonis 16. Sisendpildid sarnasusalgoritmidesse.

Eukleidiline sarnasus	**Koosinus sarnasus**	**Jaccard sarnasus**
0.48000972402615716=48%	0.999994871947683=99.9%	0.000977517106549364 6=0%

Tabel 7. Sarnasusalgoritmide tulemused samade sisendpiltidega

Eksperimendi esimesest osast saab juba järeldada, et koosinus sarnasus on kõige täpsem. Seda oli ka oodata, kuna koosinus sarnasus on ainuke, mis arvutab kauguse multidimensionaalsete vektorite vahel. Viin eksperimendi lõpuni, kasutades sisendpiltideks kaht erinevat, kuid sarnast pilti (joonis 17). Tulemused on nähtavad tabelis 8.

Joonis 17. Sisendpildid sarnasusalgoritmidesse

Eukleidiline sarnasus	**Koosinus sarnasus**	**Jaccard sarnasus**
0.003484548704468167=0.3%	0.720975450006365=72.1%	0.00097751710654936 46=0%

Tabel 8. Sarnasusalgoritmide tulemused erinevate sisendpiltidega

Eksperimendi teine pool kinnitab, et ainuke sobiv algoritm on koosinus sarnasus. Lisaks täpsusele mõõtsin koosinus sarnasuse arvutamise ajakulu, selleks tegin 11 katset. Esimesel katsel laadis päring pilte sisse, seetõttu on esimese katse ajakulu suurim ja jätan selle arvestusest välja. Edasiste 10 katse hulgast arvutasin keskmise ajakulu, milleks on 37,1 millisekundit. Sellest lähtuvalt valin oma töös sarnasuste arvutamiseks koosinus sarnasuse algoritmi.

3.5.2 Sarnasusalgoritmi rakendamine ja kvaliteedi hindamine

Sarnasusalgoritmid jagunevad k lähima naabri algoritmiks, kus luuakse kõikide sõlmede vahel sarnasusseosed ning ligikaudseks k lähima naabri algoritmiks, kus luuakse ligikaudsete sõlmede vahel sarnasusseosed. Koosinus sarnasuse mõõdikut saab kasutada mõlemal juhul.

Algandmete vahel ei ole loodud seoseid, need luuakse sarnasusalgoritmi rakendamise käigus. Kuna antud töö eesmärgi raames ei ole vajalik luua sarnasusseoseid piltide vahel, mille sarnasus on nullilähedane, siis olen algoritmile määranud mõned parameetrid. Sarnasuste arvutamisel salvestatakse ainult 20 kõige sarnasema pildiga sarnasussuhted. Algoritmi kiiremaks töötamiseks olen määranud lävendi, millest allapoole jäävaid sarnasusi ei arvestata. Sarnasusalgoritmi Cypher päring on nähtav joonisel 18.

```
MATCH (p:Image)
WITH {item:id(p), weights: p.vec} as userData
WITH collect(userData) as data
CALL gds.alpha.similarity.cosine.write({
  nodeProjection: '*',
  relationshipProjection: '*',
  topK:20,
  data:data,
  showComputations: true,
  similarityCutoff: 0.1,
  writeRelationshipType: "SIMILAR20"
})
YIELD nodes, similarityPairs, computations
RETURN nodes,
       apoc.number.format(similarityPairs) AS similarityPairs,
       apoc.number.format(computations) AS computations
```

Joonis 18. Sarnasusalgoritmi Cypher päring

Algoritmil kulus aega 75,822 sekundit, mille jooksul salvestati 182 380 seost ning tehti 41 573 521 arvutust. Andmebaasis on nüüd sõlmede vahele loodud seosed, mis kirjeldavad omavahelist sarnasust.

Kuna töö eesmärgiks on leida ühele pildile sarnased pildid, siis on vaja luua veel päring, mis tagastaks ette antud pildiga kõige sarnasemad vasted. Sarnasusseoste salvestamine toimus ainult 20 kõige lähema sõlme jaoks, seega ühe pildi päringuga tagastatakse kõik seosed, millega konkreetne sõlm on ühendatud. Otsingupäring on välja toodud joonisel 19.

Kõikidel sõlmedel on omadus *path*, mis kirjeldab pildi asukohta. CalTech-101 andmekogus on pildid jaotatud 101 kategooria vahel, mille nimi iseloomustab kataloogis olevaid objekte. Otsingupäringu tagastatud vastete kvaliteeti saan kontrollida just kategooria nime järgi, veendudes, et otsingupäringu pilt ning tagastatud piltide asukoht sisaldaks sama nimega kategooriat. Joonisel 19 olen välja toonud päringu, millega ühele pildile tagastatakse kõik sarnased pildid.

```
MATCH (r:Image) WHERE
r.path='/content/101_ObjectCategories/dolphin/image_0042.jpg'
WITH r,
     [(r)-[:SIMILAR20]->(i)| i.path ] AS bruteForce
RETURN bruteForce
```

Joonis 19. Sarnaste piltide otsingu päring sisendpildiga

Päringu tulemusena tagastati 20 sarnast pilti, millest 19 on sisendpildiga samast kategooriast ning üks pilt, millel kujutatakse ahvenat. Kuna ahven on üpris sarnane delfiini pildile, siis saadud tulemust võib lugeda kriteeriumile vastavaks. Ajakulu mõõtmiseks jooksutasin sama otsingu päringut 11korda, millest 10 katse keskmiseks ajakuluks sain 21,3 millisekundit.

Teek *Neo4J Graph Data Science Library* pakub ka ligikaudse lähima naabri algoritmi, millega otsitaks sarnasusseosed vaid ligikaudsete naabritega. See lähenemine nõuab vähem arvutamist ning on seetõttu kiirem, sest omavahel ei võrrelda enam kõiki sõlmesid. Ligikaudse lähima naabri algoritmi puhul võtsin samuti aluseks koosinussarnasuse, päring koos tulemustega on lisas 2. Tulemused ei ole ootuspärased, päringupildiga samast kategooriast ei tagastatud ühtegi pilti, seega ligikaudse lähima naabri algoritm graafiandmebaasis ei ole selle töö kontekstis minu esimene valik.

3.5.3 Graafiandmebaasi protsesside automatiseerimine

Töö eesmärk on luua soovitussüsteem e-poodi, kus toimub pidavalt uute toodete lisamine. Andmete lisamine ja sarnasuste leidmine on siiani olnud manuaalsed protsessid. See tähendab, et iga päringut olen teatud järjekorras käsitsi tööle pannud.

Esmalt kontrollin, kas Neo4j andmebaasides uuendatakse uute andmete lisamisel seoseid. Selleks eemaldasin andmebaasist 4 samast kategooriast pilti. Seejärel laadisin need pildid uuesti andmebaasi ning tegin uute piltidega samast kategooriast pildile otsingu päringu. Saadud tulemuste seas ei olnud ühtki lisatud pilti. Siinkohal saab järeldada, et peale iga uue pildi lisamist andmebaasi peab uuesti käivitama sarnasuse algoritmi ja looma kõikidele uued seosed.

Selle protsessi automatiseerimiseks kasutan Neo4J funktsionaalsust, mis käivitab andmete muutmisel teatud tegevused. Päästiku *(trigger)* idee on andmete lisamisel kutsuda välja sarnasuse seoste lisamine ja uuendamine. Kui lisatav pilt on sarnane andmebaasis oleva pildiga, siis salvestatakse nende vahel seos, kuid seejärel on vaja uuendada kõik andmebaasis olevad seosed.

Esmalt lõin päästiku, mis uute andmete lisamisel kustutab kõik seosed andmebaasist ning lisab uued seosed. Tegin selle protsessi läbi 4 uue pildi lisamisel, päring on toodud lisas 3. Päring võttis aega 113 sekundit. Protsessi saab arvatavasti kiiremaks muuta muu päringuga, kus ei kustutataks seoseid, vaid lihtsalt uuendataks väärtusi, kuid selle töö raames ei suudetud leida ega välja töötada paremat lahendust. E-poes on võimalik kasutada ka praegust loogikat, sest kui laadida uued pildid sisse üks kord ööpäevas kui saidi liiklus on kõige väiksem, siis ei pea kasutama kõige kiiremat lahendust sarnasuste arvutamiseks.

3.5.4 Mudelite tulemused

Sarnaselt asukohatundlikule paisksalvestamisele teen graafiandmebaasi protsessi läbi ResNet-101, Inception ja DenseNet-121 mudelitega ning seejärel võrdlen saadud tulemusi. Kuna andmete sisse laadimine ning sarnasuste arvutamise ajad ei muutunud oluliselt mudelitest olenevalt, siis võrdlen ainult ette antud pildile tagastatud sarnaseid vasteid. Otsingu päring on sama, mis joonisel 19. Kõikide mudelite tulemused tagastavad pildid otsingupildiga samast kategooriast ehk vastavad kvaliteedikriteeriumile.

3.6 Järeldused ja analüüs

Omadusvektorite eraldamiseks proovisin Inception, DenseNet-121 ning 3 erinevat ResNet arhitektuuri. ResNet mudelitest näitas parimat tulemust täpsuses ResNet-101, seega edaspidises protsessis kasutasin ResNet mudelitest ainult ResNet-101 mudelit. Kolme mudeli treenimisel parima tulemuse andis Inception, mille täpsus oli 94,8%.

Kasutasin sarnasuse leidmiseks kaht lähenemist – asukohatundlikku paisksalvestamist ning graafiandmebaasi. Mõlemad meetodid on minu töö kontekstis sobivad ning mõjuvat põhjust üht teisele eelistada ei ole.

Vaatamata sellele, et asukohatundlik paisksalvestamine on ligikaudse lähima naabri algoritm, näitas see meetod olenemata mudelist täpseid tulemusi väga kiire ajaga. Asukohatundlikku paisksalvestamist soovitan kasutada poodides, mille tootevaliku suurus algab 10000 tootest, väiksema tootevaliku puhul ei pruugi ligikaudse lähima naabri algoritm anda rahuldavaid tulemusi. Väiksema tootevalikuga poodides soovitan kasutada graafiandmebaasi ja lähima naabri algoritmi, sest see meetod annab kõige täpsemad vasted ning ajakulu on veel võrdlemisi väike.

3.7 Edasised arendused

Töö käigus välja töötatud lahendusi saaks kindlasti optimiseerida paremaks. Praegu on mõistlik toodete sarnasusmõõte arvutada taustal kindla aja tagant. Ühe ideena võiks graafiandmebaasi protsessi automatiseerimise ajakulu saada väiksemaks, et uute toodete e-poodi lisamine saaks toimuda jooksvalt.

Antud töö raames leidsin lahenduse, kuidas soovitussüsteem peaks töötama, kuid kindlasti järgmine samm on panna välja töötatud lahendus ühilduma e-poe platvormiga. Üks võimalus selleks on arendada moodul, mis suhtleks platvormiga API kaudu.

4 Kokkuvõte

E-kaubanduses on oluline sisu personaliseerimine, mis väljendub kliendile relevantsete toodete pakkumises. Töö eesmärgiks oli luua soovitussüsteem, mis keskenduks piltide sarnasuste leidmisele, et kaardistada piltide omavahelised sarnasussuhted ning nende põhjal leida etteantud pildile kindel hulk sarnaseid vasteid. Piltide sarnasuste leidmiseks kaaluti kaht võimalust - siiami närvivõrkude kasutamine või erinevate konvolutsiooniliste mudelite abil omadusvektorite eraldamine ja sarnasuse tuvastamiseks sarnasusalgoritmide rakendamine. Alustati eeltreenitud mudelite kasutamisega, selle meetodiga saadud tulemused olid kvaliteetsed, seega töö raames ei ole kasutatud siiami närvivõrke.

Konvolutsioonilised närvivõrgud sobivad piltide analüüsimiseks ja äratundmiseks. Piltide omavaheliseks võrdlemiseks sarnasuse leidmisel on tarvis need teisendada numbrilisteks omadusvektoriteks. Omadusvektorite eraldamiseks kasutati siirdeõppe meetodit, mille käigus treeniti mudelid treeningandmestikuga üle. Treenimisel optimaalse õpisammu leidmiseks kasutati funktsiooni, mis erinevate õpisammudega joonistas välja graafiku. Töö eesmärgi saavutamiseks parima konvolutsioonilise närvivõrgu leidmiseks viidi siirdeõppe meetod läbi ResNet-18, ResNet-34, ResNet-101, Inception ning DenseNet-121 mudelitega. Kõik mudelid andsid testandmestikul üsna sarnaseid tulemusi, kuid parima täpsusega tulemusi näitas Inception mudel 94,8%.

Sarnaste piltide leidmiseks analüüsiti asukohatundlikku paisksalvestamist ning graafiandmebaasi. Mõlemad meetodid näitasid kiiruses ja täpsuses häid tulemusi. Graafiandmebaasi puhul tagastati sarnased pildid keskmise ajakuluga 21,3 millisekundit, asukohatundliku paisksalvestamise puhul keskmise ajakuluga 1,4 sekundit. Saadud tulemusi hinnati kvantitatiivselt ette antud pildi kategooria järgi. Graafiandmebaasi puhul õnnestus automatiseerida kogu protsess andmete laadimisest sarnaste piltide leidmiseni. Töö tulemusena on mõlemad meetodid aktsepteeritud kui töötavad lahendused sarnaste piltide leidmiseks. Autor on soovitanud kasutada kumbagi lahendust vastavalt tootevaliku suurusele.

Töö käigus loodud lahendus on e-poe platvormiga integreerimiseks valmis.

Kasutatud kirjandus

[1] The 2017 State of Personalization Report. Segment. http://grow.segment.com/Segment-2017-Personalization-Report.pdf (19.03.2020)

[2] 2019 Trends in Personalization Report. Evergage. https://www.evergage.com/wp-content/uploads/2019/04/2019_Trends_in_Personalization_Report.pdf (21.03.2020)

[3] Nosto hinnakiri. https://www.nosto.com/pricing/ (30.03.2020)

[4] Klevu demokoosolek. Autori märkmed. Üleskirjutus.

[5] STACC soovitussüsteem. https://www.stacc.ee/soovitussusteemid-kellele-soovitatakse-ja-mida/ (30.03.2020)

[6] Overview: Extracting and serving feature embeddings for machine learning. Google Cloud. https://cloud.google.com/solutions/machine-learning/overview-extracting-and-serving-feature-embeddings-for-machine-learning (23.04.2020)

[7] L. Deng, "Deep Learning: Methods and Applications," Found. Trends® Signal Process., vol. 7, no. 3–4, pp. 197–387, 2014.

[8] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," p. 22.

[9] "ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks," CV-Tricks.com, Aug. 09, 2017. https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/ (22.04.2020)

[10] C. Szegedy et al., "Going deeper with convolutions," in 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), USA, Jun. 2015, pp. 1–9.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," ArXiv151203385 Cs, Dec. 2015. http://arxiv.org/abs/1512.03385 (27.04.2020)

[12] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in 2016 23rd International Conference on Pattern Recognition (ICPR), Dec. 2016, pp. 378–383.

[13] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A Survey on Deep Transfer Learning,". http://arxiv.org/abs/1808.01974 (21.04.2020)

[14] Q. Yang, Y. Zhang, W. Dai, and S. J. Pan, Transfer Learning. Cambridge University Press, 2020.

[15] J. Brownlee, “4 Distance Measures for Machine Learning,” Machine Learning Mastery. https://machinelearningmastery.com/distance-measures-for-machine-learning/ (26.04.2020)

[16] “Cosine Similarity,” DeepAI. https://deepai.org/machine-learning-glossary-and-terms/cosine-similarity (28.04.2020)

[17] G. Yona, “Fast Near-Duplicate Image Search using Locality Sensitive Hashing,” Medium, May 05, 2018. https://towardsdatascience.com/fast-near-duplicate-image-search-using-locality-sensitive-hashing-d4c16058efcb (26.04.2020).

[18] K. Kise, K. Noguchi, and M. Iwamura, “Simple Representation and Approximate Search of Feature Vectors for Large-Scale Object Recognition,” in Procedings of the British Machine Vision Conference 2007, Warwick, 2007, pp. 37.1-37.10.

[19] C. Safka, christiansafka/img2vec. https://github.com/christiansafka/img2vec (30.03.2020).

[20] “Deep_learning_explorations/8_Image_similarity_search at master · aayushmnit/Deep_learning_explorations.” https://github.com/aayushmnit/Deep_learning_explorations/tree/master/8_Image_similarity_search (06.05.2020).

[21] L. Fei-Fei, R. Fergus and P. Perona. Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories. IEEE. CVPR 2004, Workshop on Generative-Model Based Vision. 2004. http://www.vision.caltech.edu/Image_Datasets/Caltech101/ (05.04.2020).

[22] “PPPW/deep-learning-random-explore,” GitHub. https://github.com/PPPW/deep-learning-random-explore (05.05.2020)

[23] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” ArXiv160806993 Cs, Jan. 2018. http://arxiv.org/abs/1608.06993 (06.05.2020)

LISA 1 – Cypher päring andmete laadimiseks

Cypher päring andmete laadimiseks graafiandmebaasi CSV failist. Vektori massiivi tekstitüüpi väärtuste konverteerimine numbrilisteks väärtusteks.

```
LOAD CSV FROM 'file:///output7.csv' AS row FIELDTERMINATOR ']'
WITH row[0] AS vec, row[1] AS path
MERGE (i:Image {path: path})
  SET i.path= path, i.vec = split(vec, ',')
  SET i.vec= [item in i.vec | apoc.convert.toFloat(item)]
RETURN count(i)
```

LISA 2 – Ligikaudse lähima naabri algoritm graafiandmebaasis

Ligikaudse lähima naabri algoritmi Cypher päring ning ette antud pildile tagastatud vasted. Tulemused ei ole ootuspärased.

```
MATCH (p:Image)
 WITH {item:id(p), weights: p.vec} AS userData
 WITH collect(userData) AS data
 CALL gds.alpha.ml.ann.write({
   nodeProjection: '*',
   relationshipProjection: '*',
   data: data,
   topK:20,
   algorithm: 'cosine',
   writeRelationshipType:"SIMILAR_APPROX",
   similarityCutoff: 0.1,
   concurrency: 1
 })
 YIELD nodes, similarityPairs, computations
RETURN nodes
```

Ette antud pilt, millele otsitakse sarnaseid

Tagastatud sarnased pildid

LISA 3 – Protsessi automatiseerimiseks Cypher käsk

Graafiandmebaasi protsessi automatiseerimiseks Cypher käsk, mis uute andmete lisamisel kustutab olemasolevad seosed andmebaasist ning loob sarnasusuhete alusel uued seosed.

```
CALL apoc.trigger.add('CRS',"UNWIND {createdNodes} AS node
MATCH (p:Image)-[r:SIMILAR20]->()
Delete r
WITH collect({item:id(p), weights:p.vec}) as data
CALL gds.alpha.similarity.cosine.write({
nodeProjection: '*',
  relationshipProjection: '*',
  topK:20,
  data:data,
  showComputations: true,
  writeRelationshipType:'SIMILAR20'
})
YIELD nodes, similarityPairs, computations
RETURN nodes,
       apoc.number.format(similarityPairs) AS similarityPairs,
       apoc.number.format(computations) AS computations",
{phase:'after'})
```