



**Kauno technologijos universitetas**

Informatikos fakultetas

# **Sistema miškų analizavimui naudojantis dirbtiniu neuroniniu tinklu individualiems medžiams aptikti iš palydovų nuotraukų**

Baigiamasis bakalauro studijų projektas

---

**Dainius Šaltenis**

Projekto autorius

---

**Lekt. Kazimieras Bagdonas**

Vadovas

---

**Kaunas, 2019**



**Kauno technologijos universitetas**

Informatikos fakultetas

# **Sistema miškų analizavimui naudojantis dirbtiniu neuroniniu tinklu individualiems medžiams aptikti iš palydovų nuotraukų**

Baigiamasis bakalauro studijų projektas

Programų sistemos (612I30002)

---

**Dainius Šaltenis**

Projekto autorius

**Lekt. Kazimieras Bagdonas**

Vadovas

**Dr. Mantas Lukoševičius**

Recenzentas

---

**Kaunas, 2019**



**Kauno technologijos universitetas**

Informatikos fakultetas

Dainius Šaltenis

## **Sistema miškų analizavimui naudojantis dirbtiniu neuroniniu tinklu individualiems medžiams aptiki iš palydovų nuotraukų**

Akademinio sąžiningumo deklaracija

Patvirtinu, kad mano, Dainiaus Šaltenio, baigiamasis projektas tema „Sistema miškų analizavimui naudojantis dirbtiniu neuroniniu tinklu individualiems medžiams aptiki iš palydovų nuotraukų“ yra parašytas visiškai savarankiškai ir visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymu nenumatyta piniginių sumų už šį darbą niekam nesu mokėjės.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

---

(vardą ir pavardę įrašyti ranka)

---

(parašas)

Šaltenis, Dainius. Sistema miškų analizavimui naudojantis dirbtiniu neuroniniu tinklu individualiems medžiams aptikti iš palydovų nuotraukų. Bakalauro studijų baigiamasis projektas vadovas lekt. Kazimieras Bagdonas; Kauno technologijos universitetas, Informatikos fakultetas.

Studijų kryptis ir sritis (studijų krypčių grupė): Informatikos mokslai, Programų sistemos.

Reikšminiai žodžiai: medžių viršūnių paieška palydovų nuotraukose, konvoluciiniai neuroniniai tinklai, kompiuterinė rega, gilus mokymasis, dirbtinis intelektas.

Kaunas, 2019. 62 p.

### **Santrauka**

Šiame darbe pristatomas kompiuterinės regos algoritmas individualių medžių viršūnių aptikimui palydovų darytose nuotraukose ir vartotojo sąsajos programa šiam algoritmui naudoti bei jo rezultatams analizuoti, siekiant gauti informaciją apie medžius miškuose, miestuose bei kitose teritorijose. Įvado dalyje aprašomos priežastys tokio tipo sistemai kurti bei iškeliami darbo uždaviniai. Analizės dalyje nagrinėjami kiti siūlomi tokio tipo sprendimai bei techninės galimybės išgyvendinti giliojo mokymo principais grįstą medžių viršūnių aptikimo algoritmą. Projekto dalyje aprašomas kompiuterinės regos algoritmas, Jame naudojamas konvoluciinis neuroninis tinklas, vartotojo sąsaja ir visos bendros sistemos veikimas. Testavimo dalyje aprašomas testavimo planas. Rezultatų dalyje nurodomi eksperimentiniai bei galutiniai kuriamo kompiuterinės regos algoritmo rezultatai. Dokumentacijos dalyje aprašomas šios sistemos paleidimas. Rezultatų ir išvadų skyriuje pateikiamos šio darbo išvados.

Šaltenis, Dainius. A Neural Network System for Individual Tree Detection in Satellite Imagery. Bachelor's Final Degree Project / supervisor lect. Kazimieras Bagdonas; Informatics Faculty, Kaunas University of Technology.

Study field and area (study field group): Computer Sciences, Software Systems.

Keywords: detection of trees in satellite imagery, convolutional neural networks, computer vision, deep learning, artificial intelligence.

Kaunas, 2019. 62.

## **Summary**

This paper presents a computer vision algorithm for individual tree detection in satellite imagery and user interface application for usage of this computer vision algorithm and analysis of its results, with a goal of extracting information about trees in forests, cities and other territories. Introductory part consists of reasoning for development of this system and goals for this work are described. In analysis part current similar systems and algorithms present in the market and technical capabilities for development of such system are analyzed. In project section a computer vision algorithm, convolutional neural network, user interface application and behavior of a whole system is described. Testing plan and process is presented in testing part. In results part the test results of intermediate and complete versions of computer vision algorithm are presented. Documentation part consists of instruction about the system deployment and usage. In results and conclusions part the conclusions of this work are presented.

## Turinys

<b>Lentelių sąrašas .....</b>	<b>7</b>
<b>Paveikslų sąrašas .....</b>	<b>8</b>
<b>Santrumpū ir terminų sąrašas .....</b>	<b>9</b>
<b>Įvadas.....</b>	<b>10</b>
<b>    1. Analizė .....</b>	<b>11</b>
1.1. Techninis pasiūlymas .....	11
1.1.1. Sistemos apibréžimas .....	11
1.1.2. Bendras veiklos tikslas .....	11
1.1.3. Sistemos pagrįstumas .....	11
1.1.4. Konkurencija rinkoje .....	12
1.1.5. Prototipai ir pagalbinė informacija .....	13
1.1.6. Ištekliai, reikalingi sistemai sukurti.....	14
1.2. Galimybių analizė.....	15
1.2.1. Techninės galimybės .....	15
1.2.2. Vartotojų pasiruošimo analizė .....	15
<b>    2. Projektas.....</b>	<b>16</b>
2.1. Reikalavimų specifikacija .....	16
2.1.1. Komercinė specifikacija .....	16
2.1.2. Sistemos funkcijos.....	16
2.1.3. Vartotojo sąsajos specifikacija .....	18
2.1.4. Realizacijai keliami reikalavimai .....	18
2.1.5. Techninė specifikacija .....	20
2.2. Projektavimo metodai.....	20
2.2.1. Projektavimo valdymas ir eiga .....	20
2.2.2. Projektavimo technologija.....	20
2.2.3. Programavimo kalbos, derinimo, automatizavimo priemonės, operacinė sistemos.....	20
2.3. Sistemos projektas .....	21
2.3.1. Statinis sistemos vaizdas .....	21
2.3.2. Dinaminis sistemos vaizdas.....	27
<b>    3. Testavimas.....</b>	<b>41</b>
3.1. Testavimo planas .....	41
3.2. Testavimo kriterijai .....	41
3.3. Komponentų testavimas .....	42
3.4. Vizualus testavimas .....	42
3.5. Vartotojo sąsajos testavimas.....	49
<b>    4. Neuroninių tinklų apmokymo rezultatai.....</b>	<b>52</b>
<b>    5. Dokumentacija naudotojui .....</b>	<b>54</b>
5.1. Apibendrintas sistemos galimybių aprašymas.....	54
5.2. Vartotojo vadovas.....	54
5.3. Diegimo vadovas .....	57
5.4. Administravimo vadovas.....	57
<b>Rezultatai ir išvados .....</b>	<b>59</b>
<b>Literatūros sąrašas .....</b>	<b>60</b>

## Lentelių sąrašas

<b>1 lentelė.</b> Konkurentų apžvalga.....	13
<b>2 lentelė.</b> Konvoluciinių neuroninių tinklų palyginimas.....	14
<b>3 lentelė.</b> 2.5 pav. pateiktos konvoluciinio algoritmo diagramos blokų spalvų bei anotacijų paaiškinimas.....	26
<b>4 lentelė.</b> Kairėje: modeliui apsimokymui pateikiamos nenormalizuotos nuotraukos pavyzdys; Dešinėje: kairėje pateiktai nuotraukai sugeneruoto apmokymui skirto karščio žemėlapio pavyzdys, išreiškiantis medžių viršūnių centrų koordinates. Nuotraukos šaltinis: Google Earth.....	30
<b>5 lentelė.</b> Kairėje viršuje: originali įvesties nuotrauka; Dešinėje viršuje: neapdorotos karščio žemėlapio išvestys pateiktai įvesties nuotraukai; Kairėje apačioje: dekoduotos išvestys pateiktai įvesties nuotraukai neatlikus filtravimo pagal apskritimų spindulių persidengimus; Dešinėje apačioje: pilnai apdorotos išvestys. Nuotraukos šaltinis: Google Earth.....	34
<b>6 lentelė.</b> 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų karpymo etapo.....	43
<b>7 lentelė.</b> 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų augmentacijos etapo.....	44
<b>8 lentelė.</b> 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų filtravimo etapo.....	45
<b>9 lentelė.</b> 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų įvesties karščio žemėlapiai generavimo etapo.....	46
<b>10 lentelė.</b> 3.2 pav. pateiktos pavyzdinės nuotraukos anotacijų spindulių bei 4 pikselių paklaidos reikšmės po nuotraukos apdorojimo apmokymo įvesčiai.....	47
<b>11 lentelė.</b> Apmokyto neuroninio tinklo išvestys pavyzdinėms nuotraukoms. Pirmoje eilutėje pateikiamos dekoduotos išvestys, antroje- atitinkamoms pirmos eilutės nuotraukoms apskaičiuotos karščio žemėlapio reikšmės, trečioje- vizualizuotos spindulių reikšmės bei ketvirtuoje- vizualizuotos 4 pikselių paklaidų reikšmės. Naudojamos pavyzdinės nuotraukos iš <i>Google Earth</i> programos....	47
<b>12 lentelė.</b> Apmokyto neuroninio tinklo klaidingos išvestys. Kairėje- nestabilios ir netolygios išvestys, dėl kurių negalima tiksliai prognozuoti medžio viršūnės centro. Dėsinėje- atsitiktiniai maži, tačiau aukštas reikšmės turintys taškai, kurie dekoduojami grąžina netikslias ( <i>false positive</i> ) medžių viršūnių reikšmes.....	49
<b>13 lentelė.</b> Apmokytyų neuroninių tinklų medžių viršūnių aptikimo tikslumo rezultatai visų dydžių medžiams.....	52
<b>14 lentelė.</b> Apmokytyų neuroninių tinklų medžių viršūnių aptikimo greičio rezultatai naudojant Nvidia RTX 2060 vaizdo plokštę. Laikas skaičiuojamas sekundėmis vienai nuotraukai apdoroti nuo įvesties pateikimo neuroniniam tinklui iki dekoduotų išvesčių gavimo.....	52
<b>15 lentelė.</b> Neuroninių tinklų parametru kiekis su <i>FPN</i> tinklo dalimi.....	52

## Paveikslų sąrašas

<b>2.1 pav.</b> Sistemos panaudojimo atvejų (use case) UML diagrama vartotojo sąsajos programai.....	17
<b>2.2 pav.</b> Sistemos komponentų UML diagrama.....	21
<b>2.3 pav.</b> Kompiuterinės regos algoritmo programos paketų UML diagrama.....	22
<b>2.4 pav.</b> Grafinės vartotojo sąsajos programos klasių UML diagrama.....	24
<b>2.5 pav.</b> Kompiuterinės regos algoritme naudojamo konvolucinio neuroninio tinklo struktūra, kai įvesties matrica dydis yra 1024x1024x3 dydžio. Pavyzdinės įvesties nuotraukos šaltinis: Google Earth.....	25
<b>2.6 pav.</b> Konvolucinio neuroninio tinklo apmokymo veiksmų diagrama.....	28
<b>2.7 pav.</b> Nuotraukos apdorojimo medžių viršūnėms rasti veiksmų UML diagrama.....	32
<b>2.8 pav.</b> Nuotraukos užkrovimų iš failų sistemos vartotojo sąsajoje UML sekų diagrama.....	35
<b>2.9 pav.</b> Vartotojo sąsaja su užkrautomis nuotraukomis. Nuotraukos šaltinis: Google Earth.....	36
<b>2.10 pav.</b> Duomenų užkrovimo iš „.json“ formato failo UML sekų diagrama.....	37
<b>2.11 pav.</b> Duomenų saugojimo „.json“ formatu UML sekų diagrama.....	37
<b>2.12 pav.</b> Nuotraukų apdorojimo su kompiuterinės regos algoritmu medžių viršūnėms rasti proceso UML sekų diagrama.....	38
<b>2.13 pav.</b> Nuotraukų apdorojimo su kompiuterinės regos algoritmu medžių viršūnėms rasti rezultatai, pateikiami peržiūrai aplikacijoje. Nuotraukos šaltinis: Google Earth.....	39
<b>2.14 pav.</b> Informacijos vaizdavimo analizės lange UML sekų diagrama.....	39
<b>2.15 pav.</b> Nuotraukų analizavimas laike raudonais apskritimais žymint potencialiai nukirstų medžių vietas. Nuotraukos pavyzdys: Google Earth.....	40
<b>3.1 pav.</b> Komponentų testavimo įvykdymo rezultatas.....	42
<b>3.2 pav.</b> Duomenų užkrovimo vizualizacija.....	43
<b>3.3 pav.</b> Vartotojo sąsajos vizualizacija. Kairėje- nuotraukų aprašų skirtukas, kuriame galima pasirinkti nuotraukas ir keisti nuotraukos parametrus. Kairėje apačioje- pagrindinių skirtuko funkcijų juosta. Viršuje- svarbiausių funkcijų antraštės juosta. Naudojama pavyzdinė palydovo atlakta nuotrauka iš <i>Google Earth</i> programos.....	50
<b>3.4 pav.</b> Vartotojo sąsajos vizualizacija. Nuotraukoje vaizduojamas analizės langas bei kompiuterinės regos algoritmo teikiami rezultatai konkretčiai miesto nuotraukai iš viršaus. Naudojama pavyzdinė palydovo atlakta nuotrauka iš <i>Google Earth</i> programos.....	51
<b>5.1 pav.</b> Vartotojo sąsajos antraštės mygtukų juosta su svarbiausiomis funkcijomis.....	54
<b>5.2 pav.</b> Vartotojo sąsajos nuotraukų aprašų stulpelis.....	55
<b>5.3 pav.</b> Vartotojo sąsajos nuotraukų aprašų stulpelis su keliomis nuotraukomis vienoje nuotraukų skiltyje.....	56
<b>5.4 pav.</b> Vartotojo sąsajos klaidos pranešimo pavyzdys.....	56
<b>5.5 pav.</b> Vartotojo sąsajos pagrindinio lango apačioje esantis mygtukų rinkinys.....	56
<b>5.6 pav.</b> Vartotojo sąsajos analizės langas.....	57

## **Santrumpų ir terminų sąrašas**

### **Santrumpos:**

FPN- Feature Pyramid Network

API- Application Programming Interface

IOU- Intersection Over Union

### **Terminai:**

Feature Pyramid Network- neuroninio tinklo struktūra, kurioje naudojantis dekonvoluciujos operacija ir tarpinėmis išvestimis iš stuburinio neuroninio tinklo apdorojamos tinklo išvestys.

Duomenų augmentacija- atsitiktinis apmokymo duomenų modifikavimas siekiant padidinti duomenų rinkinio apimtį.

Recall- (jautrumas) dalis duomenų rinkinio duomenų, kuri buvo teisingai prognozuota.

Precision- (specifiškumas) dalis algoritmo prognozavimo rezultatų, kuri buvo teisinga.

Intersection Over Union- dvejų objektų ploto persidengimo dalis, gauta persidengimo plotą padalinant iš bendro abiejų objektų ploto.

## **Įvadas**

Šiame darbe kuriama sistema, skirta leisti paprastam vartotojui naudojantis giliojo mokymo principais sukurtu algoritmu apdoroti palydovą darytas žemės nuotraukas, aptinkant jose individualias medžių viršunes bei analizuojant po apdorojimo gautą informaciją. Sistema susideda iš kompiuterinės regos algoritmo, kuris apdoroja informaciją naudodamas apmokytu konvoliuciniu neuroniniu tinklu bei vartotojo sėsajos programos šiam algoritmui naudoti bei analizuoti gautai informacijai. Šiuo darbu siekiamas įvertinti konvoliucinių neuroninių tinklų potencialą greitai ir tiksliai aptikti medžių viršunes palydovą ar bepiločių orlaivių darytose nuotraukose bei galimybę patogiai naudotis tokiu konvoliuciniu neuroniniu tinklu bet kokiam vartotoju, siekiančiam gauti informaciją apie medžius miškuose, miestuose ar kitose teritorijose.

Medžių aptikimo palydovų nuotraukose tema yra ilgą laiką nagrinėjama akademinėje ir komercinėje aplinkoje. Sékmingai funkcionuojanti tokio tipo programa su papildomomis galimybėmis leistų potencialiai aptikti kirtimus, žalą po audros, gauti statistinę informaciją apie medžius konkrečiose teritorijose, stebėti jų būklę, įvertinti miško kainą ir kt. Sékmingas tokio kompiuterinės regos algoritmo sukūrimas ir pritaikymas suteiktų didelę visuomeninę bei komercinę naudą.

Šiame darbe siekiamas sukurti tikslų ir greitą giliojo mokymo kompiuterinės regos algoritmą individualių medžių viršunių aptikimui palydovų nuotraukose bei sukurti vartotojo sėsają darbui su šiuo algoritmu. Darbą sudaro šios užduotys:

1. Atliliki viešai pateikiamų konvoliucinių neuroninių tinklų analizę išsiaiškinant, kokio tipo stuburinis neuroninis tinklas ir neuroninio tinklo komponentai geriausiai veikia medžių viršunių aptikimo užduočiai;
2. Sukurti naują konvoliucinį neuroninį tinklą medžių aptikimo tikslumui arba greičiui pagerinti pagal antroje užduotyje gautus rezultatus ir iš jų sudarytas išvadas;
3. Sukurti vartotojo sėsajos aplikaciją, leidžiančią patogiai atliliki paprastas analizavimo užduotis naudojantis šiuo algoritmu;
4. Išsiaiškinti, kokios yra šio algoritmo stipriosios bei silpnosios savybes, kokioje aplinkoje jis veikia prastai ir gerai bei kokie turėtų būti potencialūs tolimesni žingsniai jo tobulinimui.

Šio rašto darbo pirmame skyriuje pateikiama analizė bei pagrindimas tokio tipo kompiuterinės regos algoritmo poreikiui bei analizuojami esami įgyvendinti sprendimai. Antrame skyriuje aprašoma sistema, aprašant jos komponentų, kompiuterinės regos algoritmo, konvoliucinio tinklo bei vartotojo sėsajos programos statinį bei dinaminį, vaizdus. Trečiame skyriuje pateikiamas sistemos testavimas, ketvirtame - dokumentacija ir naudojimas. Darbo pabaigoje pateikiami šiame darbe sukurto konvoliucinio neuroninio tinklo, prijungto prie medžių viršunių aptikimo algoritmo, bei kitų išbandytų tinklų veikimo rezultatai bei išvados apie visos sistemos veikimą ir jos kūrimo procesą.

Kuriant šią sistemą buvo praleista apie 250 valandų analizuojant medžių aptikimo algoritmą, pritaikant rinkoje esamus objektų aptikimo algoritmus bei neuroninius tinklus, kuriant naują neuroninį tinklą, analizuojant rezultatus, analizuojant algoritmo daromas klaidas, koreguojant apmokymo procesą bei modelio architektūrą ir žymint apmokymo bei testavimo duomenis. Dar apie 80 valandų praleista kuriant vartotojo sėsajos programą bei jos komunikaciją su kompiuterinės regos algoritmo programa.

## **1. Analizė**

### **1.1. Techninis pasiūlymas**

#### **1.1.1. Sistemos apibrėžimas**

Sistema miškų analizavimui naudojantis dirbtiniu neuroniniu tinklu individualiems medžiams aptikti iš palydovų nuotraukų- tai sistema, kuri naudodamas šiai užduočiai sukurtu kompiuterinės regos algoritmu apdoroja vartotojo pateiktas aukštostos kokybės palydovų nuotraukas, jose automatiškai suranda medžių viršunes ir vartotojui pateikia duomenis apie individualius medžius bei statistinę jų informaciją vartotojo sasajoje.

#### **1.1.2. Bendras veiklos tikslas**

Šio darbo tikslas yra sukurti kompiuterinės regos algoritmą, kuo tiksliau aptinkantį medžių viršunes aukštostos kokybės nuotraukose, nufotografuotose iš palydovų ar dronų, bei aplikaciją su vartotojo sasaja šio algoritmo naudojimui ir rezultatų vizualizacijai. Siūlomas kompiuterinės regos algoritmas yra apmokomas planetos nuotraukomis iš palydovų, kuriose rankiniu būdu sužymėtos medžių viršūnės, aptikimui naudoja konvoluciinį neuroninį tinklą, o apmokytas aptinka bei pateikia medžių pozicijas bei dydžio spindulius nuotraukose. Siekiama ištirti, kokie rinkoje naudojami bei šiam projektui kuriami dirbtinio intelekto sprendimai pateikia geriausius rezultatus šiai užduočiai, kokiems panaudojimo atvejams tinka šio kompiuterinės regos algoritmo pritaikymas, bei ar ši dirbtinio intelekto algoritmą yra paprasta naudoti vartotojui, neturinčiam plačią informacinių technologijų žinių. Naudojantis medžių viršūnių aptikimo algoritmo pateiktais rezultatais galima apytiksliai apskaičiuoti statistinę informaciją apie medžius. Siekiama išsiaiškinti, ar kuriamo kompiuterinės regos algoritmo rezultatai tinkami tiksliai apskaičiuoti statistinę informaciją, pavyzdžiuui miško tankį, medžių uždengto ploto dalį, ar informaciją tam tikrame laiko spektre, kuri potencialiai leistų aptikti kirtimus, žalą po audrų. Šis algoritmas leistų greitai gauti apytikslę informaciją apie medžius ir jų masyvus įvairose teritorijose nereikalaujant fizinio buvimo bei žmogaus darbo atliekant skaičiavimus bei matavimus. Medžių aptikimo palydovų darytose nuotraukose problema yra ilgai nagrinėjama akademinėje bei komercinėje aplinkoje, pilnas ir sėkmingas tokio kompiuterinės regos algoritmo išvystymas atneštų visuomeninę bei komercinę naudą. Šiame darbe kuriant giliojo mokymo kompiuterinės regos algoritmą įvertinamas konvoluciinių neuroninių tinklų pritaikymas šiai užduočiai, gautų modelių tikslumas įvairiems pritaikymo atvejams. Šiam algoritmui naudoti sukurta grafinė vartotojo sasaja leidžia įvertinti šio algoritmo praktinio naudojimo galimybes.

#### **1.1.3. Sistemos pagrįstumas**

Dauguma informacijos apie konkrečius medžius bei jų masyvus yra apskaičiuojama naudojantis žmonių darbu. Žmonėms reikia keliauti ilgus atstumus, patikrinti didelius plotus, to finansinės išlaidos yra didelės bei tai užtrunka ilgai. Siūlomo sprendimo sėkmingas pritaikymas potencialiai leistų šį procesą automatizuoti bei nereikalautų fizinio žmogaus buvimo tiriamoje vietoje. Naudojantis rastų individualių medžių duomenimis galima sekti pokyčius laike, pavyzdžiuui rasti nukirstus medžius miestuose, ieškoti nelegaliai nukirstų medžių miško masyvuose, stebeti audros sukeltą žalą, taip nereikalaujant žmogui fiziškai nuolatos keliauti po tiriamas teritorijas ir meginti aptiki pokyčius. Naudojantis tais pačiais duomenimis galima apskaičiuoti statistiką, pavyzdžiuui miškų duomenis, tokius kaip mišrumas, tankis arba medienos tūris, miškų kiekis miestuose. Tai leistų miškų prižiūrėtojams, verslininkams ar urbanistams daug greičiau gauti apytikslę informaciją,

nereikalaujant atliskti įvairiausius skaičiavimus ar stebėjimus. Netgi ne itin tikslūs algoritmo rezultatai gali susiaurinti stebimos teritorijos kiekį ir leisti tam tikroms užduotims medžių stebėjimus perkelti iš realios erdvės į skaitmeninę. Kadangi šis siūlomas algoritmas sprendžia abstrakčią ir ne galutinę problemą, jo galutinius rezultatus riboja kūrybiškumas, rinkos poreikis, pritaikymo galimybės, tikslumas ir greitis.

#### 1.1.4. Konkurencija rinkoje

Tyrimai individualių medžių viršunių paieškoje palydovo nuotraukose vyksta ilgai, tačiau šiuo metu rinkoje vis dar nėra daug tokio tipo paslaugų, programų ar algoritmų. Dauguma siūlomų sprendimų yra pateikiami kaip mokslinės publikacijos, neretai be papildomų priedų (duomenų rinkinių, galutinio algoritmo programinio kodo), tai apsunkina šių eksperimentų atkūrimą ir integravimą, bei nepateikia pakankamai aukštų rezultatų, kurie tiktų komerciniam pritaikymui. Nemažai sprendimų naudoja primityvius kompiuterinės regos algoritmus (1, 2, 3), kurie nėra apmokomi duomenimis, todėl jų veikimas gali smarkiai skirtis pakitus geografinei pozicijai, palydovo nuotraukų kokybei, fotografavimo kryptčiai, oro sąlygoms, metų laikams ir kt. Nemažai daliai sprendimų pateikiami testavimo rezultatai taip pat yra apriboti konkrečiai situacijai, geografinei vietai ar tik konkrečiai nuotraukai (2, 4, 5). Taip pat egzistuoja daug sprendimų bei duomenų rinkinių medžiaisiais apaugsių teritorijų analizei kaip abstrakčiam masyvui, nesigilinant į konkrečius tą teritoriją sudarančius objektus, miško sudėtį bei ją sudarančių objektų savybes (6, 7). Tokio tipo sprendimai bei iš šių duomenų sukurti duomenų apdorojimo modeliai neanalizuoją konkrečių mišką sudarančių objektų, todėl jų galimybės yra apribotos tiek, kiek abstraktūs yra naudojami apmokymo duomenys.

Šiuo metu rinkoje yra pateikiami šie patrauklūs sprendimai:

**1 sprendimas.** Mokslinėje publikacijoje „Individual Tree-Crown Detection in RGB Imagery Using Semi-Supervised Deep Learning Neural Networks“ (8) pateikiamas giliuoju mokymusi grįstą sprendimas, kuriam dalis naudojamų apmokymo duomenų yra nesužymėti. Algoritmo kūrėjai taip pat pateikia eksperimento programinio kodo implementaciją. Nors ir algoritmo rezultatai yra geri, šio aprašyto algoritmo testavimas vykdomas duomenų rinkiniui, turinčiam 271 medžių viršunių. Tokio dydžio testavimo duomenų rinkinio dydis neleidžia tinkamai įvertinti algoritmo tikslumo, įrodo algoritmo galimybes tik labai konkrečiai panaudojimo situacijai. Algoritmas yra pritaikytas veikti tik konkretaus tipo geografinėje zonoje.

**2 sprendimas.** Mokslinėje publikacijoje „Detection of Tree Crowns in Very High Spatial Resolution Images“ (3) pateikiamas sprendimas yra grįstas primityvia kompiuterinė rega, nenaudojant mokymosi metodų. Nepaisant to, šio algoritmo rezultatai yra taip pat pakankamai geri, tačiau rezultatai pateikiami teritorijose, kuriose nėra didelis medžių tankis bei medžiai yra pasodinti vienodai nutolusiui atstumu vienas nuo kito pasikartojančiose struktūrose, pavyzdžiu plantacijose. Autoriai nepateikia šio algoritmo programinio kodo.

**3 sprendimas.** Įmonė Katam siūlo komerciniam naudojimui skirtą programą (9), kuri randa individualių medžių duomenis, apskaičiuoja įvairią informaciją, yra patogi naudoti bei patogiai pateikia išvestį. Šio algoritmo veikimui reikia pateikti tiriamo miško specialiai atliktas nuotraukas iš drono, kurios turi būti aukštos kokybės ir tinkamai nufotografuotos, todėl ši paslauga neišsprendžia dalies praeituose skyriuose aptartų problemų (pavyzdžiu vis dar reikalingas fizinis žmogaus buvimas informacijai surinkti).

**1 lentelė.** Konkurentų apžvalga

Palyginimo kriterijai	Individual Tree-Crown Detection in RGB Imagery Using Semi-Supervised Deep Learning Neural Networks	Detection of Tree Crowns in Very High Spatial Resolution Images	Katam TreeMap
Sprendimo tipas	Mokslinis straipsnis	Mokslinis straipsnis	Komercinė paslauga
Ar algoritmas apmokomas ranka žymėtais duomenimis	Dalinai	Ne	Nėra informacijos
Tikslumo <i>recall</i> įvertis	0.69	0.94	Nėra informacijos
Tikslumo <i>precision</i> įvertis	0.61	0.90	Nėra informacijos
Ar naudojimui tinkamos standartinės palydovų nuotraukos	Taip	Taip	Ne

Pateikiama atlikto darbo analizė indikuoja, kad nemažos dalies praeituose skyriuose minėtų problemų išspręsti neįmanoma naudojantis dabartiniais rinkoje pateikiamais sprendimais arba tokio tipo sprendimas nepateikiamas viešam naudojimui.

### 1.1.5. Prototipai ir pagalbinė informacija

Individualių medžių viršūnių radimo problemą galima apibrėžti kaip objektų aptikimo problemą nuotraukoje, tačiau aptiktam siekama grąžinti ne objekto gaubiantį stačiakampį, o objekto apskritimo koordinates (centrą) bei spindulį. Dėl šios priežasties šiame projekte siūlomas kompiuterinės regos algoritmas kuriamas medžių viršūnių aptikimo užduočiai implementuojant giliuoju mokymu paremtą objektų aptikimo algoritmą. Įvertinus įvairių viešai prieinamų objektų aptikimo algoritmų tikslumus (10), algoritmų sudėtingumus, viešas implementacijas, išvescių tipus, pritaikymo galimybes bei tobulinimo galimybes pasirinktas CenterNet (11) objektų aptikimo algoritmas šios užduoties implementavimui. Šis objektų aptikimo algoritmas aptiktų objektų neapdorotas išvestis pateikia kaip rastų objektų centrus, kiekvienam išvesties taškui pateikiant to taško ilgio ir pločio prognozę. Tai padaro šį algoritmą lengviau pritaikomu šiai užduočiai, kadangi bandoma pateikti išvestį kaip apskritimo centrą ir jo spindulį. Šis algoritmas nenaudoja „inkarų“ (12) strategijos gaubiantiems aptiktų objektų stačiakampiams prognozuoti, todėl jo pritaikymas ir apmokymas apskritimų aptikimui yra paprastesnis bei leidžia sumažinti aptikto objekto dydžiui skirtų išreikštį parametru skaičiu iš dvejų (ilgio ir pločio) iki vieno (spindulio). Taip pat šis algoritmas yra sėkmingai pritaikytas kitose aptikimo užduotyse bei pateikia aukščiausius rezultatus tarp konkurentų, pavyzdžiui realaus laiko veidų aptikime (13). Darbas su šiuo algoritmu pradedamas nuo viešai pateikiamos objektų aptikimo algoritmo Python programavimo kalbos implementacijos Keras karkasui (14). Eksperimentavimui su objektų aptikimo algoritmu naudojamos MobileNetV2 (15), MobileNetV3-Large (16), EfficientNet-B0 (17) ir ResNet-50 (18) neuroninių tinklų architektūros siekiant rasti kuo tinkamesnę neuroninio tinklo struktūrą, gebančią apsimokyti itin nedideliam duomenų kiekiui bei skaičiavimus atlikti patenkinamu greičiu. MobileNetV2, MobileNetV3-Large, EfficientNet-B0 architektūrų įgyvendinimui naudojamos viešai pateikiamos jų implementacijos Keras karkasui (19, 20, 21) bei ResNet50 naudojama Keras karkase integruota implementacija.

Eksperimentavimui pasirenkamos architektūros su įvairiomis savybėmis, renkantis architektūras atsižvelgiama į šiuos kriterijus:

- **Parametru skaičius.** Per mažas parametru skaičius gali neleisti tinklui pakankamai gerai apsimokyti, susitvarkyti su sudėtingesniais aptikimo atvejais, tačiau per didelis parametru skaičius gali neuroninį tinklą priversti persimokyti (overfitting) apmokymo duomenis.
- **Sluoksnių skaičius.** Didesis sluoksnių skaičius kartais gali apsunkinti modelio apmokymą dėl nykstančių gradientų problemos (22).
- **Ar sukurta naudojantis sustiprintuoju mokymu (*deep reinforcement learning*) naudojantis Neural Architecture Search technika (23).** Sustiprintuoju mokymu sukurti tinklai yra labiau optimizuoti nei standartiniai, todėl juos sunkiau patobulinti nesudėtingais sprendimais, lengva išbalansuoti jų struktūrą ir gauti prastesnį rezultatą.
- **Ar naudojami „bottleneck“ tipo blokai?** Naudojant bottleneck tipo blokus neuroniniams tinklui konstruoti pasiekiamas didesis greitis apdorojant informaciją, tačiau neprarandamas tinklo tikslumas.
- **Konvolucių tipas „bottleneck“ blokuose.** Depthwise tipo konvoluciujos yra naudojamos parametru skaičiui ir apdorojimo trukmei sumažinti, lyginant su standartinėmis konvolucijomis bei efektyviau išnaudotai modelio parametrus (24). Šiame darbe siekiama išbandyti abu variantus.
- **Ar naudojami *Squeeze & Excitation* blokai (25)?** Šių blokų panaudojimas gali padidinti modelio tikslumą su minimaliu greičio nuostoliu.

Žemiau pateikiame lentelėje palyginami šie konvoluciiniai neuroniniai tinklai:

**2 lentelė.** Konvoluciinių neuroninių tinklų palyginimas

Palyginimo kriterijai	MobileNetV2 (kai $\alpha = 1$ )	MobileNetV3 Large	EfficientNet-B0	ResNet-50
Parametru skaičius (mln.)	1.82	3.88	3.63	23.58
Sluoksnių skaičius	150	175	218	175
Ar sukurta naudojantis sustiprintuoju mokymu?	Ne	Taip	Taip	Ne
Ar naudojami „bottleneck“ tipo blokai?	Taip	Taip	Taip	Taip
Kokio tipo konvoluciujos naudojamos „bottleneck“ blokuose? (Depthwise / Standartinės)	Depthwise	Depthwise	Depthwise	Standartinės
Ar naudojami <i>Squeeze &amp; Excitation</i> blokai?	Ne	Taip	Taip	Ne

2 lentelėje vaizduojamas neuroninių tinklų palyginimas parodo plačią tinklų savybių įvairovę. Tai leis objektyviai įvertinti įvairių neuroninių tinklų efektyvumą medžių viršūnių aptikimo užduočiai su pasirinktu objektų aptikimo algoritmu bei aprašyti naują architektūrą iš gautų eksperimentavimo išvadų.

### 1.1.6. Ištekliai, reikalingi sistemai sukurti

Šiam algoritmui apmokyti naudojama Nvidia RTX 2060 vaizdo plokštė. Apmokymui naudojama virš 200 šiam projektui sužymėtų apie 3000 pikselių ilgio ir pločio aukštos kokybės palydovo nuotraukų.

Testavimui naudojamos 52 apie 1024 pikselių ilgio ir pločio palydovų atliktos nuotraukos. Siekiant sukurti aukštos kokybės algoritmą, tinkamą komerciniam naudojimui, eksperimentams ir galutiniams apmokymui reikėtų naudoti 3-5 aukščiausio galingumo vaizdo plokštes, tokias kaip Nvidia GTX 2080. Taip pat siekiant padidinti algoritmo tikslumą iki konkurencingo rinkoje algoritmui reikėtų plataus ir įvairaus duomenų rinkinio iš įvairių geografinių teritorijų nuotraukų, įvairių skirtingų palydovų su skirtingomis nuotraukų kokybėmis, filtrais, fotografavimo pozicijomis, metų laikais ir paros laikais. Duomenų žymėjimui prireiktų 300-600 valandų žmonių darbo.

## **1.2. Galimybių analizė**

### **1.2.1. Techninės galimybės**

Esminiai šio algoritmo kokybę ribojantys veiksnių yra skaičiavimo resursų trūkumas, mažas sužymėtų duomenų kiekis, ne itin aukšta duomenų žymėjimo kokybė bei duomenų vienodus. Kuriant tokį algoritmą reikalingas pakankamas vaizdo plokščių kiekis arba biudžetas skaičiavimams debesyse, norint atlikti įvairius eksperimentus apmokant didelį kiekį parametrų turintį neuroninį tinklą su gausiu ir įvairiu duomenų rinkiniu, architektūrinėmis modifikacijomis ir kitais apmokymo metodais bei galutinai apmokyti gerai veikiantį neuroninio tinklo variantą. Nepakankamas investicijų kiekis į šiuos resursus neleistų sukurti kokybiškai veikiančio algoritmo, jis būtų lengvai nugalimas rinkoje. Rinkoje taip pat nėra nemokamų ir lengvai prieinamų duomenų rinkinių, kuriuose yra sužymeti individualūs medžiai palydovų nuotraukose. Taip pat yra mažai nemokamai prieinamų aukštos kokybės nuotraukų bei kitų formatų duomenų, pvz. „lidar“ radaro žemės paviršiaus nuskanavimų. Šiuo metu aukščiausios kokybės palydovų nuotraukos yra apie 0.3 metro per pikselį raiškos, tai nėra pakankama norint lengvai atskirti medžio kontūrus iš nuotraukos. Bepiločiai orlaiviai gali pasiekti didesnę raišką, tačiau tokio tipo duomenys yra sunkiai prieinami ir brangūs. Kokybiškų duomenų gavimas ar kūrimas reikalauja papildomų laiko bei finansinių investicijų.

### **1.2.2. Vartotojų pasiruošimo analizė**

Kuriant šį projektą siekiama, kad vartotojui būtų reikalingas minimalus supratimas apie iš palydovų darytas nuotraukas, programinę įrangą bei jos naudojimą ir informacines technologijas. Nepaisant sudėtingos kompiuterinės regos algoritmo išvesties bei jo paties sudėtingumo siekiama, kad jo rezultatus savo pateiktai įvesčiai galėtų nesudėtingai gauti apie šių algoritmų kūrimą bei naudojimą žinių neturintis potencialus galutinis tokios aplikacijos vartotojas. Tai įgalinti siekiama kuriant vartotojo sąsajos aplikaciją, kurią siekiama padaryti kuo paprastesnę, tačiau kuo mažiau ribojančią pateikiamas informacijos kiekį. Abstraktus potencialus vartotojas yra žmogus, kurio konkrečiai veiklos sričiai nėra reikalingos gilios IT žinios bei visiškai nėra reikalingos kompiuterinės regos, dirbtinio intelekto, duomenų mokslo ar mašininio mokymo žinios, tačiau tai vartotojas, kuriam reikalinga apytikslė informacija apie medžius konkrečioje teritorijoje. Tai gali būti urbanistas, miškų prižiūrėtojas, miškų pirkėjas ar pardavėjas ir kt. Naudoti sukurtai vartotojo sąsajos aplikacijai vartotojui reikėtų turėti esmines ir pagrindines kompiuterinio raštingumo žinias, pavyzdžiui, sugebėti atskirti failo tipus, turėti minimalų supratimą apie palydovų nuotraukas, sugebėti, jei yra galimybė, įvertinti nuotraukos rezoliuciją.

## **2. Projektas**

### **2.1. Reikalavimų specifikacija**

#### **2.1.1. Komercinė specifikacija**

Priežastys šio projekto kūrimui yra potencialus rinkos poreikis efektyvesniams miškų stebėjimui ir priežiūrai bei siekis pritaikyti ir išbandyti moderniausias dirbtinio intelekto technologijas tikintis gautu rezultatu aplenkti praeityje sukurtus panašaus tipo įrankius. Projektas neturi konkrečių užsakovų.

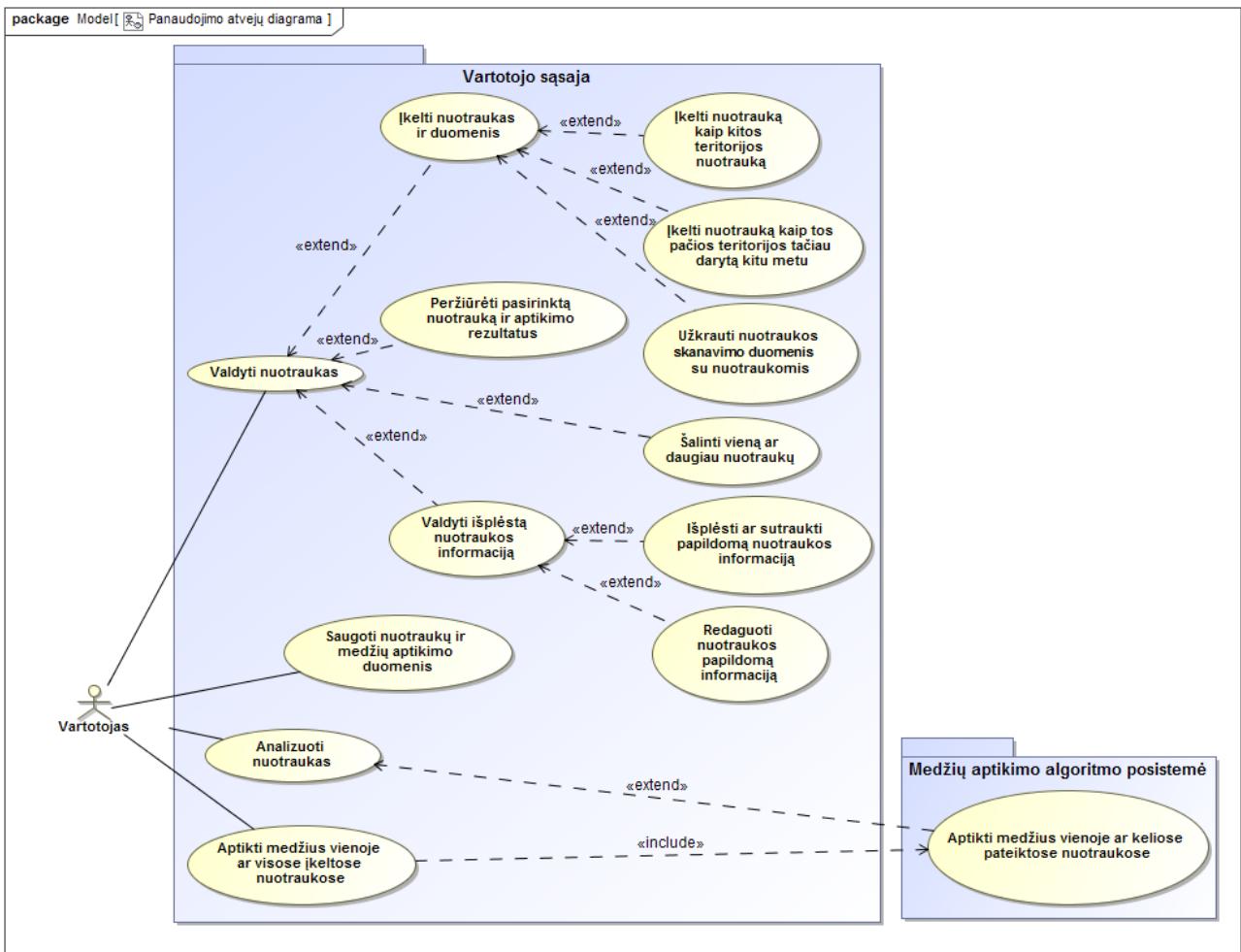
Šios kuriamos programos potencialūs vartotojai gali būti visi žmonės, kuriems yra svarbu atlikti skaičiavimus, susijusius su medžiais tam tikroje teritorijoje, pavyzdžiu apskaičiuoti medžių kiekį, miško tankį, medžių kiekį mieste, kiek medžių mieste buvo nukirsta per metus ir t.t. Tai gali būti urbanistai, siekiantys stebeti medžių kiekių kaitą miestuose, miškų prižiūrėtojai, kurie gali ieškoti nelegalių kirtimų, ar paprasti žmonės, siekiantys greitai įvertinti tikslinę perkamo ar parduodamo miško kainą.

Minimalios apimties produktui, kurį būtų galima išleisti vartojimui į rinką, sukurti reikėtų mažiausiai 6 IT specialistų (2 kompiuterinės regos ir dirbtinio intelekto specialistų kurti tiksliam ir greitam medžių radimo algoritmui sukurti, 2 programuotojų taikomajai programai plėsti ir tobulinti, 1 programuotojo web sistemos kurti siekiant algoritmą iškelti į „debesų“ aplinką ir optimizuoti jo veikimą, sukurti duomenų bazę bei jos valdymą patogiam ir efektyviam duomenų gavimui, bei 1 programuotojo interneto svetainei kurti, kuri reprezentuočia produktą bei leistų jį įsigyti, suteiktų informaciją bei pagalbą vartojimui.

Į platesnę projekto ateities viziją galima įtraukti algoritmą, kuris gebėtų ne tik daugumoje geografinių teritorijų aptikti medžius, bet sugebėti atpažinti abstrakčią medžių rūšį (lapuotis ar spylgiuotis) ir aukštį. Tai potencialiai galima daryti stebint tos pačios teritorijos palydovų nuotraukas ir ieškant didelių medžių pokyčiu laike, siekiant įvertinti, ar medis žiemą prarado žalią spalvą ar ne, taip įvertinant, ar medis numetė lapus ir žymimuose duomenyse priskirti medžiui rūšį. Taip pat duomenis galima žymėti teritorijose, kurioms yra atlirkas „lidar“ skanavimas ir sužymėtomis nuotraukomis iš jų priskirti medžių aukštį. Taip galima apmokyti platesnį veikimo spektrą turintį modelį. Tokio tipo apmokymas, kai modelis apmokomas grąžinti daugiau skirtingo tipo išvesčių aptinkamam objektui, taip pat turi tendenciją pagerinti ir pagrindinę aptikimo užduotį. Siekiant aplikaciją padaryti efektyvesne ir pagerinti vartotojo patirtį, ją reikėtų pritaikyti itin didelio kiekių duomenims apdoroti ir stebeti. Tam būtų naudinga turėti vartotojams lengvai prieinamą duomenų bazę su kuo didesniu planetos nuotraukų kiekiu bei šią duomenų bazę reguliarai pildyti naujais duomenimis. Leidimas vartotojams greitai prieiti ir apdoroti duomenis, pasirinktus pagal koordinates, vizualiai ar pagal kitus parametrus leistų daug greičiau atlirkti įvairias stebėjimo užduotis, nei su nuotraukomis, kurias įkelia pats vartotojas. Visi pasiūlymai projekto vizijai reikalautų dar daugiau laiko, žmogiškųjų išteklių ir finansinių investicijų.

#### **2.1.2. Sistemos funkcijos**

Sistemos funkcioniniai reikalavimai vaizduojami 2.1. pav. Esančioje UML panaudojimo atvejų diagramoje:



**2.1 pav.** Sistemos panaudojimo atvejų (use case) UML diagrama vartotojo sasajos programai.

Pagrindiniai sistemos funkciniai reikalavimai:

1. Patogus nuotraukų valdymas- jų įkėlimas, tvarkymas, nuotraukų peržiūréjimas, informacijos įrašymas turi būti patogiai integruotas į vartotojo sasają. Šis funkcinis reikalavimas skirstomas į kitus papildomus reikalavimus:
  - 1.1. Nuotraukų ir duomenų įkėlimas. Nuotraukas turi būti galima įkelti iš vartotojo sasajos naudojantis patogiu failų atidarymo dialogu. Nuotraukas galima įkelti kaip:
    - 1.1.1. Naujos teritorijos nuotraukų įkėlimas, atidarant naują nuotraukų skyrių programe;
    - 1.1.2. Esamo nuotraukų skyriaus išplėtimas, įkeliant tos pačios teritorijos nuotraukas, tačiau atlirkas kitu laiku. Tai daroma tam, kad būtų galima analizuoti nuotraukas tam tikrame laiko spektre, pvz. ar kažkuris medis nebuvo nukirstas.
    - 1.1.3. Nuotraukų užkrovimas iš prieš tai atlirkos nuotraukų apdorojimo sesijos. Tokiu būdu užkraunami ir visi prieš tai gauti skanavimo rezultatai.
  - 1.2. Nuotraukų peržiūréjimas. Bet kuria įkeltą nuotrauką galima peržiūrėti bei vizualizuoti medžių aptikimo rezultatus, jei nuotrauka buvo nusiusta apdorojimui algoritmu;
  - 1.3. Nuotraukų šalinimas iš aplikacijos, pašalinant ir aptikimo rezultatus, jei tokie buvo atlirki;
  - 1.4. Valdyti nuotraukos papildomą informaciją:
    - 1.4.1. Išplėsti ar sutraukti nuotraukos ar visų nuotraukų papildomą informaciją. Taip galima stebeti tokią informaciją kaip nuotraukos pavadinimas, rezoliucija, dydis ir kt.;
    - 1.4.2. Redaguoti dalį nuotraukos papildomos informacijos.

2. Duomenų saugojimas. Aplikacija leidžia atlikti šias duomenų saugojimo funkcijas:
  - 2.1. Saugoti nuotrauką apdorojimo sesiją, kurios metu bus išsaugota nuotraukų informacija su medžių aptikimo rezultatais, jei aptikimas buvo atliktas;
  - 2.2. Saugoti apdorotą nuotrauką bendros analizės rezultatus.
3. Medžių aptikimas nuotraukose. Vartotojui suteikiama galimybė pasirinktą ar visas nuotraukas per API užklausą nusiųsti modeliu. Modelis, esantis kitoje posistemėje sulaukęs užklausos apdoroja nuotraukas ir grąžina rezultatus JSON formatu. Aplikacija, sulaukus rezultatų juos dekoduoją ir jei kitą kartą pasirinkus nuotrauką vaizdavimui papildomai vaizduos ir aptikimo rezultatus;
4. Analizuoti aptikimo rezultatus. Vartotojui pasirinkus galima gauti analizės rezultatus iš apdorotų nuotraukų, vartotojui yra pateikiama statistinė informacija apie nuotraukoje aptiktus medžius. Papildomos funkcijos atsidarius analizės langą:
  - 4.1. Saugoti analizės duomenis;
  - 4.2. Atlikti medžių aptikimą visoms aplikacijoje įkeltoms nuotraukoms.

#### **2.1.3. Vartotojo sasajos specifikacija**

Reikalavimai vartotojo sasajai:

1. Vartotojo sasaja turi išpildyti visus jai iškeltus funkinius reikalavimus;
2. Vartotojo sasaja turi leisti patogiai peržiūrėti vartotojo įkeltas nuotraukas bei aiškiai vizualizuoti algoritmo pateiktus aptikimo rezultatus, aiškiai demonstruojant algoritmo veikimą ir jo tikslumą;
3. Įkeltas paveikslėlis gali būti dedami į skyrių, kuriame jis bus lyginami laiko atžvilgiu su kitomis nuotraukomis (keliose nuotraukose nufotograuota ta pati teritorija, tačiau nuotraukos yra atlertos skirtingu laiku), arba sukurs naują skyrių, jei vartotojas įkelia nuotrauką su nauja teritorija.
4. Vartotojas turi galimybę peržiūrėti bendrą grupių, konkrečios nuotraukos ar viso skanuojamo ploto analitinę informaciją.
5. Programoje iškilus problemai, vartotojui turi būti pateikiamas pranešimas naujame lange su informacija, kokia klaida įvyko, bei kaip tos klaidos daugiau nekartoti, jei klaidą sukėlė pats vartotojas, pavyzdžiui pateikdamas netinkamą įvestį.
6. Programoje esantis algoritmo rezultatų analizės langas turi paprastą statistinę informaciją apie aptiktas medžių viršūnes.
7. Vartotojas turi turėti galimybę pasirinkti, ar aplikacijoje nori matyti algoritmo apdorojimo rezultatus.
8. Vaizduojami algoritmo apdorojimo rezultatai turi grafiškai suteikti vartotojui informaciją apie kiekvieno modelio aptikto medžio užtikrintumą, kad toje vietoje iš tikrujų yra medis.
9. Algoritmo pateikti medžių aptikimo rezultatai taip pat turi būti analizuojami laike su kitomis tame pačiame skyriuje įkeltomis nuotraukomis, jei jų yra. Rasti medžiai, kurie nebuvu rasti toje pačioje vietoje kitose nuotraukose turi būti pažymėti tame pačiame lange, kaip ir kiti aptikti medžiai, tačiau kitokiu žymėjimo būdu ar kita spalva.

#### **2.1.4. Realizacijai keliami reikalavimai**

Šioje dalyje aprašomi aplikacijos nefunkciniai reikalavimai.

Kompiuterinės regos algoritmui ir jo implementacijai keliami nefunkciniai reikalavimai:

1. Kuriamas dirbtinio intelekto algoritmas turi neužtruktū ilgiau nei 1 sekundę apdoroti vienam 1024x1024 pikselių dydžio paveikslėliui naudojantis Nvidia RTX 2060 vaizdo plokštę.

2. Kuriamas modelis įvesties nuotraukai kaip rezultatą turi pateikti n spėjimų, kur kokioje konkrečioje planetos nuotraukos vietoje yra medžiai, kiekvienam medžiui pateikiant x ir y centro koordinates nuotraukos pikselių koordinačių sistemoje ir medžio spindulį, spindulio ilgi išreiškiant nuotraukos pikselių kiekiu, bei kiekvieno aptikimo spėjimo tikimybę (užtikrintumą).
3. Testavimo duomenyse algoritmas turi nepateikti ne daugiau kaip 60% neigiamų spėjimų, atliekant ne mažiau kaip 60% teisingų spėjimų visam testavimo duomenų rinkiniui. Spėjimas laikomas teisingu, kai modelio spėjamas apskritimas turi ne mažesnį kaip 50% ploto persidengimą (0.5 IoU vertė) su testavimo duomenyse pažymėtu apskritimu. Jei algoritmas tenkina šį testavimo reikalavimą konkrečiai testavimo duomenų rinkinio daliai, turinčiai tas pačias abstrakčias savybes (pavyzdžiu miškingoms teritorijoms, tačiau netenkina kitoms, pavyzdžiu dažnai klysta miesto teritorijoje, klaudingai atpažstant pastatus kaip medžius)- reikalavimas laikomas įgyvendintu, kadangi modelis nebūtinai gali būti pritaikomas visiems galimiems atvejams bei testavimo rezultatai gali būti iškreipti priklausomai nuo testavimo duomenų rinkinio subjektyvumo ir duomenų savybių pasiskirstymo. Šiuo tyrimu taip pat siekiama atpažinti, kuriems atvejams galima pritaikyti šį algoritmą.
4. Kuriamas modelis turi gebeti aptikti iki 1000 medžių vienoje 1024x1024 pikselių dydžio nuotraukoje.
5. Kuriamo giliojo neuroninio tinklo apmokomų ir neapmokomų parametru failas turi neužimti daugiau nei 1GB atminties.
6. Kompiuterinės regos algoritmas turi aptikti medžius, kurie nuotraukoje yra ne didesni nei 512 pikselių skersmens ir ne mažesnius nei 20 pikselių skersmens. Algoritmas gali aptikti medžius už šią dydžio ribą, tačiau testavime j tai neturėtų būti kreipiamas dėmesys.
7. Algoritmo aplikacija, priimanti užklausas iš vartotojo sėsajos aplikacijos, turi sugebeti apdoroti užklausą su iki 100 skirtingų 1024x1024 pikselių dydžio nuotraukomis.
8. Algoritmas turi sugebeti apdoroti bet kokio dydžio nuotrauką, didesnę nei 128x128 pikselių dydžio. Itin didelės nuotraukos turi atitinkamai būti karpomos į mažesnes ir paduodamos algoritmui atliglioti aptikimo užduotis atskirai, sujungiant aptiktus rezultatus į bendrą koordinačių sistemą vėlesniuose apdorojimo etapuose, taip išlaikant spėjimų stabilumą, tikslumą ir neviršijant atminties kieko skaičiavimo įrenginyje.

Grafinės vartotojo sėsajos programai keliami nefunkciniai reikalavimai:

1. Giliųjų neuroninių tinklų modelis turi būti patogiai ir lengvai naudojamas iš taikomosios programos. Programa turi komunuoti su modeliu API užklausomis, kaip užklausos duomenis pateikiant apdorojimui skirtas nuotraukas ir kaip atsakymą gaunant aptikimo rezultatus.
2. Programa turi leisti palaikyti iki 100 užkrautų 1024x1024 pikselių dydžio nuotraukų.
3. Programa turi leisti vizualizuoti iki 4000x4000 pikselių dydžio nuotraukas.

Apribojimai:

1. Grafinės sėsajos programai paleisti reikalinga Microsoft Windows operacinė sistema.
2. Algoritmo posistemei paleisti komunikavimo su grafinės sėsajos aplikacija režimu reikalinga Python programavimo kalba, TensorFlow 1.14 versijos Keras karkasas, Flask biblioteka API komunikacijai bei OpenCV bei NumPy bibliotekos nuotraukų ir matricų apdorojimui.

### **2.1.5. Techninė specifikacija**

Siekiant atlikti nuotraukų apdorojimą naudojantis grafinės sasajos programa reikalinga 2.1.4 skyriuje apibojimų skiltyje aprašyta programinė įranga. Norint naudotis sistema nefunkciniuose reikalavimuose apribotu greičiu lokalioje aplinkoje reikalingas kompiuteris su mažiausiai 8GB vidinės RAM atminties bei vaizdo plokšte, turinčia mažiausiai 4GB vidinės atminties.

Norint pakartotinai apmokyti modelį papildomai reikalinga TensorFlow karkaso 1.14 versija, albumentations bei matplotlib Python bibliotekos, Nvidia CUDA 10.0 karkasas su CuDNN biblioteka, kompiuteris su mažiausiai 8GB RAM atminties, vaizdo plokšte, turinčia mažiausiai 4GB vidinės atminties, bei tam tinkamas duomenų rinkinys.

## **2.2. Projektavimo metodai**

### **2.2.1. Projektavimo valdymas ir eiga**

Ši sistema buvo kuriama naudojantis iteraciniu programinės įrangos kūrimo modeliu. Darbas buvo skirstomas į tris iteracijas. Pirmosios iteracijos metu buvo siekiama sukurti patenkinamai veikiantį kompiuterinės regos algoritmą, kuris gebėtų būti tinkamai apmokomas pateiktais duomenimis, apmokytas pateiktų tinkamas išvestis bei tenkintų kuo daugiau, bet ne visus šios realizacijos dalies reikalavimus. Šios iteracijos metu taip pat buvo sužymėtas apmokymo duomenų rinkinys pirmosios testavimo ir validacijos duomenų rinkinių versijos. Antrosios iteracijos metu buvo kuriama vartotojo sasajos programa siekiant patenkinti kuo daugiau šiai sistemos daliai išskeltų reikalavimų, tuo pačiu pasyviai tobulinant kompiuterinės regos algoritmą, atliekant sudėtingesnius bei ilgesnį apmokymo laiko reikalaujančius eksperimentus. Paskutinės iteracijos metu buvo kuriama programos dalis skirta vartotojo sasajos programos ir kompiuterinės regos algoritmo komunikacijai per API, naujas konvoliucinis neuroninis tinklas medžių viršūnių aptikimo užduočiai pagal praetoje iteracijoje gautus eksperimentų rezultatus, žymimi galutiniai testavimo bei validacijos duomenų rinkiniai, atliekamas klaidų taisymas.

### **2.2.2. Projektavimo technologija**

Šiai sistemai projektuoti ir aprašyti buvo naudojami *UML 2.5* standarto grafiniai elementai aprašyti klasių, paketų, sekų ir veiksmų diagramoms. Šie grafiniai elementai buvo kuriami naudojantis MagicDraw 19 modeliavimo įrankiu. Konvoliucinio neuroninio tinklo diogramos buvo kuriamos naudojantis PlotNeuralNet (26) įrankiu, kuris naudojasi LaTeX dokumentų ruošimo sistema neuroninio tinklo grafiniams elementams vaizduoti.

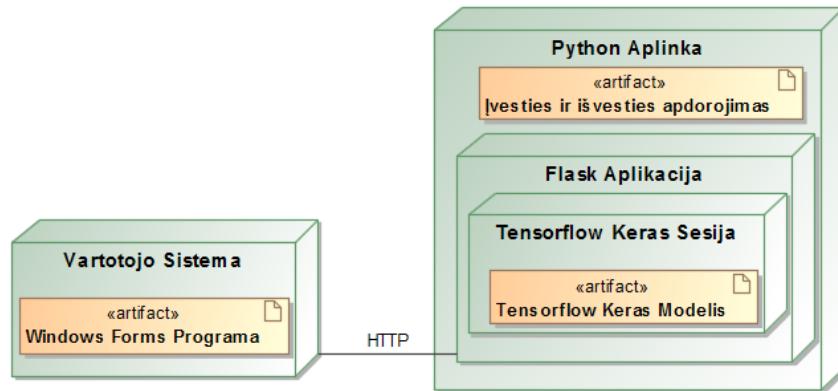
### **2.2.3. Programavimo kalbos, derinimo, automatizavimo priemonės, operacinė sistemas**

Šiame projekte kurtas kompiuterinės regos algoritmas yra kurtas naudojantis Python 3.7 programavimo kalba PyCharm programavimo aplinkoje. Šiam algoritmui kurti buvo naudotos TensorFlow, TensorFlow Keras, Flask, Albumentations, OpenCV bibliotekos Python programavimo kalbai. Vartotojo sasajos aplikacija buvo kuriama naudojantis Windows Forms įrankiu, ją kuriant Visual Studio aplinkoje. Visa projekte aprašoma sistema buvo kuriama Windows 10 operacinėje sistemoje.

## 2.3. Sistemos projektas

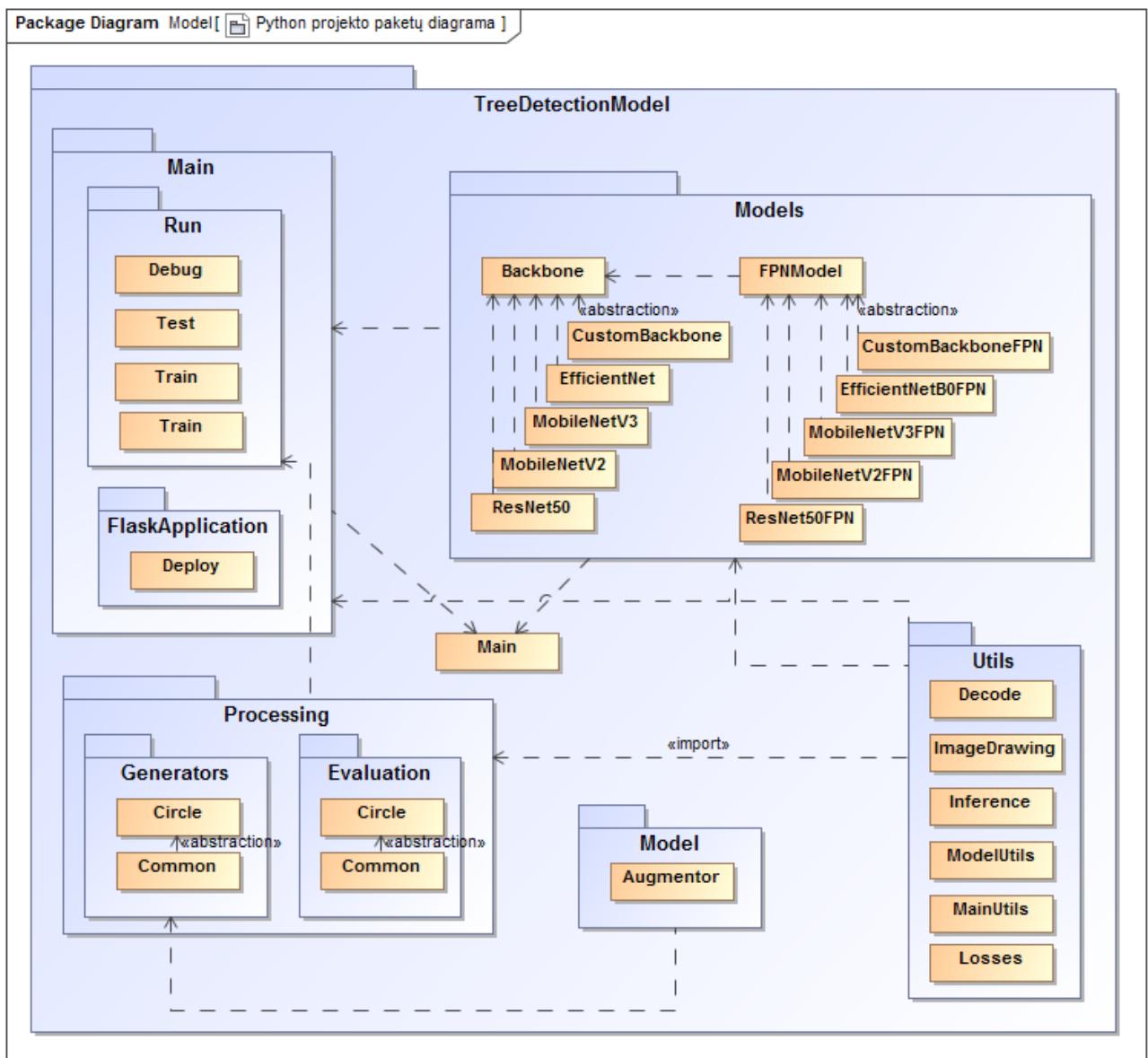
### 2.3.1. Statinis sistemos vaizdas

Šiame skyriuje naudojantis UML diagramomis yra aprašoma sukurtos sistemos struktūra. Naudojantis išdėstymo diagrama aprašoma sistemos struktūra ir vartotojo sėsajos programos komunikacija su kompiuterinės regos algoritmo programa veikimo metu. Naudojantis paketų ir klasių diagramomis yra aprašomos vartotojo sėsajos programos bei kompiuterinės regos algoritmo programos struktūros.



2.2 pav. Sistemos komponentų UML diagrama.

Sistema sudaryta iš dvejų esminių komponentų - vartotojo sistemos, kurioje naudojama grafinės sėsajos aplikacija, kurta su Windows Forms, bei Python aplinka, kurioje veikianti *Flask aplikacija* laukia HTTP užklausos iš vartotojo sistemos. Flask užklausa, gavusi vartotojo užklausą apdoroti nuotraukas, inicializuja *Tensorflow Keras* modelį, vėliau nuotraukas apdoroja taip, kad jos būtų tinkamo formato modelio įvesčiai. Modelio pateikta išvestis yra apdorojama, konvertuojama į JSON formatą ir siunčiama atgal į aplikaciją, kur yra dekoduojama ir vaizduojama.

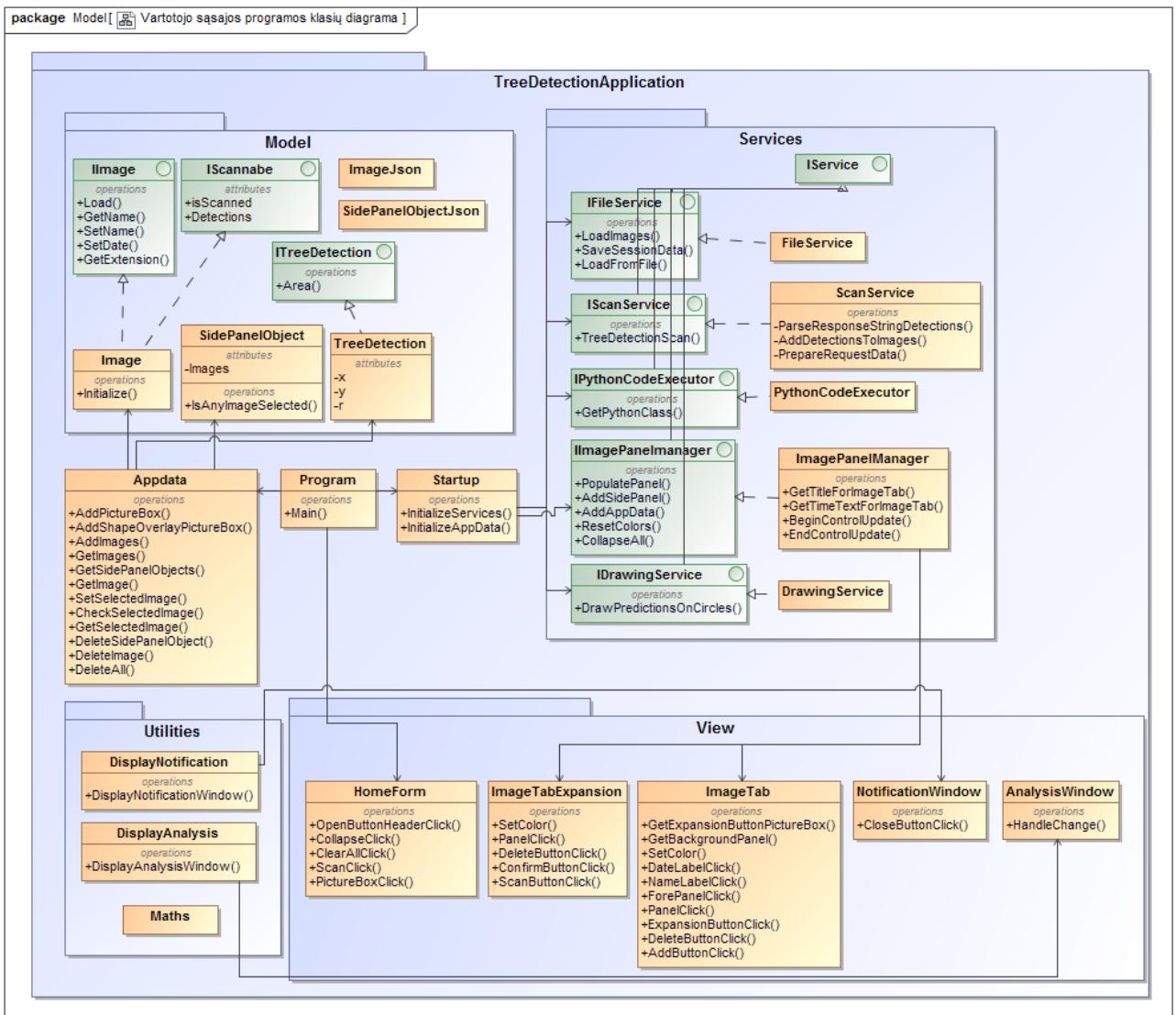


**2.3 pav.** Kompiuterinės regos algoritmo programos paketų UML diagrama.

Šioje pateiktoje paketų UML diagramoje vaizduojama kompiuterinės regos programos struktūra. Programa sudaryta iš šių komponentų ir failų:

- „Main“ paketas. Šiame pakete sudėti esminiai failai, kuriuose yra aprašytas abstraktus programos veikimas įvairiuose režimuose. Šis paketas skirstomas į du sub-paketus.
  - „Run“ paketas. Šiame pakete yra failai, kurie skirti modelio paleidimui įvairiais jo kūrimui skirtais režimais. Skirstomas į šiuos failus:
    - „Train“ python failas. Šiame faile yra visos esminės abstrakčios funkcijos ir procesas modelio apmokymui.
    - „Test“. Šiame faile yra procesai, skirti testuoti apmokyto algoritmo tikslumui, paleisti jį veikimo režime, peržiūrėti jo galutinius spėjimus tam tikrai įvesčiai ir tikrinti, ar tiksliai veikia tam tikri jo algoritmai, skirti įvesčiai apdoroti.
    - „Debug“ faile aprašomi procesai, kurie paleidžia modelį testavimo režimu, kurio tikslas yra tikrinti neapdorotas modelio išvestis. Tai leidžia susidaryti išvadas apie modelio apsimokymo stabilumą ar priežastis, kodėl modelis veikia prasčiau tam tikrais atvejais.

- „FlaskApplication“ paketas. Šiame pakete yra algoritmo veikimo implementacija, skirta komunikavimui su vartotojo sąsajos aplikacija.
- „Models“ paketas. Šiame pakete esančių failų atsakomybė yra inicializuoti Tensorflow Keras modelį, savyje turintį neuroninį tinklą, priimantį apdorotą įvestį ir grąžinantį dekoduotas išvestis (apskritimų koordinates paveikslėlio koordinačių ašyje, apskritimo spindulį bei spėjimo užtikrintumo tikimybę kiekvienam rastam medžiui). Ši paketą sudarantys esminiai failai:
  - Backbone failas aprašo stuburinio neuroninio tinklo inicializavimą. Šis neuroninis tinklas yra pagrindinė algoritmo dalis, kurioje bus vykdomi skaičiavimai ir išgaunamos svarbiausios išvesties savybės, naudojamos tolimesniuose apdorojimo etapuose.
  - FPNModel faile sukuriamas pilnas neuroninis tinklas prie Backbone tinklo prijungiant *Feature pyramid network* tinklo dalį, kuri stuburiniam tinklui sukuria galutinę ir tarpines savybių išvestis, jas sujungia ir galutinai apdoroja, grąžindama tikimybę, apskritimo spindulio spėjimą ir 4 pikselių paklaidą išvesties nuotraukos pikseliams. FPNModel klasę paveldintis failas grąžina apmokymui skirtą modelį, savyje turintį paklaidos (*loss*) funkcijos skaičiavimo bloką, testavimui ir naudojimui skirtą modelį, turintį dekodavimo bloką ir grąžinantį apdorotas išvestis, bei modelį, grąžinantį neapdorotą išvestį. Šie modeliai veliau naudojami Main pakete priklausomai nuo paleidimo režimo.
- „Processing“ pakete aprašytos funkcijos, dirbančios su apmokymo duomenimis.
  - „Generators“ pakete aprašomas funkcijos, sukuriančios generatorių Tensorflow Keras modelio apmokymui. Generatorius paskirtis yra apmokymo metu pateikti modelio apmokymo procesui vieną partiją apmokymui apdorotų duomenų, kai to paprašo Tensorflow Keras Backend procesas. Šiame pakete esantis „Common“ failas aprašo abstraktų duomenų apdorojimo procesą, o „Circle“ failas aprašo konkretaus duomenų rinkinio, naudojamo šiame projekte, nuotraukų bei anotacijų užkrovimą ir pirmąjį pirmąjį apdorojimą. Apmokymo metu yra sukuriama vienas generatorius objektas duomenų pateikimui apmokymo procesui, taip pat sukuriamas kitas generatorius objektas validacijos duomenims, kurie yra nepriklausomi nuo apmokymo duomenų, pateikimui.
  - „Evaluation“ pakete aprašomas funkcijos, kurios pakartotinai po fiksuoto apsimokymo žingsnių periodo tikrina modelio tikslumą modeliui apsimokant.
- „Augmentor“ pakete aprašytos funkcijos augmentuoja duomenis modelio apmokymo metu, yra naudojamos kiekvienam modelio apsimokymui pateiktam paveikslėliui.
- „Utils“ pakete yra aprašomi visi pašaliniai bei pakartotinai naudojami algoritmai ir statinės klasės. Šie algoritmai naudojami Main procesų inicializavimui, nuotraukų apdorojimui, kitiems matematiniams skaičiavimams, duomenų konvertavimui, modelio testavimui ir modelio neapdorotų išvescių dekodavimui.

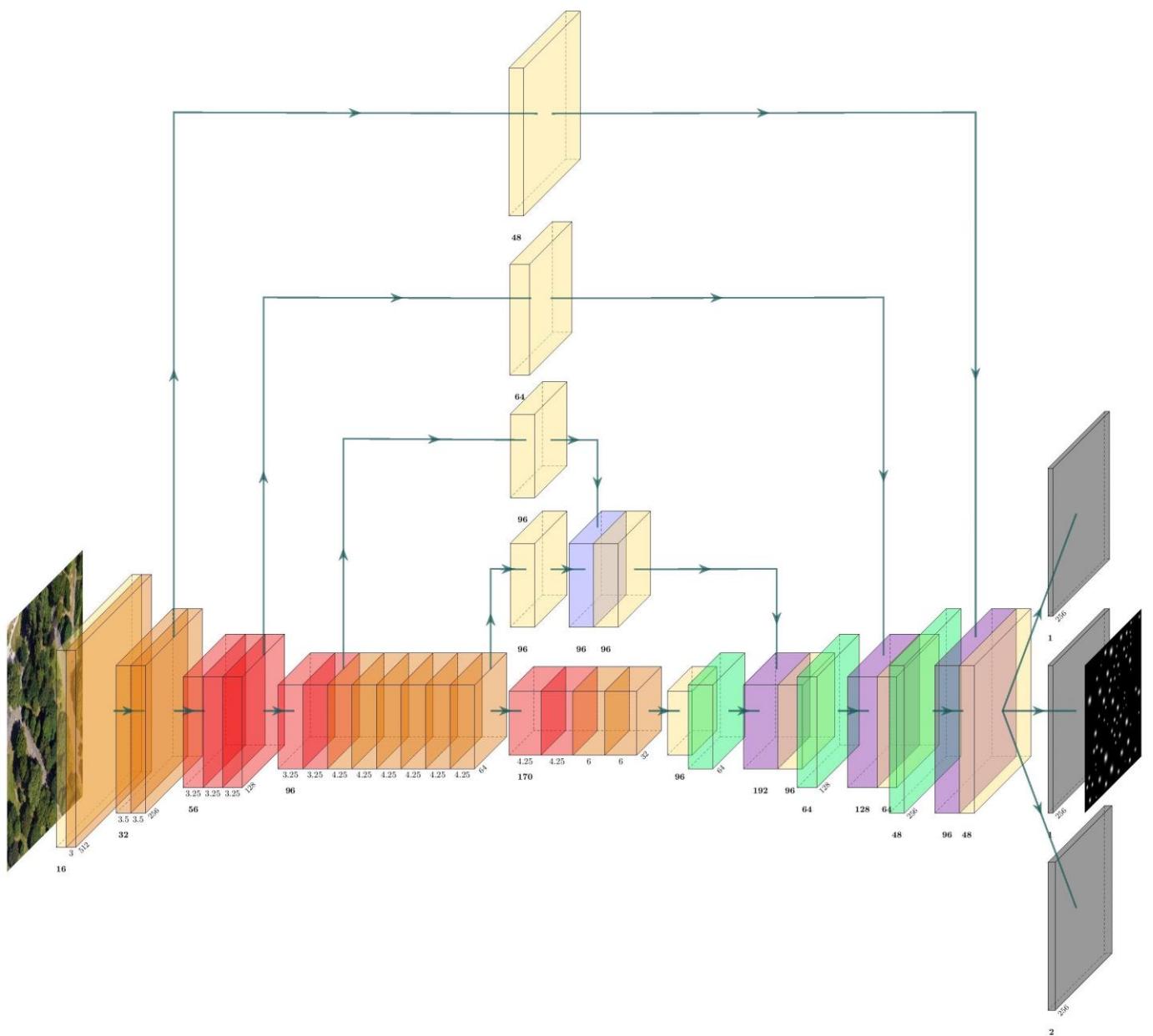


2.4 pav. Grafinės vartotojo sąsajos programos klasų UML diagrama.

Šioje pateiktoje UML diagramoje pateikiame svarbiausios vartotojo sąsajos programos klasės bei metodai. Programoje yra naudojama MVC (Model View Controller) architektūra, pagal tai suskirstytos klasės į nurodytus paketus. Controller architektūros dalis yra padalinta tarp View paketo klasų įvykių metodų (pavyzdžiu paspaudimas ant kurio nors konkretaus View objekto komponento) bei Services paketo klasės. Programą sudaro šie paketai ir komponentai:

- Model pakete talpinamos klasės, skirtos konkretaus duomenų tipo aprašymui, kurie yra inicializuojami daug kartų ir laikomi duomenų struktūrose.
- Services pakete aprašomos klasės, kurios apdoroja programoje įvykusius įvykius, pavyzdžiu paspaudimus ar užklausas, yra atsakingi valdymą procesų valdymą, kuriems svarbus gijų valdymas, ar valdo procesus, kuriems reikalinga savyje laikyti konkrečius duomenis.
- View pakete talpinami langai bei kiti vizualūs objektai, su kuriais sąveikauja vartotojas, su jų logikai bei valdymui aprašyti reikalingais metodais.
- Utilities pakete talpinamos statinės klasės.
- Appdata klasė veikia kaip laikina duomenų talpykla, kurioje talpinami Model paketo klasų objektais, kuriuos talpina, kuria, apdoroja įvairūs Service paketo klasės objektais. Programa klasėje vyksta visa paleidimo ir inicializavimo logika, tuo pačiu paleidžiant ir Startup klasės metodus, inicializuojančius Services klasės objektus bei pradinis duomenis.

Šiame projekte aprašomas kompiuterinės regos algoritmas rezultatams prognozuoti naudoja konkrečiai šiai užduočiai sukurtą FPN (Feature Pyramid Network) (27) struktūros tipo konvolucių neuroninį tinklą. Ši architektūra sukurta remiantis eksperimentiniais rezultatais išbandant kitų tipų architektūras, perimant bei implementuojant gerus rezultatus pateikiančius sprendimus bei išmèginant kitus architektūrinius sprendimus, kurie leidžia tinklą labiau optimizuoti medžių aptikimo užduočiai. Žemiau pateikiama šio neuroninio tinklo bendros struktūros diagrama:



**2.5 pav.** Kompiuterinės regos algoritme naudojamo konvoluciūnio neuroninio tinklo struktūra, kai įvesties matrica dydis yra  $1024 \times 1024 \times 3$  dydžio. Pavyzdinės įvesties nuotraukos šaltinis: Google Earth.

Aukšciau pateiktoje nuotraukoje aprašoma naudojamo konvoluciūnio neuroninio tinklo struktūra. Paveikslėlyje pateikta diagrama vaizduoją konvoluciūnio neuroninį tinklą jo dalinių išvesties blokais. Kiekviena spalva išreiškia konkretaus tipo operacijos išvesties bloką. Iš bloką rodyklėmis atvesti arba konkretaus bloko kairėje esantis blokas yra šio konkretaus bloko apdorojimo operacijos išvestis. Kiekvieno bloko dimensijų ilgi ir ploti išreiškia bloko pabaigoje pateiktas ištirižai pateiktas skaičius. Kiekvienos blokų serijos pradžioje parašytas skaičius išreiškia kiekvieno bloko kanalų

skaičių. Kiekvieno siauro praėjimo bloko (oranžinė bei raudona spalvos) apačioje parašytas skaičius išreiškia to bloko kanalų išplėtimo koeficientą nuo įvesties kanalų skaičiaus. Žemiau pateiktoje lentelėje aprašomi operacijų blokai, kurie yra spalvomis pažymėti 2.5 paveikslėlyje.

**3 lentelė.** 2.5 pav. pateiktos konvoliucinio algoritmo diagramos blokų spalvų bei anotacijų paaiškinimas.

- **Konvoliucijos blokas.** Šiame bloke įvesčiai pirmiausiai naudojama konvoliucijos operacija (Tensorflow Keras aplinkoje naudojamas Conv2D sluoksnis). Pačiame pirmame šio bloko panaudojime, skirtame įvesčiai apdoroti, konvoliucijos operacijai naudojamas  $3 \times 3$  dydžio filtras, visiems kitiems šio bloko panaudojimams naudojamas  $1 \times 1$  dydžio filtras, kurio pagrindinė paskirtis yra apdoroti informaciją tarp skirtingų kanalų ir sumažinti arba padidinti kanalų skaičių sekančiai įvesčiai po šio bloko. Po konvoliucijos operacijos naudojama *Batch Normalization* (28) operacija konvoliucijos išvestims normalizuoti ir treniravimo procesui pagreitinti. Po *Batch Normalization* operacijos išvestys yra pateikiamos „ReLU“ aktyvacijos funkcijai, kiekvieną įvesties vienetą apdorojant pagal  $\max(0, x)$  funkciją.
- **Siauro praėjimo (bottleneck) konvoliucijos blokas.** Šis blokas yra panašus į MobileNetV2 konvoliucinio neuroninio „bottleneck“ bloką. Šio bloko pradžioje naudojama konvoliucija su  $1 \times 1$  filtru dimensijoms išplėsti. Dimensijų kiekis padidinamas tokiu išplėtimo, kokiui šis blokas yra pažymėtas pagrindinėje neuroninio tinklo diagramoje (2.5. pav.) bloko apačioje, t.y. jei bloko įvestis yra 64 kanalų skaičiaus, o pateiktas koeficientas išplėtimo koeficientas yra 3.5, tai konvoliucijos operacijos išvesties kanalų skaičius bus 224. Po šios operacijos sekā *Batch Normalization* operacija bei „ReLU“ aktyvacijos operacija. Po kanalų išplėtimo naudojama „Depthwise“ konvoliucijos operacija su  $3 \times 3$  dydžio filtru, po kurios sekā *Batch Normalization* operacija. Jei siauro praėjimo blokas yra naudojamas dimensijoms mažinti, tai „Depthwise“ konvoliucija grąžins dvigubai mažesnio ilgio ir pločio išvesčių matricą (*stride* parametras lygus 2). Galiausiai išvesties kanalai yra pakartotinai sutraukiami tokiu pačiu koeficientu, kokiui buvo išplėsti, naudojantis konvoliucijos operacija su  $1 \times 1$  dydžio filtru, išvestis apdorojant su *Batch Normalization* operacija ir „ReLU“ aktyvacijos funkcija. Visose, išskyruis pačioje pirmoje modelio diagramoje pavaizduotoje siauro praėjimo konvoliucijos operacijoje, šios apdorotos išvestys yra sudedamos su šios operacijos įvestimis, sudedant visus tose pačiose matricų pozicijose esančius skaičius. Jei siauro praėjimo konvoliucijos blokas yra naudojamas dimensijoms mažinti (*stride* parametras lygus 2), tai prieš pridedant ivestis prie išvesčių išvestys yra papildomai apdorojamos su „Depthwise“ konvoliucija, kurios išvesties ilgis ir plotis bus dvigubai mažesnis, *BatchNormalization* operacija, paprastos konvoliucijos operacija su  $1 \times 1$  filtru, *BatchNormalization* operacija bei galiausiai „ReLU“ aktyvacijos funkcija.  
Šio tipo blokas leidžia modeliui pridėti efektyviau išnaudoti parametrus, taip pat iššvaistytį mažiau skaičiavimo laiko, lyginant „ResNet“ tipo *bottleneck* blokais, kuriuose naudojamos standartinės konvoliucijos.
- **Siauro praėjimo konvoliucijos blokas su  $5 \times 5$  „Depthwise“ konvoliucijos filtru.**
- **Matricų sudėties operacija.** Šios operacijos metu dvi identiškų dimensijų matricos sudedamos į vieną, kiekvienos iš jų elementą sudedant su kitos matricos toje pačioje vietoje esančiu elementu.
- **Dekonvoliucijos blokas.** Šis blokas pradedamas dekonvoliucijos operacija (29), kuriai naudojamas  $3 \times 3$  dydžio filtras ir kuri grąžina dvigubai didesnio ilgio ir pločio išvestį, sumažinant kanalų skaičių iki pagrindinėje neuroninio tinklo diagramoje pažymėto skaičiaus.

Po dekonvoliuicijos operacijos išvestys yra apdorojamos su *Batch Normalization* operacija ir „ReLU“ aktyvacijos funkcija.

- - **Sujungimo (Concatenate) operacija.** Šioje operacijoje dvi įvesčių matricos, kurių ilgis ir plotis yra vienodi, yra sujungiamos į vieną didesnę matricą, prijungiant tarpusavio kanalus.
- - **Išvesties konvoluciujos blokas.** Šiuo bloku žymima neuroninio tinklo išvestis. Jai apskaičiuoti naudojama konvoluciujos operacija su  $1 \times 1$  dydžio filtru. Karščio žemėlapui prognozuoti po išvesties naudojama „sigmoid“ aktyvacijos funkcija, po kurios išvestys yra intervale  $[0, 1]$ .

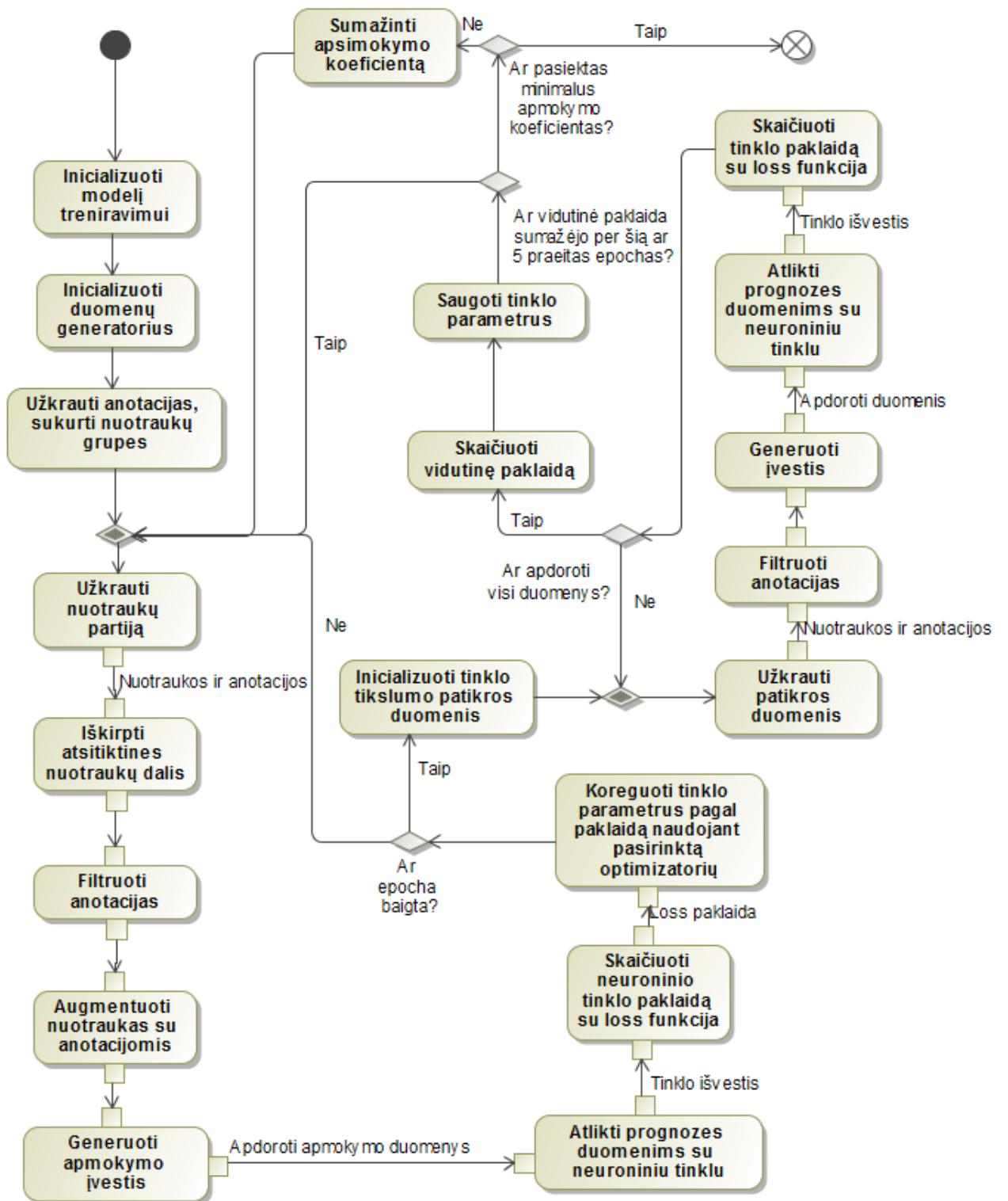
Pavyzdinei  $1024 \times 1024 \times 3$  dydžio įvesčiai neuroninis tinklas grąžina trijų tipų išvestis:

- Karščio žemėlapio išvestį (pavaizduota prie išvesties konvoluciujos bloko 2.5 pav.),  $256 \times 256 \times 1$  dydžio matrica.
- Medžio viršūnių spindulių prognozių  $256 \times 256 \times 1$  dydžio matricą.
- Karščio žemėlapio prognozių 4 pikselių paklaidų  $256 \times 256 \times 2$  dydžio matricą.

Iš viso šis neuroninis tinklas turi 2121748 apmokomus parametrus bei 35940 statiskų parametru.

### 2.3.2. Dinaminis sistemos vaizdas

Šiame skyriuje naudojant UML sekų ir veiksmų diagramas aprašomas sistemos veikimas. Grafiniais vaizdais bei aprašymais paaiškinama vartotojo komunikacija su vartotojo sasajos programa bei algoritmu ir kitos svarbiausios vartotojo sasajos funkcijos, aprašomas kompiuterinės regos algoritmo veikimas prognozių teikimo režime nuo išvesties pateikimo iki išvesties bei algoritmo neuroninio tinklo komponento apmokymas. Kur tinkama, ikeliamos nuotraukos pateiktai informacijai papildyti.



2.6 pav. Konvoliucinio neuroninio tinklo apmokymo veiksmų diagrama.

Pateikta UML veiksmų diagrama aprašo esminio šio kompiuterinės regos algoritmo komponento, konvoliucinio neuroninio tinklo, apmokymo eigą. Šio proceso esmė - pakartotiniai teiki sukurtam konvoliuciniam neuroniniams tinklui apmokymo įvestis, t.y. apdorotas nuotraukas, kurioms neuroninis tinklas generuotų spėjimus, ir skaičiuojant šiu spėjimų paklaidą koreguoti modelio parametrus taip, kad neuroninio tinklo parametrai sudarytų kuo tinkamesnį teikiamų duomenų abstrakcijos modelį tinkamiems spėjimams atliki ateityje pateiktims panašiemis duomenims. Šis

procesas kartojamas tol, kol pasiekiamas aukščiausias galimas neuroninio tinklo tikslumas, papildomas apmokymas nepateikia žymaus pagerėjimo ilgą laiko tarpą.

Apmokymo procesas pradedamas inicializuojant Tensorflow Keras aplinkoje aprašytą neuroninį tinklą, kuris bus apmokomas (galutinei algoritmo versijai naudojamas 2.5 pav. pavaizduotas konvoliucinis neuroninis tinklas). Šiuo metu yra inicializuojamas modelis bei jo optimizatorius (pradedama su Adam (30) optimizavimo algoritmu) su pasirinktu apmokymo koeficientu (pradedama su 0.001 reikšme). Vėliau yra inicializuojami duomenų generatoriai treniravo duomenims generuoti bei patikros duomenims generuoti. Inicializacijos metu taip pat yra užkraunamos apmokymo ir patikros duomenų rinkinių visų nuotraukų anotacijos, sudaromos duomenų partijų grupės, kurių atsitiktinai bus užkraunamos ir pateikiamos apmokymo procesui. Apmokymo procesas yra skaidomas į epochas, t.y. iteruojant apmokomas pasirinktas kiekis duomenų partijų (*batch*), tai dažniausiai visas duomenų rinkinys. Apmokius visą epochą duomenų yra vykdomas modelio tikslumo tikrinimas duomenims, kurie yra panašūs į testavimo bei realius naudojimo duomenis, patikrinus tikslumą modelis yra apmokomas su sekancios epochos duomenimis, tikrinimas tenkina visas sąlygas.

Šiame treniravimo procese vienos treniravimo epochos metu nerertraukiama yra apmokoma 500 duomenų partijų. Šio darbo metu naudojamas duomenų partijos dydis yra 1-2 apmokymui skirtu paveikslėliai (priklasomai nuo apmokomo modelio parametru kiekiui). Vienos partijos apmokymo procesas pradedamas užkraunant partijos nuotraukų duomenis („užkrauti nuotraukų partiją“ veiksmas veiksmų diagramoje). Užkrautoms nuotraukoms iškerpama atsitiktinė nuotraukos dalis, kuri priklasomai nuo iškirpimo dydžio yra sumažinama arba padidinama iki 1024x1024 dydžio nuotraukos. Tai daroma siekiant sugeneruoti fiksuočio dydžio įvestis su skirtingo dydžio medžiais, tokiu būdu kuriamas modelis būna labiau prisitaikęs prie įvairių medžių dydžių bei nuotraukų raiškos. Iš iškirptos nuotraukos yra pašalinamos iš ribų išeinančios anotacijos („filtruoti anotacijas“ veiksmas). Vėliau duomenų partijos nuotraukos yra augmentuojamos šiais nuotraukų apdorojimo veiksmais iš „*albumentations*“ Python bibliotekos:

- Horizontalus apvertimas (50% tikimybė, kad augmentacija bus įvykdыта);
- Vertikalus apvertimas (50% tikimybė);
- RGB kanalų verčių pastūmimas (*red* kanalo iki 15 skaitinės reikšmės pastūmimas, *green* iki 25, *blue* iki 20; 50% tikimybė);
- Atsitiktinis šviesumo ir kontrasto pakeitimas (50% tikimybė);
- Atsitiktinė *gamma* korekcija (50% tikimybė);
- Vienas iš šių suliejimo algoritmų (30% tikimybė):
  - Gauso suliejimas (*Gaussian Blur*);
  - Medianos suliejimas (*Median Blur*);
  - Judesio suliejimas (*Motion Blur*).
- Gauso triukšmo algoritmas (*Gaussian Noise*; 25% tikimybė);
- Pastūmimo, dydžio pakeitimo ir pasukimo algoritmai (*Shift Scale Rotate*; 50% tikimybė).

Šios nuotraukų augmentacijos yra itin svarbi modelio apsimokymo dalis, jomis siekiama išplėsti nedidelio turimo duomenų rinkinio kiekį ir išmokyti tinklą prisitaikyti prie kuo įvairesnių testavimo pavyzdžių, neprieti persimokymo (*overfitting*) būsenos.

Apdorotos nuotraukos su kartu apdorotomis anotacijomis yra normalizuojamos, jų pikselių reikšmių intervalą keičiant nuo [0, 255] į [-1, 1] stabilesniams tinklo apsimokymui. Normalizuotos nuotraukos sugeneruojamos į tinklui skirtą apmokymo įvestį („Generuoti apmokymo įvestis“ veiksmas).

Siekiamas, kad modelis kaip išvesti pateiktą (1) karščio žemėlapį (*heatmap*), nurodantį tikimybes, kad konkrečioje nuotraukos vietoje yra medis (karščio žemėlapio ilgis ir plotis yra 4 kartus mažesni už įvesties paveikslėlio ilgį ir plotį), (2) matricą kiekvienam karščio žemėlapio taškui nurodyti medžio spindulio reikšmę ir (3) 4 pikselių paklaidos vertę kiekvienam karščio žemėlapio taškui kompensuoti už sumažintą karščio žemėlapio rezoliuciją. Šioms CenterNet objekto aptikimo algoritmo išvestims sugeneruoti naudojama viešai pateikama algoritmo apmokymo implementacija (14). Pagal sugeneruotus apmokymo duomenis kiekvienai nuotraukai sugeneruojamas karščio žemėlapis, kiekvienam nuotraukoje pažymėtam medžiui piešiant sulietą apskritimą, kurio suliejimas apskaičiuojamas pagal Gauso pasiskirstymą. Kiekvienam nuotraukoje pažymėtam medžiui apskaičiuojamos koordinacijų vertės karščio žemėlapje ir spindulio reikšmė, kuri bus lyginama su neuroninio tinklo pateikta reikšme. Taip pat kiekvienam pažymėto medžiui apskaičiuojama 4 pikselių paklaida.

**4 lentelė.** Kairėje: modeliui apsimokymui pateikiamos nenormalizuotos nuotraukos pavyzdys; Dešinėje: kairėje pateiktai nuotraukai sugeneruoti apmokymui skirti karščio žemėlapio pavyzdys, išreiškiantis medžių viršūnių centrų koordinates. Nuotraukos šaltinis: Google Earth.



Pateikiant Tensorflow Keras modeliui sugeneruotas apmokymo įvestis, modelis atlieka spėjimus pateiktai nuotraukai ir pateikia minėtas išvestis (karščio žemėlapio matricą, medžio spinduliu matricą, 4 pikselių paklaidos matricą). Gautos modelio reikšmės yra lyginamos su pateiktomis apdorotų nuotraukų įvestimis apdorojant jas naudojantis *loss* funkcijomis. CenterNet tipo objekto aptikimo algoritmui naudojama modifikuota „focal loss“ paklaidos skaičiavimo funkcija:

$$\text{heatmap loss} = \frac{1}{N} \sum_{xy} \begin{cases} (1 - \hat{Y}_{xy})^\alpha \log(\hat{Y}_{xy}), & \text{kai } Y_{xy} = 1 \\ (1 - Y_{xy})^\beta (\hat{Y}_{xy})^\alpha \log(1 - \hat{Y}_{xy}) & \text{kitu atveju} \end{cases}$$

**2.1. funkcija.** *Loss* funkcija karščio žemėlapio paklaidai apskaičiuoti.

Funkcija  $\hat{Y}_{xy}$  naudojama konkrečiai neuroninio tinklo spėjamai karščio žemėlapio reikšmei matricoje aprašyti,  $Y_{xy}$  aprašoma apmokymui pateikiamos nuotraukos (*ground truth*) karščio žemėlapio

konkrečiai reikšmė matricoje, kurią modelis siekia išmokti. Visuose eksperimentuose reikšmės  $\alpha = 2$  bei  $\beta = 4$ .

Kiekvienai apmokymui pateiktai medžio spindulio reikšmei bei 4 pikselių paklaidai tose vietose skaičiuoti tose vietose, kur  $\hat{Y}_{xy} = 1$  naudojama „L1 Loss“ funkcija:

$$L1 \ loss = \sum_{i=1}^n |\hat{Y} - Y|$$

**2.2. funkcija.** *Loss* funkcija medžio spindulio bei 4 pikselių nuotraukoje paklaidai apskaičiuoti.

Galutinė *loss* reikšmė apskaičiuojama sukombinavus visas prieš tai apskaičiuotas *loss* reikšmes:

$$loss = heatmap \ loss + 0.05 * spindulių \ L1 \ loss + pikselių \ paklaidos \ L1 \ loss$$

**2.3. funkcija.** Galutinės *loss* reiškmės apskaičiavimas.

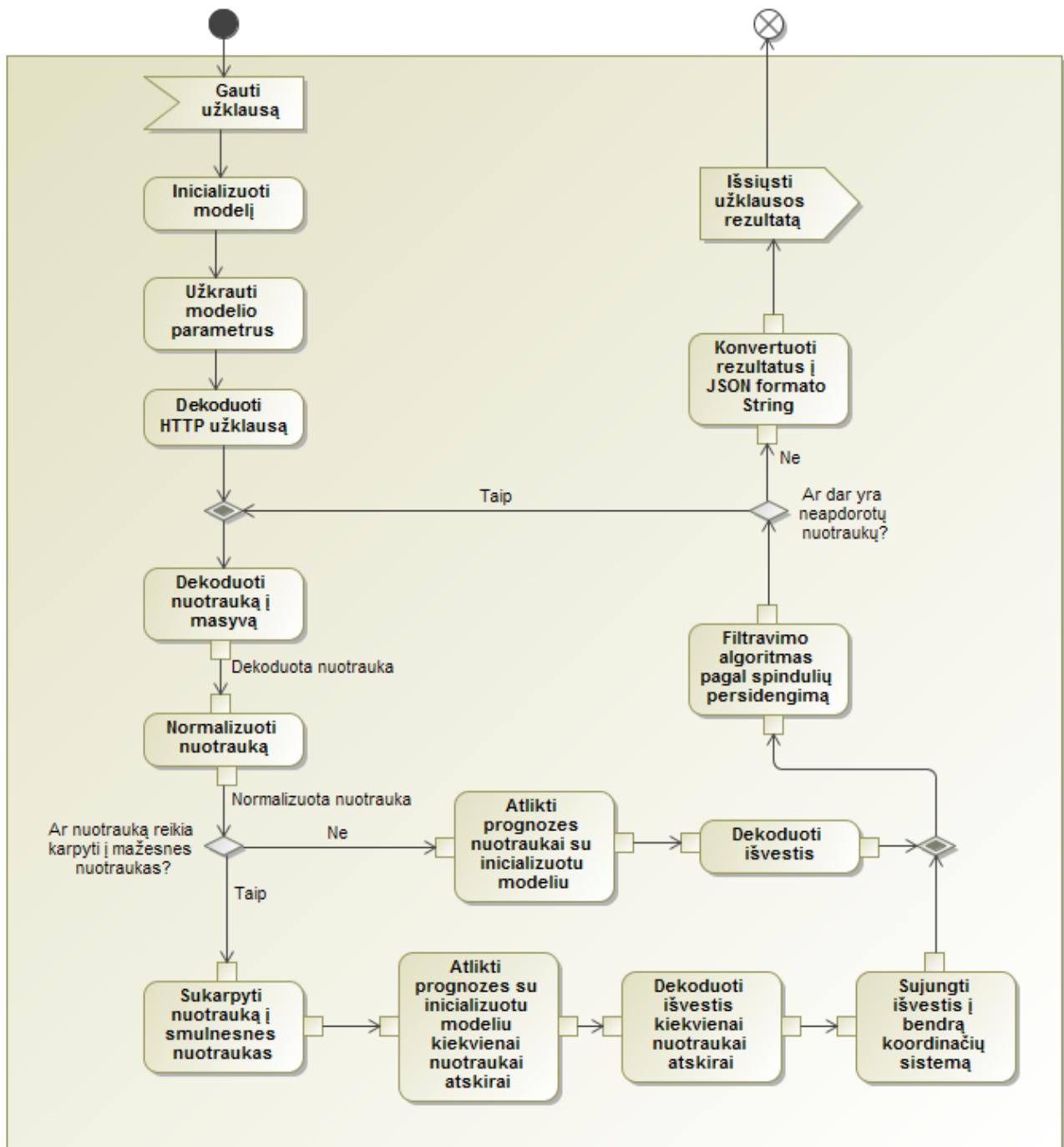
Galiausiai, ši *loss* reikšmė yra pateikiama Tensorflow Keras optimizavimo procesui ir modelis yra optimizuojamas pasirenkamu Tensorflow Keras optimizavimo algoritmu (pradedama su Adam optimizatoriumi) koreguojant neuroninio tinklo parametrus. 2.5 pav. juodai pažymėti neuroninio tinklo sluoksniai yra modelio išvesties sluoksniai, nuo kurių pradedama koreguoti tinklo parametrus. Optimizavimo metu Tensorflow Keras procesas koreguoja konvoluciinių sluoksnių filtrų bei *Batch Normalization* sluoksnių parametrus taip, kad kitą kartą skaičiuojant modelio pateikiamos išvesties *loss* funkcijos reikšmė būtų žemesnė.

Po modelio apmokymo yra tikrinama, ar epocha yra baigta (ar visi 500 paveikslėlių buvo nepertraukiamai apdoroti). Jei ne, visas apmokymo procesas yra kartojamas nuo „Užkrauti nuotraukų partiją“ žingsnio iki kol bus tenkinama sąlyga. Jei sąlyga tenkinama- prasideda modelio tikslumo patikros procesas.

Neuroninio tinklo tikslumo patikros (validacijos) procesas vyksta apdorojant visą validacijai skirtą duomenų rinkinį, jam apskaičiuojant vidutinę *loss* reikšmę. Šio proceso paskirtis yra įvertinti, ar konkrečios epochos apmokymas pagerino modelio tikslumą bei ar tinklas nepateko į persimokymo (*overfitting*) būseną. Validacijos duomenų rinkinys yra kiek įmanoma panašesnis į testavimo bei realaus atvejo duomenis, siekiant kuo objektyviau patikrinti modelio tikslumą. Procesas pradedamas veiksmu „Užkrauti patikros duomenis“, užkraunant validacijos duomenų rinkinio nuotrauką (anotacijos užkrautos generatorius inicializavimo metu). Nuotraukai yra atfiltruojamos neteisingos anotacijos, atliekama duomenų kontrolė. Šiam duomenų rinkiniui augmentacijos nėra atliekamos siekiant išlaikyti duomenų tikrovą. Neuroniniams tinklei sugeneruojamos identiškos išvestys, kaip ir treniravimo metu „Generuoti apmokymo išvestis“ veiksme. Lygiai tokiu pačiu būdu, kaip ir treniravimo metu, neuroninis tinklas atlieka prognozes ir joms yra skaičiuojama bendra *loss* reikšmė. Šis procesas yra kartojamas tol, kol yra apdorojamos visos validacijos duomenų rinkinio nuotraukos, iš joms visoms gautų atskirų *loss* reikšmių yra apskaičiuojamas vidurkis, gaunant vidutinę modelio paklaidą. Šiame etape yra išsaugomi modelio parametrai į „h5“ formato failą, šiuos duomenis galima užkrauti testavimo ar kitame apmokymo procese. Po šio veiksmo yra tikrinama, ar validacijoje gauta vidutinė *loss* reikšmė neaplenkė viso apmokymo metu gautos žemiausios validacijos *loss* reikšmės.

Jei *loss* reikšmė buvo pagerinta- apmokymas tėsiamas pradedant naują epochą nuo „Užkrauti nuotraukų partiją“ veiksmo. Priešingu atveju yra mažinamas apmokymo koeficientas, padalinant jį iš 2. Jei pasiekta žemiausia leidžiama apsimokymo koeficiente reikšmė (šiuo atveju naudojamas 0.0000005 limitas)- treniravimas sustabdomas ir laikoma, kad modelis buvo optimizuotas ir pilnai apmokytas.

Kituose procesuose yra naudojami neuroninio tinklo parametrai, kuriomis apdorojant nuotraukas validacijos etape gauta vidutinė *loss* reikšmė buvo žemiausia.



**2.7 pav.** Nuotraukos apdorojimo medžių viršūnėms rasti veiksmų UML diagrama.

Šioje pateiktoje UML diagramoje vaizduojama Flask Python aplikacijoje prognozavimo režimu paleisto viso kompiuterinės regos algoritmo veiksmų diagrama. Ši sistemos dalis, paleista kaip

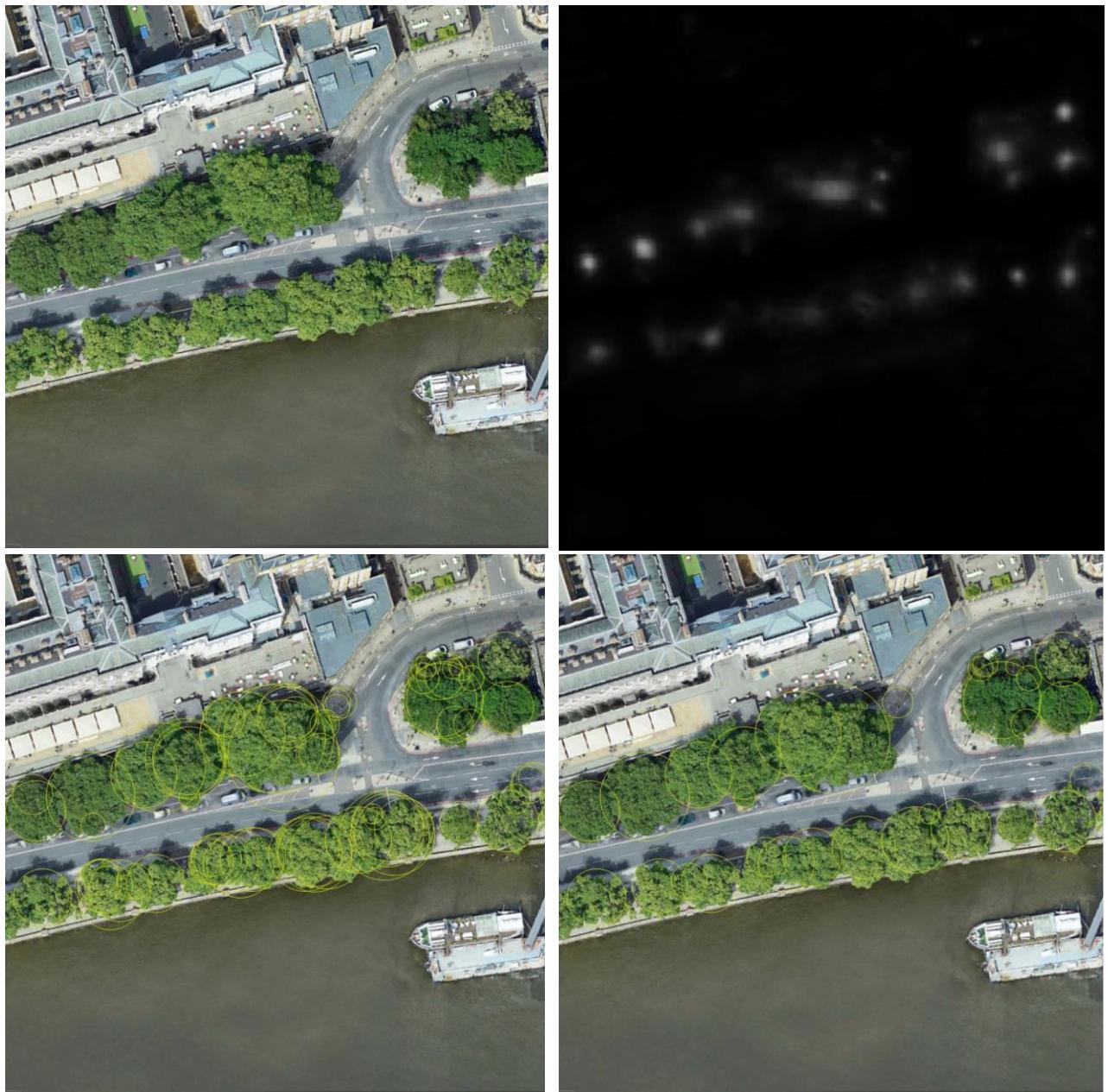
Python programa, laukia HTTP POST užklausos iš vartotojo sąsajos aplikacijos, kuria atsiunčiamos nuotraukos apdorojimui, o apdorojusi nuotraukas gautas prognozes (kokiose pozicijose yra medžiai kiekvienai nuotraukai) siunčia atgal užklausos siuntėjui.

Programa, gavusi užklausą, pirmiausia inicializuojant Tensorflow Keras modelį ir backend sesiją, taip pat užkrauna skaičiavimams atliskti paskirtus modelio parametrus. Po šio veiksmo yra dekoduojama HTTP užklausa iš gautų baitų į atskiras nuotraukas.

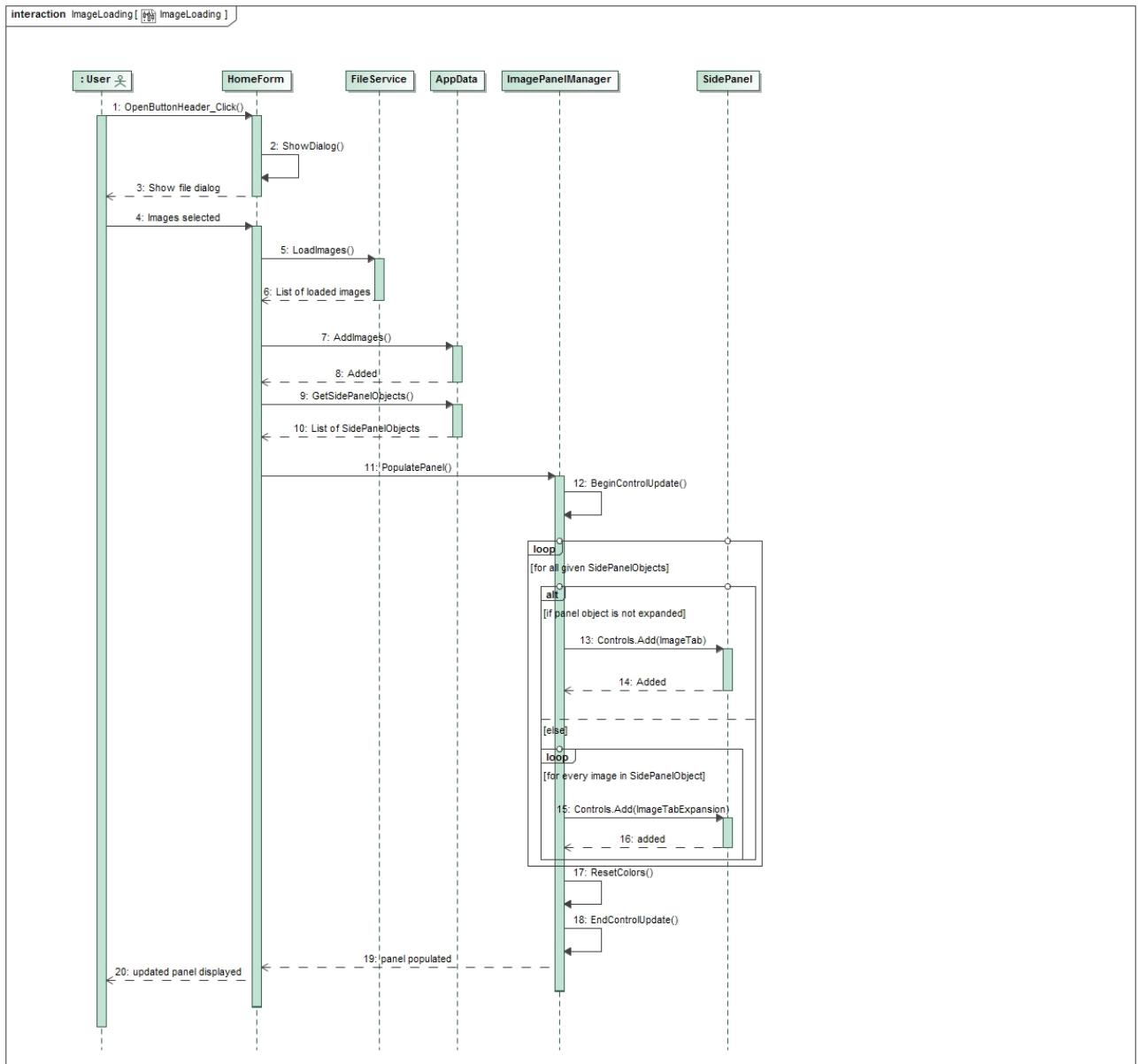
Pradedant nuotrauką apdorojimą, kiekviena nuotrauka yra konvertuojama į Python bibliotekos numpy masyvą, kurį gebėtų priimti inicializuotas modelis prognozēmams atliskti. Kiekvienos nuotraukos pikselių reikšmės prieš pateikimą modeliui yra normalizuojamos iš intervalo [0, 255] į [-1, 1]. Taip pat šiame etape yra nusprenaudžiama, kokiu būdu pateiktai nuotraukas neuroniniam tinklui: originalaus dydžio ar sukarpytas. Jei nuotraukos kažkuri briauna viršija 1536 pikselių ilgį - nuotrauka karpoma į mažesnes, iki 1024 ilgio arba pločio smulkesnes dalis. Vėliau nuotraukos yra pateikiamas modeliui atliskti spėjimus atskirai. Prieš pateikiant duomenis Tensorflow Keras modeliui, kiekviena nuotrauka, nepaisant to, kokiu būdu buvo apdorojama prieš tai, yra papildomai sumažinama iki tokio ilgio ir pločio, kad nuotraukai, pateiktai modeliui, būtų galima 5 kartus atliskti dvigubą dimensijų mažinimą (*stride* reikšmė 5 konvoluciiniuose sluoksniuose yra 2 ilgiui ir pločiui). Šiuo atveju, kiekviena nuotrauka yra sumažinama iki artimiausio ilgio ir pločio skaičiaus, kuris dalintuosi iš 32. Kitu atveju nuotrauka negali būti pateikiama modeliui, kadangi atliekant konvoluciujos operacijas bus gaunamos trupmeninės ilgio arba pločio vertės. Gautos modelio karščio žemėlapio (*heatmap*), ilgio ir pločio bei 4 pikselių paklaidos išvesčių matricos yra dekoduojamos į apskritimų formatą „Dekoduoti išvestis“ arba „Dekoduoti išvestis kiekvienai nuotraukai atskirai“ veiksmuose, apskritimą išreiškiant x ir y koordinatėmis įvesties paveikslėlio koordinačių sistemoje, spindulio reikšme bei modelio užtikritumo įverčiu, išreiškštū reikšme intervale [0, 1]. Dekodavimas atliekamas randant aukščiausias tikimybų vertes karščio žemėlapyje, joms priskiriant spindulio reikšmes tose iš kitos įvesties matricos tose pačiose koordinatėse bei taip pat koreguojant kiekvieno rasto medžio apskritimo poziciją 4 pikselių paklaidos kiekvienai karščio žemėlapio reikšmei įvesties matricos reikšmes. Itin glaudžiai esantys apskritimai yra konvertuojami į keturkampius ir filtruojami *non-maximum suppression* algoritmu. Veiksme „Sujungti išvestis į bendrą koordinačių sistemą“ kiekvienos sukarpytos nuotraukos spėjimai konvertuojami iš lokalios nuotraukos koordinačių sistemos į bendrą įvesties nuotraukos koordinačių sistemą, jei apdorojama nuotrauka prieš tai buvo sukarpta į smulkesnes nuotraukas ir pateikta modeliui atskirai. Galiausiai, nepaisant to ar nuotrauka buvo karpoma apdorojimui, glaudžiai esantys apskritimai yra pakartotinai filtruojami pagal jų spindulių persidengimą, šiuo procesu užbaigiant vienos nuotraukos apdorojimą, randant joje medžių viršunes, išreiškštās apskritimų.

Praeitoje pastraipoje aprašytas procesas yra kartojamas tol, kol apdorojamos visos nuotraukos. Galiausiai, turint visų nuotraukų dekoduotas išvestis, rastos medžių viršunių koordinatės yra konvertuojamos į JSON formato string reikšmę ir išsiunčiamos HTTP užklausos siuntėjui kaip užklausos rezultatas, tuo baigiant nuotrauką, gautą iš vartotojo sąsajos aplikacijos apdorojimą.

**5 lentelė.** Kairėje viršuje: originali įvesties nuotrauka; Dešinėje viršuje: neapdorotos karščio žemėlapio išvestys pateiktai įvesties nuotraukai; Kairėje apačioje: dekoduotos išvestys pateiktai įvesties nuotraukai neatlikus filtravimo pagal apskritimų spindulių persidengimus; Dešinėje apačioje: pilnai apdorotos išvestys. Nuotraukos šaltinis: Google Earth.

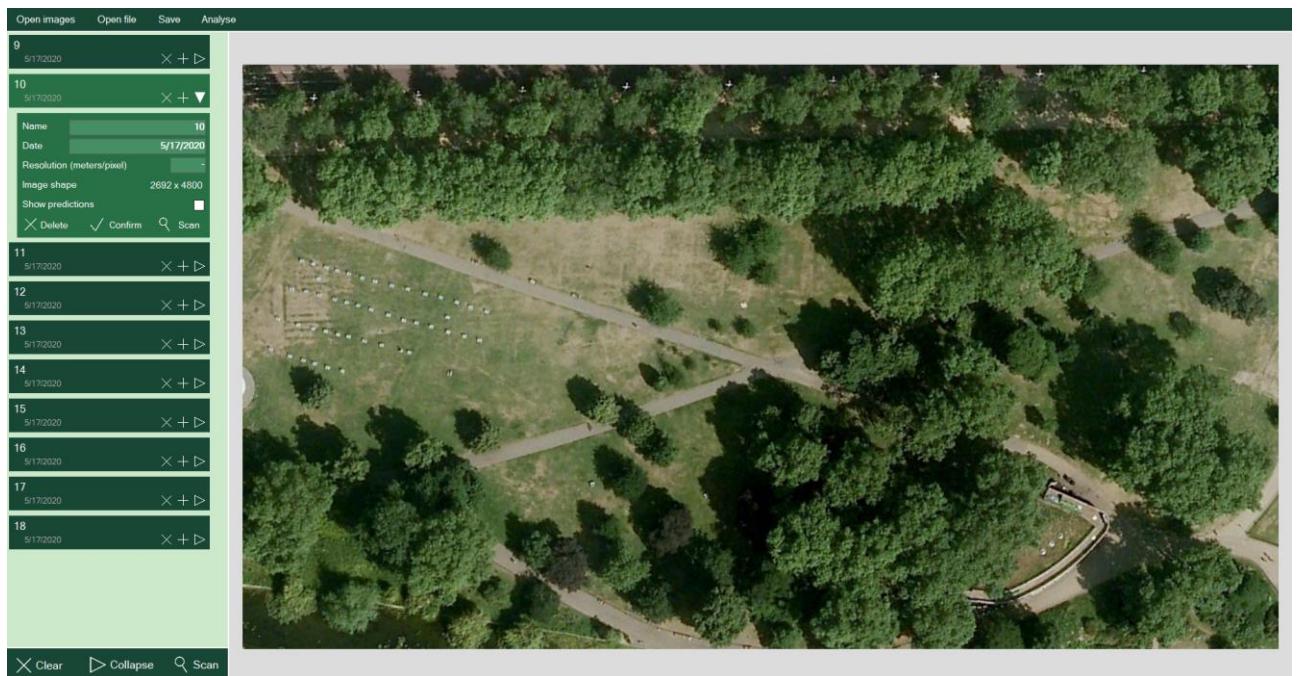


Žemiau pateiktoje medžiagoje aprašomas vartotojo sąsajos aplikacijos veikimas. Sistemos veikimas vaizduojamas UML sekų diagramomis, aprašant nuotraukų užkrovimą į aplikaciją, nuotraukų apdorojimą naudojantis kompiuterinės regos algoritmu, rezultatų saugojimą bei analizės atlikimą. Žemiau pateikiama nuotraukų užkrovimo sekų diagrama:



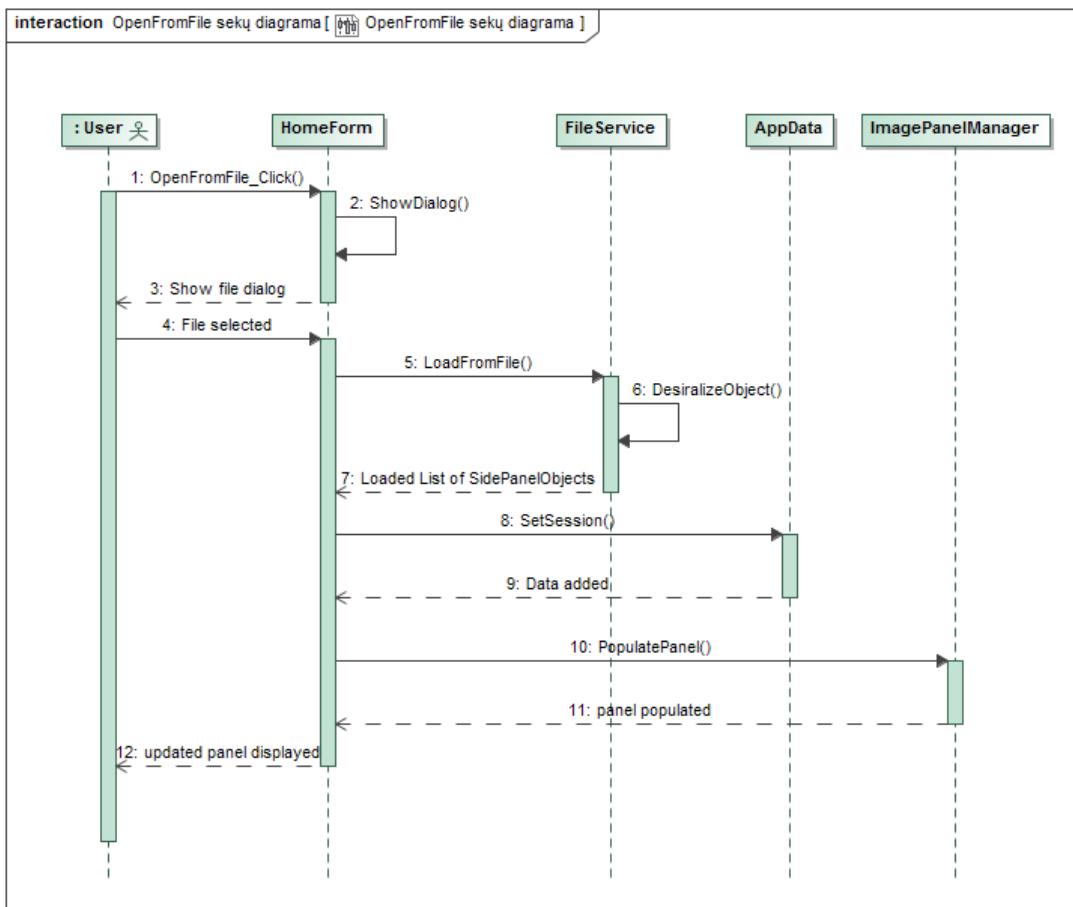
**2.8 pav.** Nuotraukos užkrovimų iš failų sistemos vartotojo sėsajoje UML sekų diagrama.

Šioje sekų diagramoje vaizduojamas nuotraukų užkrovimo ir rezultato vizualizavimo procesas vartotojo sėsajoje. Paspaudus „Open“ mygtuką vartotojo sėsajoje atidaromas failų dialogas, kuriam vartotojas gali pasirinkti failus, kuriuos nori atidaryti. Naudojantis „FileService“ klase nuotraukos yra užkraunamos ir aprašomos kaip Image klasės objektas. Galimos „.jpg“ ir „.png“ tipų nuotraukos. Image klasės objektai yra pridedami į AppData klasės objektą, kur gali būti pasiekiami kitų klasių. Vėliau duodama komanda ImagePanelManager klasės objektui iš Services paketo atvaizduoti vartotojo sėsajoje skiltis nuotraukų, kur vartotojas gali pasirinkti, kokias nuotraukas nori stebėti ir atliskti kitas funkcijas. Šios nuotraukų skiltys (ImageTab ir ImageTabExtension klasių objektai) yra pridedamos į SidePanel FlowLayout objekto Windows Forms lange, kur yra vaizduojamos vartotojui. 2.9 pav. pavaizduotos užkrautos nuotraukos, kurių skiltys yra sudėtos į kairėje esantį FlowLayout objekto. Vartotojas, paspaudęs ant norimos skilties, gali pasirinkti, kokią nuotrauką nori paržiūrėti. Pasirenkant išplėtimo mygtuką galima iškleisti ImageTab objekto, pateikiant toje skiltyje esančiu nuotraukų išplėstinių duomenų skiltis.



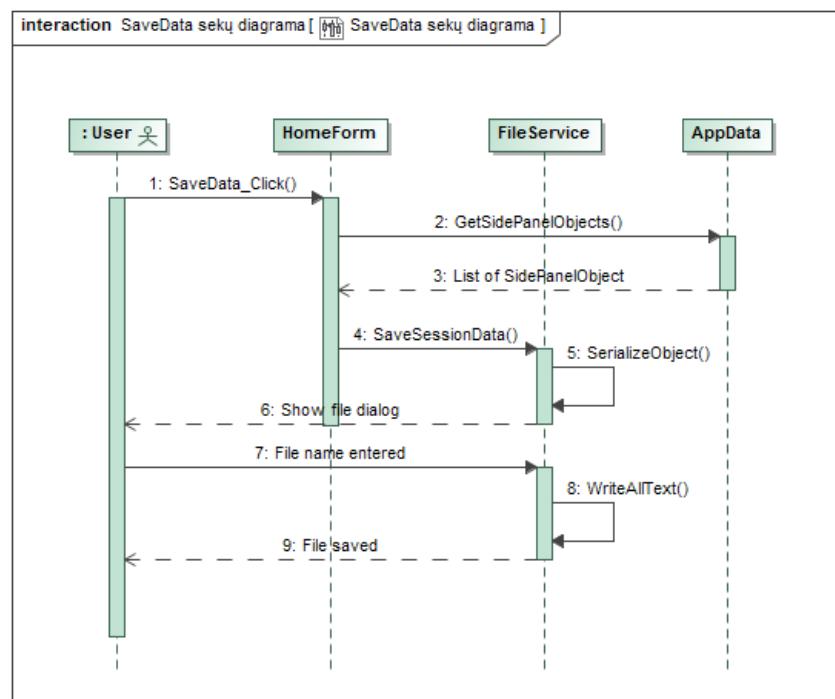
**2.9 pav.** Vartotojo sąsaja su užkrautomis nuotraukomis. Nuotraukos šaltinis: Google Earth.

Duomenis taip pat galima užkrauti iš „json“ formato failo. Šio proceso metu užkraunami programos duomenys, kurie buvo išsaugoti ankščiau, praeito naudojimo metu. Užkraunant duomenis šiuo būdu užkraunami visi SidePanelObject objektai su savyje esančių Image klasės objektų duomenimis. Kiekvieno Image objekto nuotrauka yra užkraunama pagal „path“ *string* tipo kintamąjį, išsaugotą „json“ faile konkrečiam Image objektui. Užkrauti duomenys pridedam iš AppData klasę, kad būtų pasiekiami kitoms programos dalims.

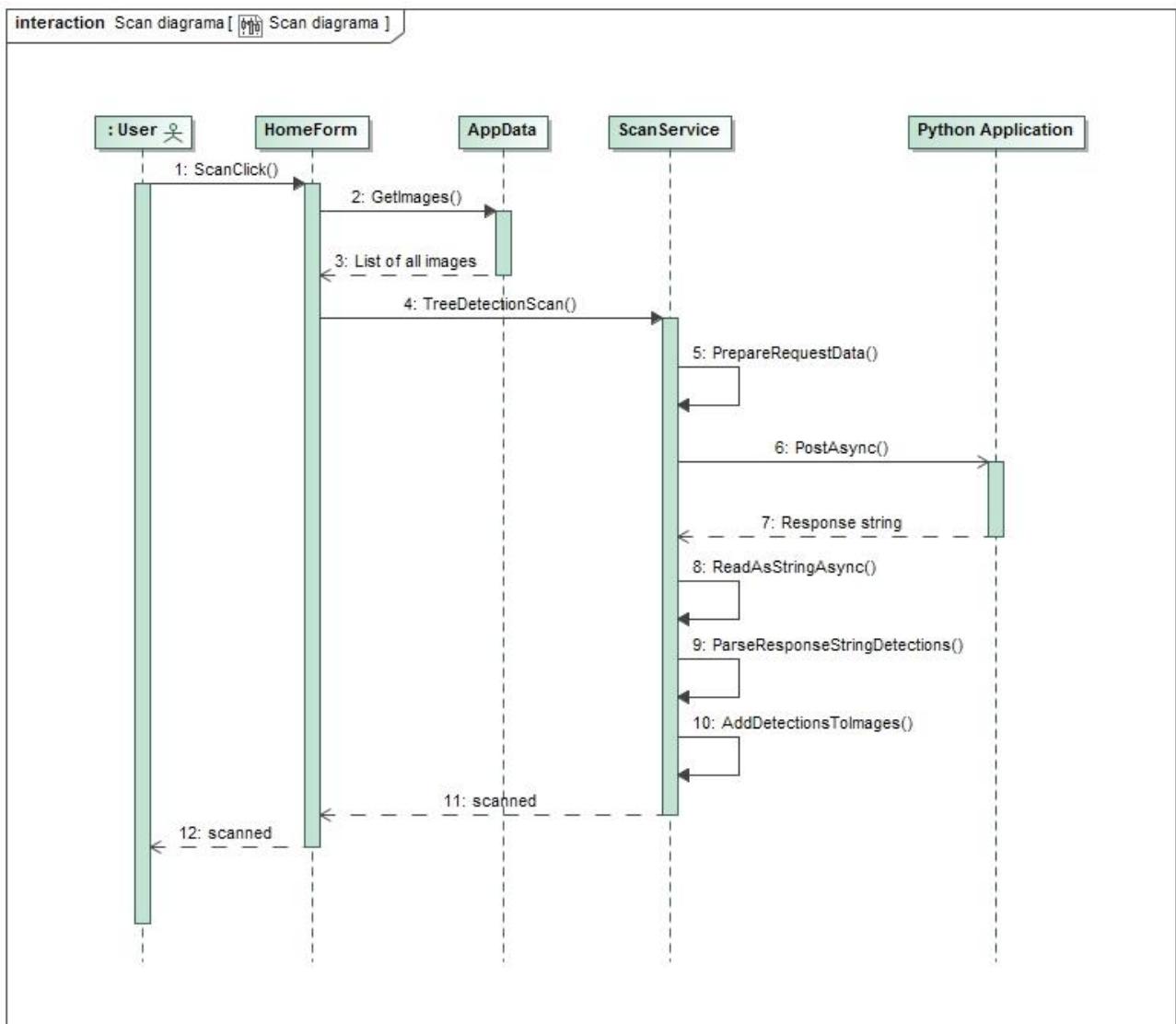


**2.10 pav.** Duomenų užkrovimo iš „json“ formato failo UML sekų diagrama.

Vartotojui užkrovus nuotraukas, pakeitus jų parametrus, apdorojus su kompiuterinės regos algoritmu, šiuos duomenis galima saugoti „json“ tipo faile (2.11 pav.), kuriuos kartu su nuotraukomis vėliau galima užkrauti pagal 2.10 pav. pateikiamą veiksmą.

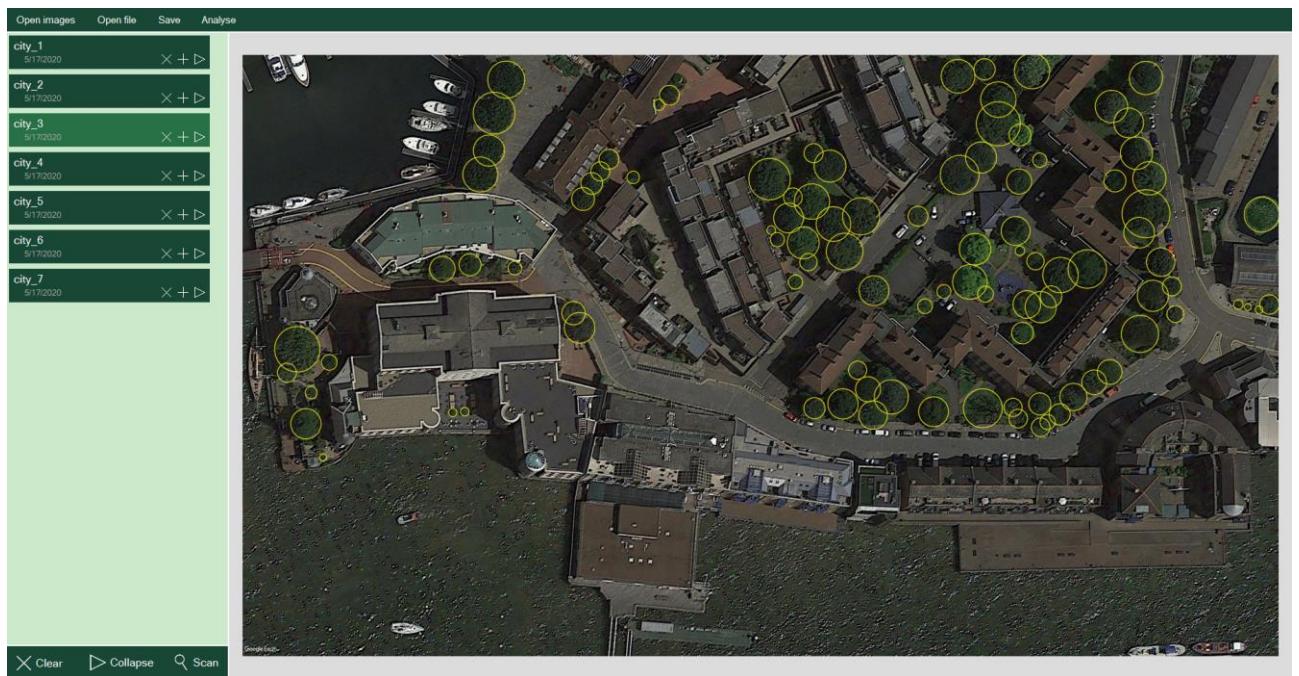


**2.11 pav.** Duomenų saugojimo „json“ formatu UML sekų diagrama.

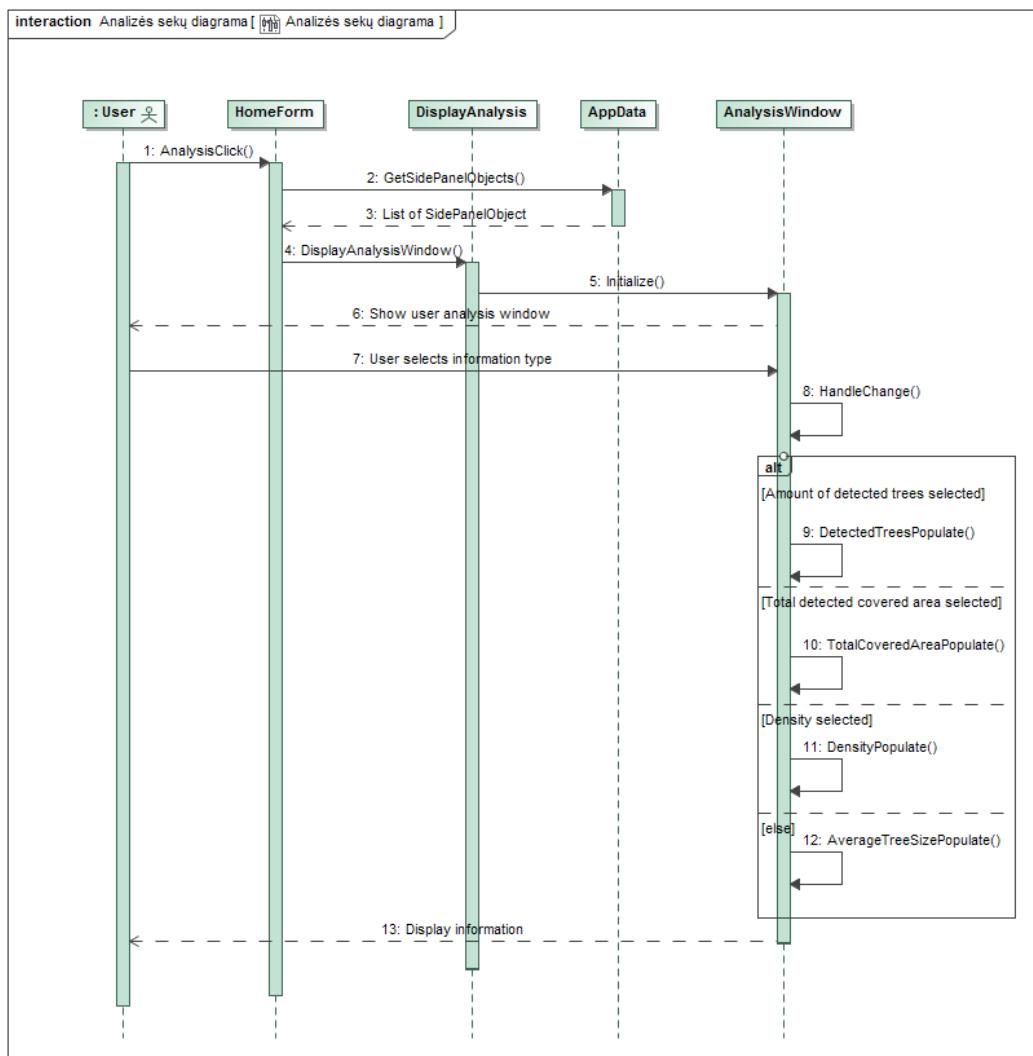


**2.12 pav.** Nuotraukų apdorojimo su kompiuterinės regos algoritmu medžių viršūnėms rasti proceso UML sekų diagrama.

Šioje sekų diagramoje aprašomas nuotraukų apdorojimo procesas, pradedant jį nuo vartotojo sasajos. Vartotojui paspaudus skanavimo mygtuką pagrindiniame lange AppData klasės objektas HomeForm Windows Forms objektui pateikia visas aplikacijoje talpinamas nuotraukas. Iš šių nuotraukų yra paruošiama užklausa, konvertuojant nuotraukų duomenis į baitų masyvą ir siunčiant HTTP POST užklausą modelio Python aplikacijai, kuri vėliau apdoroja visas nuotraukas pagal 2.5 pav. pavaizduotą veiksmų diagramą. Gautą atsakymą programa konvertuoja į TreeDetection klasės objektus, juos talpina List duomenų struktūroje ir priskiria Image klasės objektams. Apdorojimo rezultatai vaizduojami ekrane pasirinktame paveikslėlyje (2.13 pav.).



**2.13 pav.** Nuotraukų apdorojimo su kompiuterinės regos algoritmu medžių viršūnėms rasti rezultatai, pateikiami peržiūrai aplikacijoje. Nuotraukos šaltinis: Google Earth.



**2.14 pav.** Informacijos vaizdavimo analizės lange UML sekų diagrama.

2.14 pav. pateikiama UML sekų diagrama nuotraukos analizės lango ir jo funkcijoms vizualizuoti. Vartotojas paspaudę „Analyse“ mygtuką programos viršuje esančioje svarbiausių funkcijų antraštės juosteje (5.1 pav.) atidaro analizės langą (3.4 pav.). Jame pateikiama paprasta statistinė informacija apie visas apdorotas nuotraukas atskirai bei visas kiekvieno skirtuko naujausias nuotraukas kartu. Šiame lange galima gauti informaciją apie rastų medžių viršūnių kiekį įvairiems viršūnių dydžiams, uždengtos teritorijos plotą, tankį, vidutinį medžių dydį ir kitą informaciją.

Nuotraukas analizuoti taip pat galima tam tikrame laiko kontekste. Nuotraukos, sudėtos į tą patį skyrių kairėje esančioje nuotraukų aprašų skiltyje yra apdorojamos kaip tos pačios teritorijos nuotraukos, tačiau atliktos kitu laiku. Tokiu būdu galima ieškoti pokyčių, pavyzdžiui, nukirstų medžių. Išskleistoje nuotraukų skiltyje pasirinkta nuotrauka vaizduojama vartotojo ekrane geltonai žymint algoritmo rastas medžių viršūnes bei raudonai žymint vietas, kuriose potencialiai buvo nukirstas medis.



**2.15 pav.** Nuotraukų analizavimas laike raudonais apskritimais žymint potencialiai nukirstų medžių vietas.  
Nuotraukos pavyzdys: Google Earth.

### 3. Testavimas

#### 3.1. Testavimo planas

Šiame darbe testuojama sukurto kompiuterinės regos algoritmo implementacija bei vartotojo sąsajos programa. Abi sistemos dalys turi atskirus testavimo kriterijus ir testavimui atliliki naudojami skirtini testavimo metodai. Šioms programoms atliekami šie testavimai:

- **Komponentų testavimas kompiuterinės regos algoritmui.** Šio testavimo metu atliekamas svarbiausių kompiuterinės regos funkcijų (komponentų) testavimas, siekiant išsiaiškinti skaičiavimų korektiškumą ir patikrinti veikimą įvairiais atvejais. Šio testavimo metu testuojami komponentai, kurie įvesčiai priima ir išvestyje pateikia nedidelį duomenų kiekį, kurį lengva patikrinti atliekant komponentų testavimą.
- **Vizualus kompiuterinės regos algoritmo testavimas.** Šio testavimo metu tam tinkamos funkcijos ir algoritmo galutinės ar tarpinės išvestys yra įvertinamos vizualiai, išsaugant jas faile arba vaizduojant ekrane veikimo metu. Stebint algoritmo išvestis siekiama įvertinti, ar kompiuterinės regos algoritmo apdorojimo veiksmai vykdomi teisingai. Šio testavimo metu testuojami komponentai, kurių testavimas komponentų testavimo būdu būtų nepraktiškas dėl didelio apdorojamo duomenų kiečio, bei tam tikras algoritmo daromas klaidas yra per sunku pastebeti nevizualizuojant gautų rezultatų. Šis testavimas taip pat yra itin svarbus norint tinkamai apmokyti neuroninį tinklą, kadangi neteisingai apmokomas tinklas gali pateikti itin mažai indikacijų, kad apmokymui pateikiami netikslūs duomenys, tačiau apmokymo rezultatas gali būti prastesnis arba visiškai netinkamas ir tokiu būdu neišnaudojamas visas algoritmo potencialas.
- **Rankinis vartotojo sąsajos testavimas.** Šio testavimo metu įvertinama, ar vartotojo sąsaja pateikia tokius rezultatus, kurių buvo tikimasi ją kuriant, bei ar pateikiami rezultatai yra tinkami vartotojui bei patogiam naudojimui.

Visos testavimo dalys atliekamos kompiuteryje su *AMD Ryzen 7 3750H* procesoriumi, 16 GB *RAM* atminties bei *Nvidia RTX 2060* vaizdo plokšte. Kompiuteryje įdiegta *Windows 10* operacinė sistema. Kompiuterinės regos algoritmo testavimas atliekamas *Python 3.7* programavimo kalbos aplinkoje naudojantis *PyCharm* programavimo aplinka bei *unittest* biblioteka. Vartotojo sąsajos testavimas atliekamas naudojantis *Visual Studio* programavimo aplinka.

#### 3.2. Testavimo kriterijai

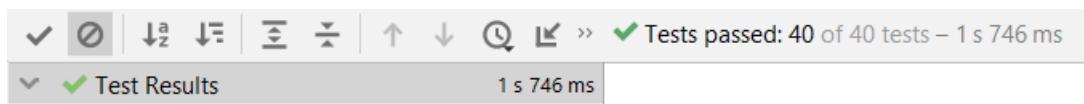
Atskiriems testavimo metodams taikomi šie kriterijai ir siekiama tokią tikslų:

- Komponentų testavimas atliekamas tik nedidelį duomenų kiekį apdorojantiems metodams. Netestuojama didžioji dalis funkcijų, kurios apdoroja ir kitaip dirba su nuotraukomis. Taip pat netestuojamos funkcijos, kurios buvo aprašytos pradinėje objekto aptikimo algoritmo implementacijoje. Taip pat netestuojamos funkcijos, kurios grąžina *TensorFlow* karkaso duomenų tipus, atlieka apmokymo užduotis ar aprašo modelį, kadangi tokų funkcijų veikimo teisingumą yra per daug sudėtinga įvertinti komponentų testavimu bei šių funkcijų testams korektiškai inicializuoti duomenų įvestis. Komponentų testavimu testuojama nedidelė bendro kodo dalis.

- Vizualų kompiuterinės regos algoritmo testavimą siekiama atlikti kiek įmanoma didesnei kompiuterinės regos algoritmo daliai. Siekiama atlikti testavimą tam tikruose algoritmo etapuose taip, kad būtų galima apytiksliai įvertinti visų praeitų apdorojimo etapų veikimą nuo praeito vizualaus išvesties įvertinimo.
- Rankinį vartotojo sąsajos testavimą siekiama atlikti visoms vartotojo sąsajos funkcijoms, patikrinant visų jos funkcijų veikimą.

### 3.3. Komponentų testavimas

Komponentų testavimas buvo atliekamas naudojant *PyCharm* programavimo aplinką su *unittest* biblioteka, testus atliekant *Python 3.7* programavimo kalbos kodui. Šioje dalyje testuojamos tik svarbiausios kompiuterinės regos algoritmo programos dalys, kurios neapdoroja didelio informacijos kieko vienu metu, pavyzdžiui nuotrauką, neima kaip įvesties arba negrąžina *TensorFlow* karkaso tipų išvesties, pavyzdžiui modelio ar jo sluoksnių. Šio testavimo metu buvo atlikta 40 komponentų testų, kurių pagalba buvo rastos ir sėkmingai ištaisytos klaidos trijose funkcijose.



**3.1 pav.** Komponentų testavimo įvykdymo rezultatas.

### 3.4. Vizualus testavimas

Šis testavimas atliekamas išvedant tarpines ar galutines išvestis apmokant ar testuojant kompiuterinės regos modelį. Testavimas atliekamas naudojantis *Python 3.7* kalba bei *OpenCV* biblioteka nuotraukų saugojimui, pateikimui ekrane bei figūrų piešimui. Šis testavimo metodas atliekamas testuoti didelius duomenų kiekius apdorojančias funkcijas, kurioms ranka rašyti testavimo duomenų rinkinius yra nepraktiška. Kadangi yra apdorojami vaizdiniai duomenys- ši savybė yra išnaudojama kontrolei atlikti vizualizuojant. Šio testavimo metu siekiama gauti apytikslius rezultatus bei iškelti hipotezes netinkamo veikimo atveju. Kitose pastraipose pateikiami rezultatai yra tik pavyzdiniai, realus testavimas buvo atliekamas su didesniu kiekiu duomenų, kuris leido patvirtinti duomenų bei jų apdorojimo korektiškumą. Vizualizacijoms naudojama pavyzdinė nuotrauka iš *Google Earth* programos.

Testavimas pradedamas patikrinant, ar duomenys, naudojami apmokymui yra teisingi bei teisingai užkraunami. Tinkama, ar tinkama yra apmokymui skirta nuotrauka bei anotacijos.



**3.2 pav.** Duomenų užkrovimo vizualizacija.

Šio vizualaus įvertinimo metu matoma, kad duomenys yra užkraunami korektiškai, naudojamas duomenų formatas yra teisingas, medžių viršūnių žymėjimai nėra paslinkti į kurią nors pusę, spinduliai atitinka medžių dydžius. Testuojant didesniam duomenų kiekiui taip pat įvertinamas apytikslis duomenų žymėjimo klaidų kiekis.

Sekančiame apmokymo duomenų generavimo apdorojimo etape iš didesnės nuotraukos atsitiktinai yra iškerpama mažesnė nuotraukos dalis. Šio proceso metu yra iškerpama 1024x1024 pikselių dydžio nuotraukos dalis, kerpamą dydį prieš tai atsitiktinai padidinus 0.5-2 karto, taip siekiant padidinti modelio prisitaikymą prie nuotraukų iš įvairaus aukščio ir įvairios raiškos.

**6 lentelė.** 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų karpymo etapo.





Stebint šio etapo išvestis nusprendžiama, ar karpymo metoda veikia teisingai, neigiamai nepaveikia nuotraukos žymėjimų bei ar nuotraukų kirpimo dydis yra tinkamas.

Iškarpytos nuotraukos yra apdorojamos augmentacijos funkcijomis naudojantis *albumentations* biblioteka, naudojamos funkcijos aprašytoje projekto dalyje.

**7 lentelė.** 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų augmentacijos etapo.





Vizualizuojant augmentuotas išvestis siekiama išsiaiškinti, ar augmentacijos yra tinkamos mokymui, ar apmokymas tokiais duomenimis bus tinkamas modeliui prisitaikyti prie sudėtingesniu variantų, ar augmentacijos nėra perteklinės bei ar nesugadina nuotraukos žymėjimų.

Apdorotos nuotraukos yra siunčiamos filtravimui pašalinant anotacijas (medžių viršūnių apskritimus), kurių centrai nėra apdorotoje nuotraukoje.

**8 lentelė.** 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų filtravimo etapo.

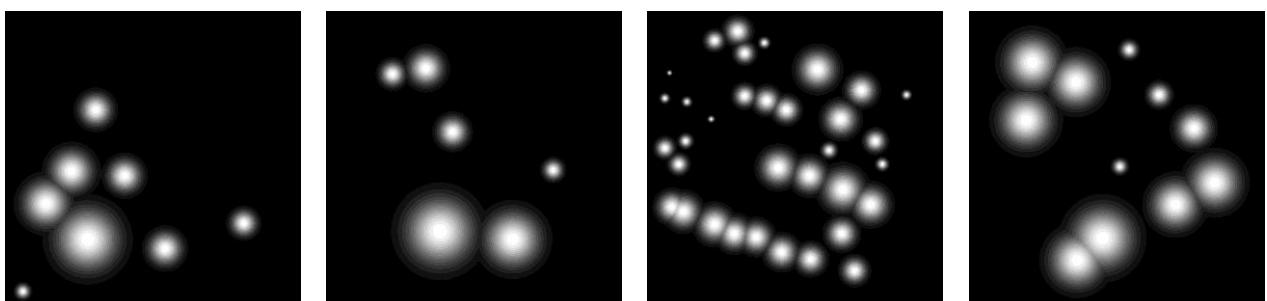




Stebint filtruotas nuotraukas siekiama išsiaiškinti, ar filtravimas yra pakankamas, ar pašalinamos visos nereikalingos anotacijos bei ar nereikalingi papildomi filtravimo kriterijai. Pateikiant modelio apmokymui nefiltruotas išvestis apmokymo procesas veikia ir yra identiškas, tačiau modelis išmoksta grąžinti karščio žemėlapio (*heatmap*) reikšmes su itin aukštomis pozityvumo reikšmėmis nuotraukų pakraščiuose bet kokiai įvesčiai.

Apmokymo duomenų generavimo pabaigoje yra tikrinamos apmokymui sugeneruotos duomenų įvestys, išreikštos kaip karščio žemėlapis, sužymėtų medžio viršunių spindulių reikšmės bei 4 pikselių paklaidos reikšmės kiekvienai karščio žemėlapio reikšmei.

**9 lentelė.** 3.2 pav. pateiktos pavyzdinės nuotraukos vaizdavimas po apmokymo nuotraukų įvesties karščio žemėlapių generavimo etapo.



Tai yra galutinė duomenų išraiška prieš jos pateikimą apmokymui. Šių duomenų vizualus tikrinimas yra pats svarbiausias, kadangi jis apima didžiausią kiekį operacijų nuo duomenų generavimo pradžios, šiame apmokymo duomenų generavimo etape lengviausia pastebėti anksčiau padarytas programines klaidas. Taip pat šiame etape tinkamiausiai galima įvertinti, ar modelis gauna tinkamą informaciją apsimokymui bei numatyti galimas apsimokymo klaidas ir pagal tai atlikti pakeitimus.

**10 lentelė.** 3.2 pav. pateiktos pavyzdinės nuotraukos anotacijų spindulių bei 4 pikselių paklaidos reikšmės po nuotraukos apdorojimo apmokymo įvesčiai.

	0	1
0	0.60583	0.47015
1	0.26274	0.69939
2	0.38632	0.40628
3	0.92456	0.77022
4	0.09084	0.89722
5	0.85754	0.79586
6	0.12000	0.55470
7	0.30615	0.90921
8	0.43854	0.85170
9	0.00000	0.00000
10	0.00000	0.00000
11	0.00000	0.00000
12	0.00000	0.00000
13	0.00000	0.00000
14	0.00000	0.00000
15	0.00000	0.00000
16	0.00000	0.00000
17	0.00000	0.00000

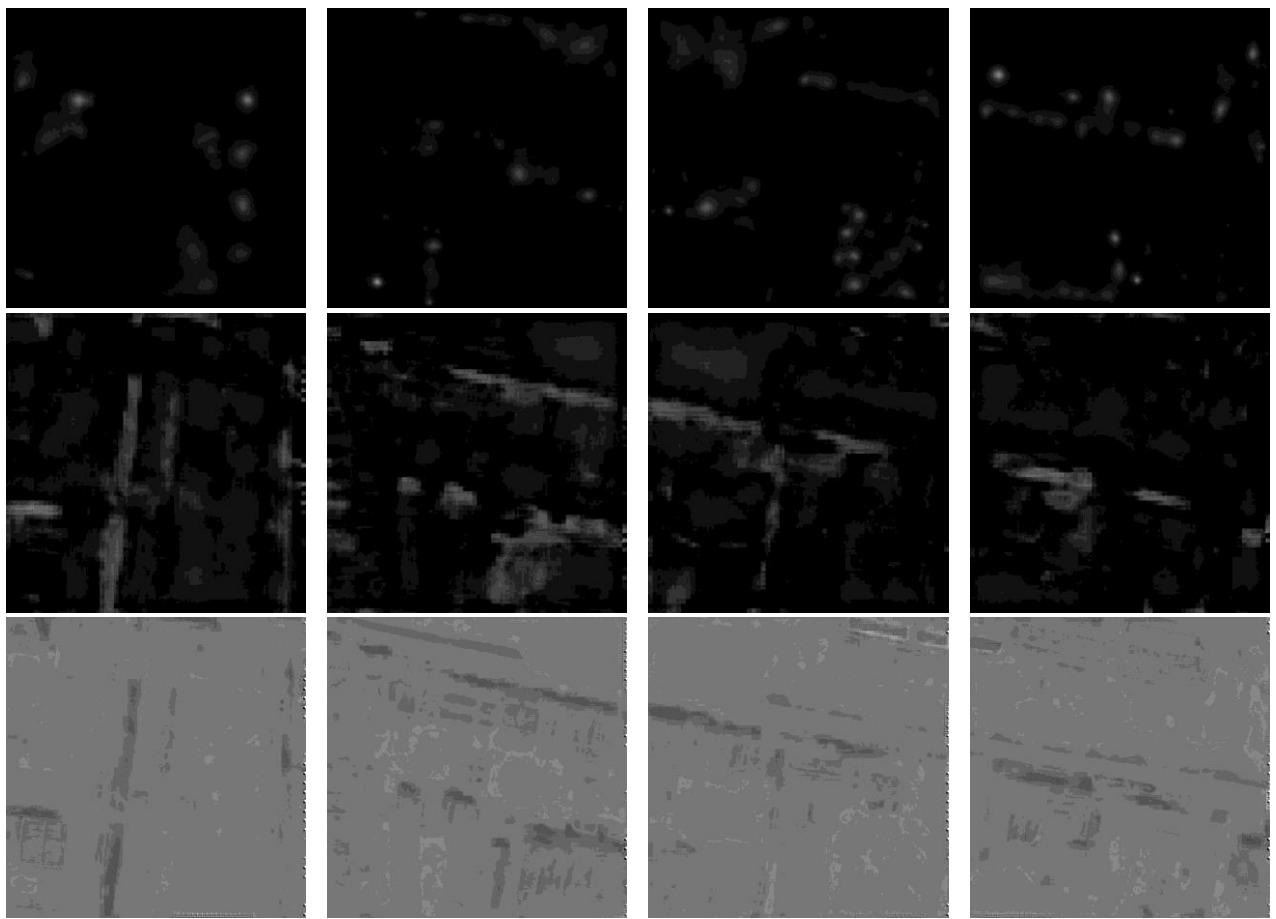
	0
0	18.70228
1	20.36470
2	30.33925
3	14.13061
4	40.72941
5	40.72941
6	31.55470
7	10.39016
8	9.55894
9	0.00000
10	0.00000
11	0.00000
12	0.00000
13	0.00000
14	0.00000
15	0.00000
16	0.00000
17	0.00000

Naudojantis *PyCharm* programavimo aplinkos *Debug* įrankiu taip pat vizualizuojami apmokymui skirtų galutinių sugeneruotų medžių viršunių spindulių bei 4 pikselių paklaidų reikšmės. Šios reikšmės yra sugeneruojamos tuo pačiu metu, kaip ir karščio žemėlapiai, ir tai yra galutinis jų formatas prieš pateikimą apmokymui. Šiuo metu taip pat yra stebimos ir kitos įvestys, skirtos apmokymui (anotacijų indeksai masyve, anotacijų kaukių (*masks*) masyvai ir kt.). Stebint šias reikšmes ieškoma reikšmių neatitikimų su karščio žemėlapio anotacijomis bei stebima, ar reikšmės tinkamos modelio apmokymui.

Dalį klaidų galima aptikti ne tik stebint modelio įvesčių generavimą, tačiau stebint ir apmokyto modelio išvestis. Šio etapo metu stebimos pilnai dekoduotos išvestys bei nedekoduotos karščio žemėlapio, prognozuojamų spindulių bei 4 pikselių paklaidų matricų reikšmės. Šio testavimo metu siekiama aptikti programines klaidas, dėl neatidumo praleistus dalykus ar kontrolės dalis, taip pat modelio apmokymo klaidas, kurios modeliui neleidžia apsimokyti tinkamai ir gaunami rezultatai yra prastesni, nei turėtų būti.

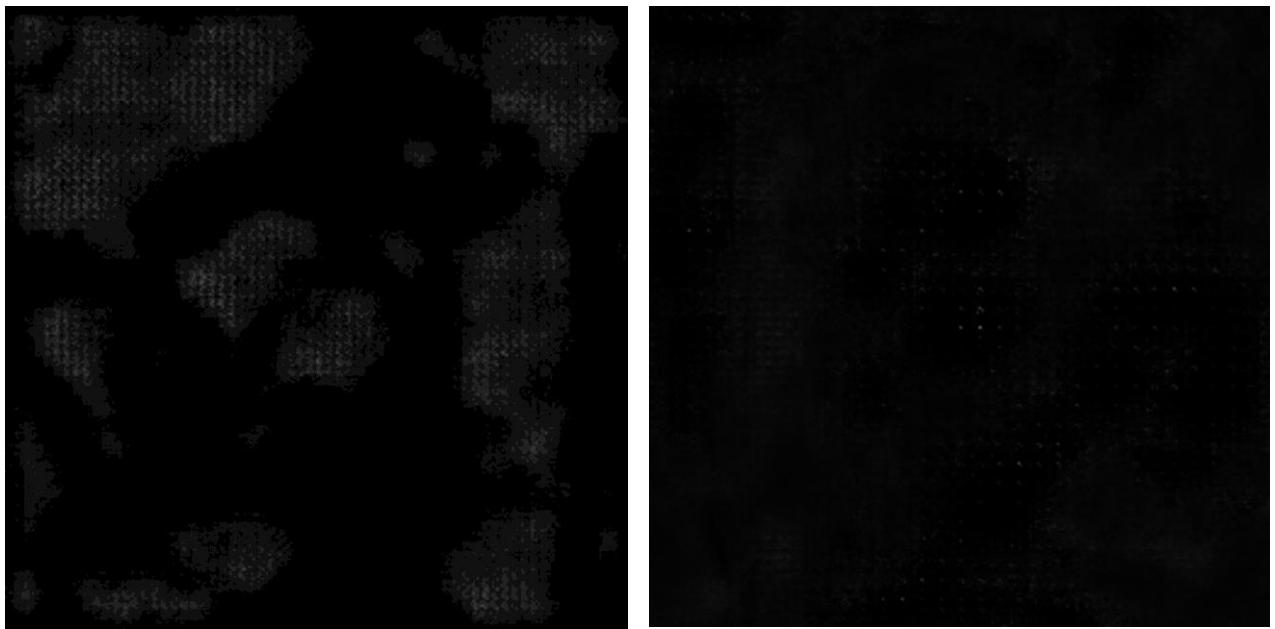
**11 lentelė.** Apmokyto neuroninio tinklo išvestys pavyzdinėms nuotraukoms. Pirmoje eilutėje pateikiamos dekoduotos išvestys, antroje- atitinkamoms pirmos eilutės nuotraukoms apskaičiuotos karščio žemėlapio reikšmės, trečioje- vizualizuotos spindulių reikšmės bei ketvirtroje- vizualizuotos 4 pikselių paklaidų reikšmės. Naudojamas pavyzdinės nuotraukos iš *Google Earth* programos.





Stebint šias reikšmės galima matyti modelio apsimokymo stabilumą (ar reikšmės yra geros raškos bei karščio žemėlapio reikšmės vientisos), tikslumą, stebint karščio žemėlapio ir pavyzdinės nuotraukos atitikimą, modelio užtikrintumą įvairiais atvejais bei kokiais atvejais modelis daro klaidas. Šiuo metu taip pat ieškoma klaidų išvesčių dekodavime, ieškant nesutapimų tarp dekoduotų ir nedekoduotų išvesčių. Lentelėje vaizduojamos neapdorotos išvestys vaizduoja teisingą ir kokybišką apsimokymą. Trečioje ir ketvirtoje eilutėje esančios spindulių ir pikselių paklaidos prognozės yra nestabilios tose vietose, kuriose nėra medžių, kadangi tose teritorijose apmokymo metu *loss* paklaidos reikmė nėra skaičiuojama. Priešingai nei viršuje pateikta lentelė, lentelė apačioje vaizduoja netinkamą modelio karščio žemėlapio apsimokymą, kuris indikuoja programavimo klaidą:

**12 lentelė.** Apmokyto neuroninio tinklo klaidingos išvestys. Kairėje- nestabilios ir netolygios išvestys, dėl kurių negalima tiksliai prognozuoti medžio viršūnės centro. Dešinėje- atsitiktiniai maži, tačiau aukštas reikšmės turintys taškai, kurie dekoduojami grąžina netikslias (*false positive*) medžių viršūnių reikšmes.

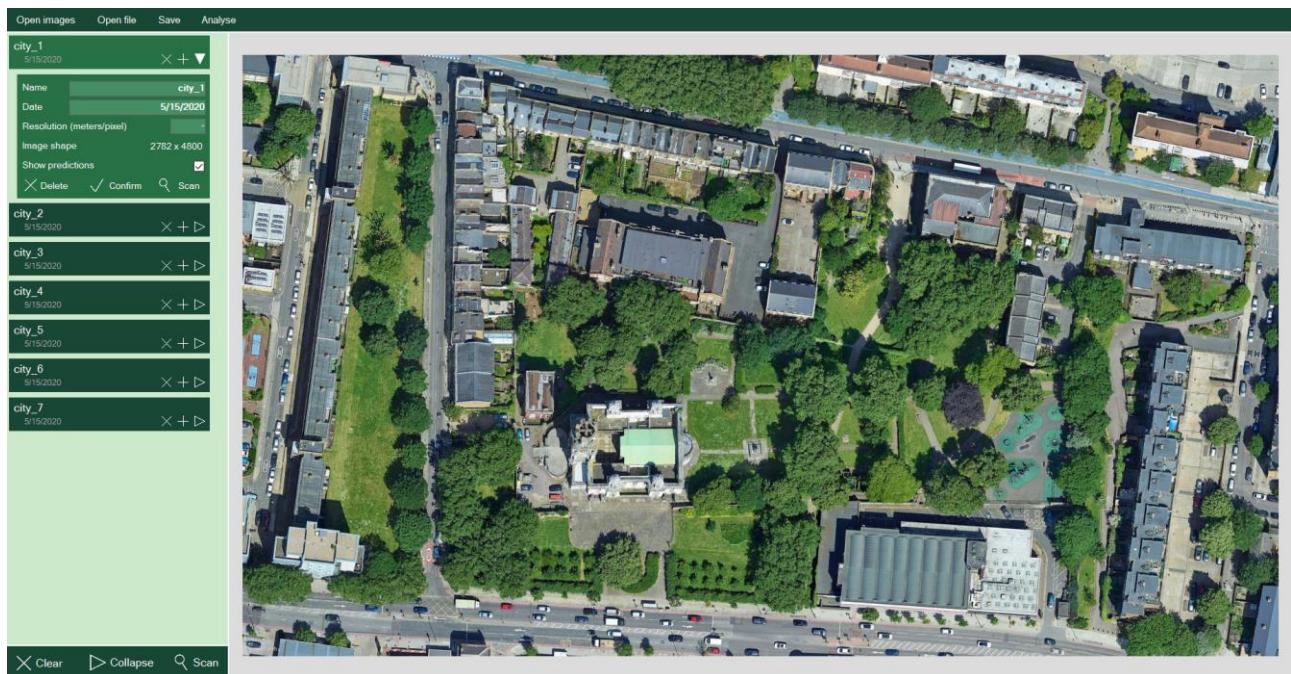


Atliekant modelio įvesčių ir išvesčių vizualizacijų analizę buvo rasta daug programavimo klaidų bei modelio ir jo apmokymo klaidų. Šis procesas padėjo geriau suprasti modelio silpnąsias vietas ir jį tinkamiau apmokyti.

### 3.5. Vartotojo sąsajos testavimas

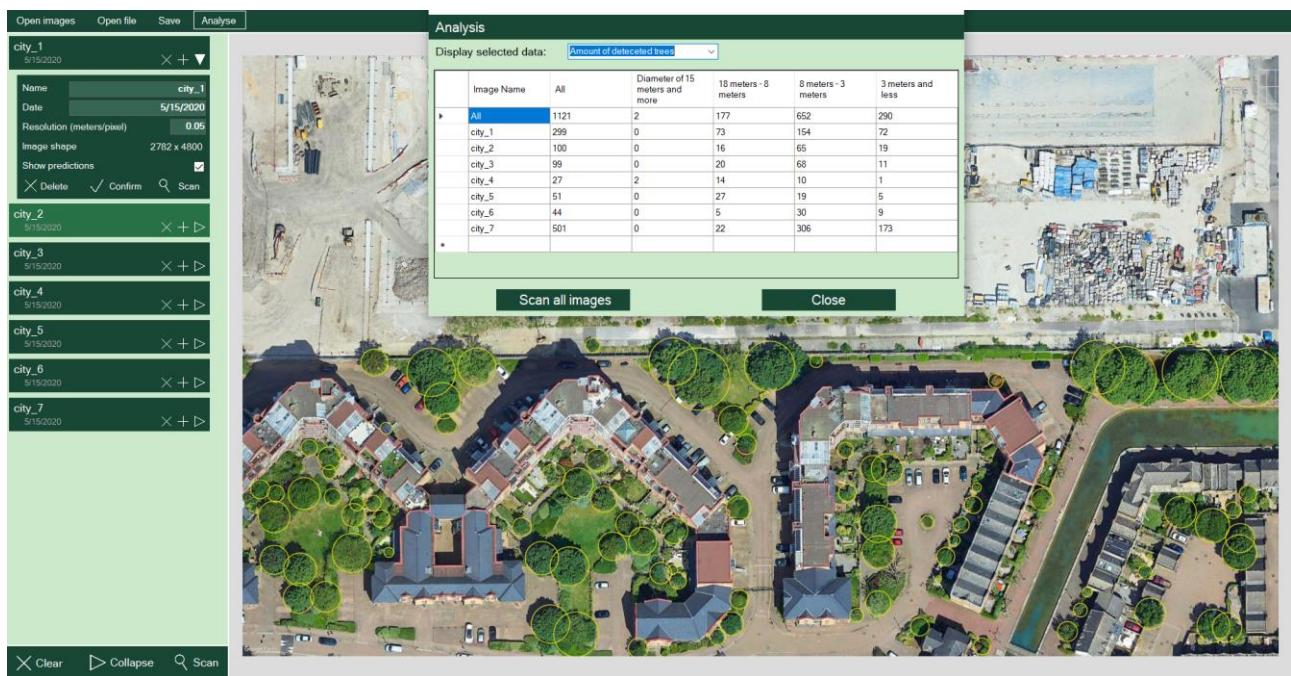
Šioje testavimo dalyje testuojama vartotojo sąsaja. Rankiniu būdu įvertinamas visos vartotojo sąsajos veikimas, įvairiomis sąlygomis atliekamos įvairios jos funkcijos siekiant rasti klaidas bei tyčia jas sukelti. Taip pat ieškomos kitos vartotojo sąsajos problemos, tokios kaip netinkamas ar nepatogus elementų ar duomenų vaizdavimas, nepatogus sąsajos veikimas bei tikrinant vartotojo sąsajos funkcijas bei stebint kompiuterinės regos algoritmo programos veikimą įvertinamas abiejų programų tarpusavio veikimas ir komunikavimas.

Pirmiausiai testavimas pradedamas nuo mygtukų ir jų funkcijų atlikimo testavimo. Tikrinama, ar programa elgiasi taip, kaip tikimasi, ar funkcijos yra patogios ir tinkamai įvykdamos, ar sąsaja įvykdo užsibrėžtus reikalavimus. Išbandomos visų mygtukų paspaudimų funkcijos kairėje esančiame nuotraukų aprašų skirtuke, žemiau esančioje funkcijų juosteje, viršuje esančioje antraštės juosteje bei papildomuose languose esančios funkcijos. Funkcijos išbandomos su įvairiu nuotraukų kiekiu, parametrais. Įvertinamas funkcijų veikimas prieš komunikaciją su kompiuterinės regos algoritmo programa, po jos ir komunikavimo metu, kai išsiusta HTTP POST užklausa ir laukiama apdorojimo rezultatų.



**3.3 pav.** Vartotojo sąsajos vizualizacija. Kairėje- nuotraukų aprašų skirtukas, kuriame galima pasirinkti nuotraukas ir keisti nuotraukos parametrus. Kairėje apačioje- pagrindinių skirtuko funkcijų juosta. Viršuje- svarbiausių funkcijų antraštės juosta. Naudojama pavyzdinė palydovo atlakta nuotrauka iš *Google Earth* programos.

Po mygtukų funkcijų testavimo atliekama pateikiamų duomenų analizė. Tikrinama, ar gauti apdorojimo duomenys tinkamai vaizduojami, ar duomenys tikslūs įvairiems nuotraukų dydžiams, ar pateikiamų nuotraukų analizės duomenys yra vaizduojami tinkamai ir atitinka vizualizuotus duomenis. Tikrinamos vartotojo sąsajos analizės funkcijos- nuotraukų pokyčio analizė laike bei jos vaizdavimas, kuri vaizduojama nuotraukos vaizdavimo skyriuje kartu su paprastomis medžių viršunių prognozėmis pateikiant medžių viršunes, bei statistinė visų rezultatų analizė, vaizduojama atskirame lange (pavyzdys 3.4 pav.). Jei reikalinga, testavimui naudojama *Visual Studio* programavimo aplinkos *Debug* funkcija programai sustabdyti ir papildomoms skaitinėms vertėms gauti, siekiant patikrinti, ar jos teisingai vaizduojamos programoje.



**3.4 pav.** Vartotojo sąsajos vizualizacija. Nuotraukoje vaizduojamas analizės langas bei kompiuterinės regos algoritmo teikiami rezultatai konkrečiai miesto nuotraukai iš viršaus. Naudojama pavyzdinė palydovo atlirkta nuotrauka iš Google Earth programos.

#### 4. Neuroninių tinklų apmokymo rezultatai

Atliekant šį darbą buvo eksperimentuojama su MobileNetV2, MobileNetV3, EfficientNet bei ResNet tipo neuroninio tinklo architektūromis. Naudojant šiuos neuroninius tinklus kaip stuburinius tinklus *Feature Pyramid Network* tinkle jie buvo pilnai apmokomi, išbandomi, stebimos jų daromos klaidos. Pagal tai sukurtas naujas neuroninis tinklas medžių viršūnių aptikimo užduočiai.

Visi tinklai, išskaitant ir šiame darbe naują sukurtą tinklą, buvo apmokyti su tuo pačiu duomenų rinkiniu ir testuoti tam pačiam testavimo duomenų rinkiniui. Testavimo rinkinys sudarytas iš 28 1024x1024 dydžio nuotraukų. Didžiąją dalį nuotraukų sudaro miestų bei parkų nuotraukos. Visi tinklai buvo apmokyti su 1024x1024 dydžio įvestimi išskyurus EfficientNet, kuris buvo apmokomas su 512x512 dydžio įvestimi. Visi tinklai apmokomi su duomenų partijos dydžiu (*batch size*) lygiu 1 bei *Adam* optimizatoriumi pradedant su 0.001 apmokymo koeficientu. Žemiau pateiktose lentelėse aprašomi neuroninių tinklų testavimo rezultatai tam pačiam duomenų rinkiniui. Šiame darbe medžių aptikimo užduočiai sukurtas neuroninis tinklas lentelėje pavadinamas *CustomNet*. Skaičiuojant Recall ir Precision reikšmes, prognozuojama reikšmė užskaitoma kaip teisinga, jei turi bent 0.5 IoU persidengimą su bet kuria reikšme testuojamoje nuotraukoje.

**13 lentelė.** Apmokyti neuroninių tinklų medžių viršūnių aptikimo tikslumo rezultatai visų dydžių medžiams.

	MobileNetV2	MobileNetV3	EfficientNet-B0	ResNet50	CustomNet
Precision	0.621	0.352	0.429	0.490	0.666
Recall	0.523	0.672	0.126	0.39	0.642

**14 lentelė.** Apmokyti neuroninių tinklų medžių viršūnių aptikimo greičio rezultatai naudojant Nvidia RTX 2060 vaizdo plokštę. Laikas skaičiuojamas sekundėmis vienai nuotraukai apdoroti nuo įvesties pateikimo neuroniniams tinklui iki dekoduotų išvesčių gavimo.

	MobileNetV2	MobileNetV3	EfficientNet-B0	ResNet50	CustomNet
Vidurkis	0.042	0.044	0.044	0.069	0.036
Mediana	0.043	0.045	0.045	0.070	0.038
Standartinis nuokrypis	0.00324	0.00274	0.00309	0.00085	0.00314

**15 lentelė.** Neuroninių tinklų parametru kiekis su *FPN* tinklo dalimi.

	MobileNetV2	MobileNetV3	EfficientNet-B0	ResNet50	CustomNet
Parametru skaičius	2921476	2735260	3627872	24175524	2157668

Iš visų tinklų, su kuriais buvo eksperimentuota modelio kūrimo pradžioje prieš kuriant naują konvoluciinį neuroninį tinklą, geriausi rezultatai gaunami naudojant MobileNetV2 tinklą. Būtent šio tinklo savybių daugiausiai yra vėlesnėje fazėje kurtame neuroniniame tinkle. Naudojant kitus tinklus nebuvvo gautas pakankamai geras tikslumas. EfficientNet tinklas pateikė prasčiausius rezultatus. ResNet tinklas turėjo per didelį parametru kiekį šiai užduočiai, o mažesnės šio tinklo versijos neturėjo pakankamai sluoksnių, kad pateikiama informacija būtų apdorojama pakankamai gerai. Dėl per-

didelio parametru kieko tinklas pasiekė persimokymo būseną, tai buvo vienintelis tinklas, kurio apmokymą reikėjo sustabdyti anksčiau. Visų neuroninių tinklų greitis, nepaisant parametru ir konvoliucijų tipo, buvo panašus, MobileNetV2 tinklui būnant greičiausiu.

Pagrindiniai šiame darbe kurto neuroninio tinklo patobulinimai, lyginant su MobileNetV2 tinklu:

- Patobulinta „bottleneck“ bloko struktūra;
- Dvi atskiros *Depthwise* konvoliucijos šakos „bottleneck“ blokams, kurie sumažina išvesties ilgį ir plotį 2 kartus, kurių gautos išvestys yra sudedamos naudojantis *Add* matricų operacija;
- 5x5 dydžio filtrų *Depthwise* konvoliucijose naudojimas tinkamose vietose;
- Patobulinta *FPN* struktūra, pridedant konvoliucijos su 1x1 dydžio filtru bloką prieš dekonvoliucijos bloką bei papildoma tarpinė tilto išvestis iš stuburinio tinklo, kuri sumuojama su kita tarpine išvestimi;
- Tinkamesnis „bottleneck“ blokų išdėstymas, perkeliant didesnį kiekį parametru į stuburinio tinklo vidurį.

Nepaisant tobulinimo ir papildomo duomenų žymėjimo, visi neuroniniai tinklai nepateikė itin gerų rezultatų tankiai medžiais apaugusiose teritorijose ir geriausius rezultatus pateikia miestų ar nemiškingose užmiesčio palydovų nuotraukose.

## **5. Dokumentacija naudotojui**

### **5.1. Apibendrintas sistemos galimybių aprašymas**

Sistema suteikia galimybę vartotojui pateikti norimas palydovo ar bepiločio orlaivio darytas nuotraukas ir naudojantis kompiuterinės regos algoritmu aptiktį jose esančias medžių viršūnes. Iš pateiktų rezultatų vartotojas gali gauti paprastą statistinę informaciją apie nuotraukose rastus medžius bei stebėti, kuriose vietose medžiai potencialiai buvo nukirsti. Kompiuterinės regos algoritmo programa patyrusiam vartotojui taip pat suteikia galimybę naudotis algoritmu tiesiogiai be vartotojo sasajos kaip tarpininko bei pakartotinai apmokyti modelį su savo duomenų rinkiniu.

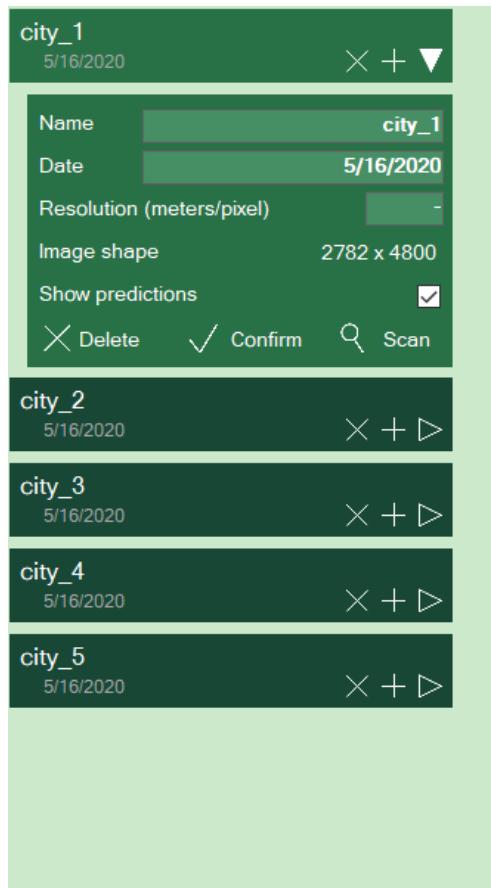
### **5.2. Vartotojo vadovas**

Pradedant naudoti vartotojo sasajos programą atsidariusiame lange kairėje esanti nuotraukų stulpelio skiltis bus tuščia. Pirma užduotis vartotojui - atidaryti nuotraukos failą ar failus. Tai galima padaryti atidarant nuotraukos failus arba praeitoje naudojimo sesijoje išsaugotus rezultatus. Norint atidaryti nuotraukų failus viršuje esančioje antraštės juostoje (5.1 pav.) spaudžiamas „Open images“ mygtukas.



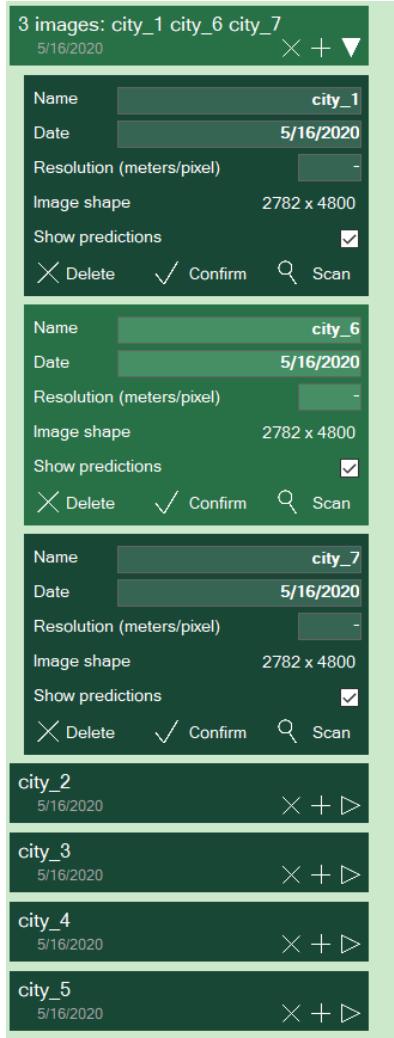
**5.1 pav.** Vartotojo sasajos antraštės mygtukų juosta su svarbiausiomis funkcijomis.

Paspaudus „Open images“ mygtuką atsidaro failų pasirinkimo dialogo langas. Pasirenkami norimi paveikslėliai „.jpg“ arba „.png“ formatais. Pasirinkti paveikslėliai užkraunami ir vaizduojami jų aprašai bei parametrai kairėje esančiame nuotraukų stulpelyje (5.3 pav.). Paspaudus „Open file“ funkciją atsidaromas failų pasirinkimo dialogas, kuriame galima pasirinkti „.json“ tipo failą ir taip užkrauti praeitą išsaugotą naudojimo sesiją su nuotraukomis, jų parametrais bei nuotraukų apdorojimo rezultatais. Tokiu būdu užkrautos nuotraukos taip pat yra vaizduojamos kairėje esančiame nuotraukų aprašų stulpelyje. „Save“ mygtukas leidžia saugoti dabartinę naudojimo sesiją „.json“ formatu.

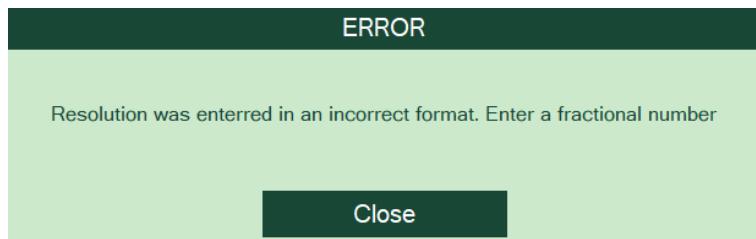


**5.2 pav.** Vartotojo sąsajos nuotraukų aprašų stulpelis.

Naudojantis 5.2 pav. vaizduojamu nuotraukų aprašų stulpeliu galima peržiūrėti ir keisti nuotraukų parametrus, pasirinkti nuotraukas peržiūrai dešinėje esančiame nuotraukų peržiūros plote, trinti nuotraukas ir kt. Suskleistoje nuotraukų aprašo skiltyje paspaudus „>“ mygtuką išsiskleidžia platesnis nuotraukos aprašas. Jame galima keisti nuotraukos pavadinimą, datą, rezoliuciją, matyti nuotraukos dydį, išjungti nuotraukos vaizdavimą. Taip pat galima trinti nuotrauką bei individualiai ją skanuoti. Iš šią skiltį vartotojas gali papildomai pridėti nuotrauką, kurios buvo nufotografuotos toje pačioje teritorijoje, tačiau kitu laiku. Tai galima atlikti paspaudus „+“ mygtuką prie nuotraukos skilties, atsidarius failų pasirinkimo dialogui pasirinkti „.jpg“ arba „.png“ formato failą. Iš vieną platesnę skiltį sudėtos nuotraukos (5.3 pav.) bus analizuojamos tarpusavyje laike nusprendžiant, kur potencialiai buvo nukirsti medžiai tarp kelių skirtingų nuotraukų. Iš atskiras skiltis sudėtos nuotraukos bus analizuojamos kaip atskirų teritorijų nuotraukos, nebus analizuojamos tarpusavyje. Paspaudus „x“ arba „X Delete“ mygtukus prie nuotraukų arba jų skilčių jas galima šalinti iš nuotraukų aprašų stulpelio. Spaudžiant „Confirm“ mygtuką galima patvirtinti įvestus duomenis, netinkamai įvestiems duomenims bus iškviesta klaidos žinutė su klaidos aprašu (5.4 pav.). Spaudžiant baltą žemyn rodančią rodyklę galima suskleisti nuotrauką skiltį. Paspaudus ant išskleisto nuotraukų stulpelio pasirinkta nuotrauka bus vaizduojama dešinėje esančiame nuotraukų peržiūros plote. Paspaudus ant suskleistos skilties bus vaizduojama pirma nuotrauka, jei skiltyje jų yra daugiau nei viena.



**5.3 pav.** Vartotojo sąsajos nuotraukų aprašų stulpelis su keliomis nuotraukomis vienoje nuotraukų skiltyje.



**5.4 pav.** Vartotojo sąsajos klaidos pranešimo pavyzdys.

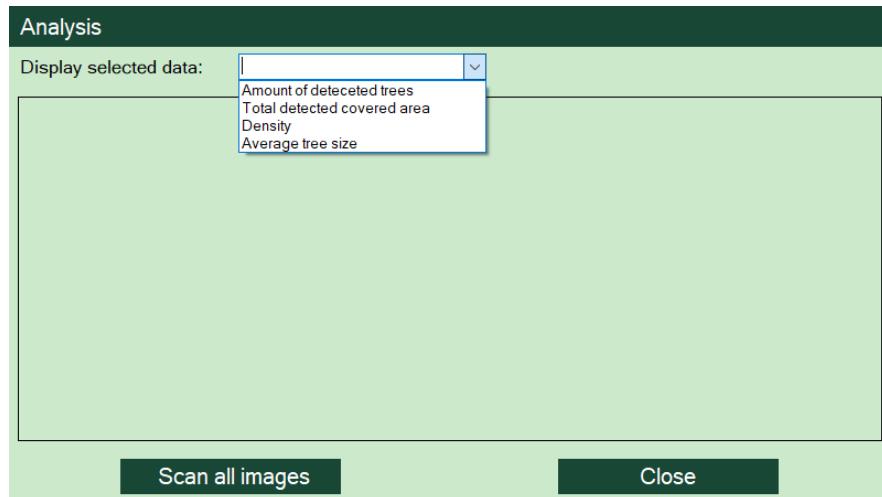
Norint pašalinti visas užkrautas nuotraukas, uždaryti visas išskleistas skiltis bei su kompiuterinės regos algoritmu apdoroti užkrautas nuotraukas galima naudotis apačioje kairėje pusėje esančiu mygtukų rinkiniu (5.5 pav.).



**5.5 pav.** Vartotojo sąsajos pagrindinio lango apačioje esantis mygtukų rinkinys.

Paspaudus „Scan“ mygtuką nuotraukos skiltyje arba apačioje esančiame mygtukų rinkinyje (3.9 pav.) išsiunčiama užklausa apdoroti nuotrauką arba visas nuotraukas kompiuterinės regos algoritmui ir po apdorojimo gaunami rezultatai. Apdorojimo rezultatai yra vaizduojami pasirinktoje nuotraukoje.

Norint gauti statistinius rezultatų duomenis- spaudžiamas „Analyse“ mygtukas antraštės mygtukų juostoje (5.1 pav.). Paspaudus mygtuką pateikiamas analizės langas, kuriame galima pasirinkti, kokio tipo duomenis norima matyti. Šiame lange vaizduojami duomenys tik naujausio pagal datą nuskanuoto paveikslėlio kiekvienoje skiltyje ir tik toms nuotraukoms, kurioms įvesta nuotraukos rezoliucija, pagal kurią galima nustatyti nuotraukos objektų realius dydžius.



**5.6 pav.** Vartotojo sąsajos analizės langas.

5.6 pav. pateiktame analizės lange pasirenkami duomenys, kuriuos norima matyti. Pasirenkamų duomenų lentelė vaizduojama žemiau tame pačiame lange.

### 5.3. Diegimo vadovas

Norint paleisti vartotojo sąsajos aplikaciją papildomas diegimas nereikalingas, aplikacija paleidžiama paleidžiant „TreeDetectionApplication.exe“ failą.

Norint atlikti skaičiavimus su kompiuterinės regos algoritmu reikalinga paleisti paties algoritmo *Python* kalbos aplinkos kodą, kuris sukurs *Flask* aplikaciją ir priims POST užklausą.

### 5.4. Administravimo vadovas

Norint paleisti kompiuterinės regos algoritmą užklausų priėmimo režimu reikalingi šie įrankiai:

- *Python 3.7* programavimo kalba;
- *TensorFlow* karkaso 1.14 versija;
- *NumPy* bibliotekos 1.18.1 versija;
- *Flask* bibliotekos 1.1.2 versija;
- *OpenCV-python* bibliotekos 4.2.0.32;
- Visos kitos bibliotekos, reikalingos aukščiau išvardintoms bibliotekoms.

Šias bibliotekas paprasta instaliuoti per *pip* įrankį.

Paleisti algoritmą galima su naudojantis konsole su šia komanda esant „main.py“ failo direktoriuje:

```
python3 main.py
--mode
Run
--model-name
CustomNetFPN
--gpu
0
--snapshot
..\checkpoints\CustomNetFPN_trained.h5
--input-size
1024
```

Šiame komandų rinkinyje pirmuoju parametru „python3 main.py“ nurodomas python failas, kuris bus iškviečiamas. „main.py“ failas apdoroja užklausą ir paleidžia algoritmą atitinkamu režimu, kuris pasirenkamas „--mode“ parametru („Run“ paleidžia programą užklausų priėmimo režimu). Parametru „--model-name“ nurodomas modelio, kurį norima paleisti, pavadinimas. „--gpu“ parametras leidžia pasirinkti, ar skaičiavimams naudoti kompiuterio vaizdo plokštę (parametras 0), ar procesorių (parametras -1). „--snapshot“ parametru nurodoma failo vieta failų sistemoje, kur yra pasirinkto modelio apmokymo parametrai, kurie bus užkrauti. „--input-size“ parametru nurodomas didžiausias galimas nuotraukos ilgio ir pločio dydis. Jei nuotrauka yra didesnė už šį dydį- ji bus karpoma į smulkesnes ir paduodama modeliui atskirai. Patariama naudoti 1024 pikselių dydį. Kompiuterinės regos algoritmo aplikacija turi būti paleista visą laiką, kol naudojama aplikacija, kitu būdu algoritmas nebus pasiekiamas.

## **Rezultatai ir išvados**

Atliekant šį darbą prieita prie šių išvadų:

1. Atlikta rinkoje ir akademinėje aplinkoje pateikiamų panašių sprendimų analizė parodė, kad šiuo metu rinkoje nėra tinkamai veikiančio sprendimo medžių viršūnių aptikimo problemai spręsti, nepaisant rinkos poreikio ir potencialios komercinės ir visuomeninės naudos sėkmingai pritaikius gerus rezultatus pateikiantį tokio tipo algoritmą.
2. Medžių viršūnių aptikimo problema buvo apibrėžta kaip objektų aptikimo problema ir atlikus rinkoje pateikiamų objektų aptikimo algoritmų analizę pasirinktas CenterNet algoritmas tolimesniams tobulinimui.
3. Siekiant įvertinti tinkamiausiai veikiančią neuroninio tinklo struktūrą ir kuo platesnį spektrą konvoliucinių neuroninių tinklų savybių eksperimentavimui pasirinktos MobileNetV2, MobileNetV3, EfficientNet bei ResNet architektūros. Įvertinus eksperimentavimo rezultatus nutarta, kad MobileNetV2 tinklas yra geriausias šio tipo užduočiai, jis buvo pasirinktas tolimesniams tobulinimui.
4. Testuojant šiame darbe kurtą algoritmą pavyko aptikti klaidas, kurios sumažino algoritmo apmokymo tikslumą, taip pagerinant galutinį algoritmo rezultatą. Atliekant komponentų testavimą buvo aptiktos klaidos testavimo algoritmuose, kurios yra kritiškai svarbios tokio tipo algoritmo kūrimui. Atliekant vartotojo sąsajos rankinį testavimą aptiktos klaidos su nuotraukų analizės bei rezultatų vizualizavimo funkcijomis.
5. Šio darbo metu sukurtas konvoliucinis neuroninis tinklas pateikė geresnius rezultatus nei MobileNetV2 tinklas, kuris iš eksperimentavimui pateiktų architektūrų pateikė aukščiausius rezultatus. Sukurtas neuroninis tinklas geriausius rezultatus pateikia netankiai medžiais apaugusių teritorijų palydovų darytoms nuotraukoms.
6. Sukurta vartotojo sąsaja leidžia lengvai apdoroti pasirinktas nuotraukas, patogiai vizualizuoti rezultatus bei vartotojo nereikalauja turėti kompiuterinės regos žinių. Ja naudojantis gauti paprastą analitinę informaciją apie medžius miškuose ar miestuose.
7. Sukurtas medžių viršūnių aptikimo algoritmas parodė, kad giliuoju mokymu gristas algoritmas šiai problemai spręsti gali pasiekti aukštus tikslumo rezultatus net su itin nedideliu duomenų rinkiniu.

Šiuo metu sistema prototipo būsenoje, ji yra skirta įvertinti tokios pilnai įgyvendintos sistemos galimybėms ir šiuo metu yra netinkama komerciniam naudojimui. Algoritmui patobulinti siekiama sužymėti didesnį kiekį įvairių tipų duomenų bei apmokyti daug didesnį kiekį parametrų turinčius konvoliucinius neuroninius tinklus. Vartotojo sąsajai nebuvo spėta įgyvendinti duomenų bazės sprendimo, kuris leistų duomenis vartotojui programoje užkrauti iš debesyse esančios aplinkos pagal pasirinktas planetos koordinates.

## Literatūros sąrašas

1. Xiao, Changlin, Qin, Rongjun ir Huang, Xu, Individual Tree Detection from Multi-View Satellite Images. IEEE International Symposium Geoscience and Remote Sensing (IGARSS). 2018. [Cituota 2020 m. gegužės 8 d.]. [https://www.researchgate.net/publication/325256619\\_Individual\\_Tree\\_Detection\\_from\\_Multi-View\\_Satellite\\_Images](https://www.researchgate.net/publication/325256619_Individual_Tree_Detection_from_Multi-View_Satellite_Images)
2. Mohan, Midhun et al., Individual Tree Detection from Unmanned Aerial Vehicle (UAV) Derived Canopy Height Model in an Open Canopy Mixed Conifer Forest. 2017. [Cituota 2020 m. gegužės 8 d.]. [https://www.fs.fed.us/rm/pubs\\_journals/2017/rmrs\\_2017\\_mohan\\_m001.pdf](https://www.fs.fed.us/rm/pubs_journals/2017/rmrs_2017_mohan_m001.pdf)
3. Gomes, Marilia Ferreira ir Maillard, Philippe, 2016, Detection of Tree Crowns in Very High Spatial Resolution Images. 2016. [Cituota 2020 m. gegužės 8 d.]. <https://www.intechopen.com/books/environmental-applications-of-remote-sensing/detection-of-tree-crowns-in-very-high-spatial-resolution-images>
4. Safanova, Anastasiia et al., Detection of Fir Trees (*Abies sibirica*) Damaged by the Bark Beetle in Unmanned Aerial Vehicle Images with Deep Learning. 2019. [Cituota 2020 m. gegužės 8 d.]. <https://www.mdpi.com/2072-4292/11/6/643>
5. Li, Weijia, Dong, Runmin, Fu, Haohuan ir Yu, Le, Large-Scale Oil Palm Tree Detection from High-Resolution Satellite Images Using Two-Stage Convolutional Neural Networks. 2018. [Cituota 2020 m. gegužės 8 d.]. <https://www.mdpi.com/2072-4292/11/1/11>
6. Kaggle iššūkis, Planet: Understanding the Amazon from Space, 2017. <https://www.kaggle.com/c/planet-understanding-the-amazon-from-space>.
7. Drivendata iššūkis, Open Cities AI Challenge: Segmenting Buildings for Disaster Resilience, 2020. [Cituota 2020 m. gegužės 8 d.]. <https://www.drivendata.org/competitions/60/building-segmentation-disaster-resilience/page/150/>
8. Weinstein, Ben G. et al., Individual Tree-Crown Detection in RGB Imagery Using Semi-Supervised Deep Learning Neural Networks. 2019. [Cituota 2020 m. gegužės 8 d.]. <https://www.mdpi.com/2072-4292/11/11/1309>
9. Produktas, Katam, Katam TreeMap, [Cituota 2020 m. gegužės 8 d.]. <https://www.katam.se/products/treemap/>
10. Papers with code, Object Detection on COCO test-dev, [Cituota 2020 m. gegužės 8 d.]. <https://paperswithcode.com/sota/object-detection-on-coco>
11. Zhou, Xingyi, Wang, Dequan ir Krahenb, Philipp, Objects as Points. 2019. <https://arxiv.org/pdf/1904.07850.pdf>
12. Ren, Shaoqing, He, Kaiming, Girshick, Ross ir Sun, Jian, 2015, Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2015. <https://arxiv.org/pdf/1506.01497.pdf>
13. Xu, Yuanyuan, CenterFace: Joint Face Detection and Alignment Using Face as Point. 2019. <https://arxiv.org/ftp/arxiv/papers/1911/1911.03599.pdf>

14. Vartotojas „xuannianz“, „keras-CenterNet“, Github. [Cituota 2020 m. gegužės 8 d]. <https://github.com/xuannianz/keras-CenterNet>
15. Sandler, Mark et al., MobileNetV2: Inverted Residuals and Linear Bottlenecks. IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2018. <https://arxiv.org/abs/1801.04381>
16. Howard, Andrew et al., Searching for MobileNetV3. 2019. <https://arxiv.org/abs/1905.02244>
17. Tan, Mingxing ir Le, Quoc V., EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. 2019. <https://arxiv.org/abs/1905.11946>
18. He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing ir Sun, Jian, Deep Residual Learning for Image Recognition. 2015. <https://arxiv.org/abs/1512.03385>
19. Vartotojas „xiaochus“, „MobileNetV2“, Github. [Cituota 2020 m. gegužės 8 d]. <https://github.com/xiaochus/MobileNetV2>
20. Vartotojas „godofpdog“, „MobileNetV3\_keras“, Github. [Cituota 2020 m. gegužės 8 d]. [https://github.com/godofpdog/MobileNetV3\\_keras](https://github.com/godofpdog/MobileNetV3_keras)
21. Vartotojas „qubvel“, „efficientnet“, Github. [Cituota 2020 m. gegužės 8 d]. <https://github.com/qubvel/efficientnet>
22. Bengio, Yoshua, Frasconi, Paolo ir Schmidhuber, Jurgen, Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. 2001. <https://www.bioinf.jku.at/publications/older/ch7.pdf>
23. Zoph, Barret ir Le, Quoc V., Neural Architecture Search With Reinforcement Learning. 2017. <https://arxiv.org/pdf/1611.01578.pdf>
24. Chollet, Francois, Xception: Deep Learning with Depthwise Separable Convolutions. 2016. <https://arxiv.org/abs/1610.02357>
25. Hu, Jie , Squeeze-and-Excitation Networks. 2017. [Cituota 2020 m. gegužės 17 d]. <https://arxiv.org/pdf/1709.01507.pdf>
26. Vartotojas „HarisIqbal88“, „PlotNeuralNet“, Github. [Cituota 2020 m. gegužės 17 d]. <https://github.com/HarisIqbal88/PlotNeuralNet>
27. Lin, Tsung-Yi, Feature Pyramid Networks for Object Detection. 2016. <https://arxiv.org/abs/1612.03144>
28. Ioffe, Sergey ir Szegedy, Christian, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. <https://arxiv.org/abs/1502.03167>
29. Noh, Hyenwoo, Hong, Seunghoon ir Han, Bohyung, Learning Deconvolution Network for Semantic Segmentation. 2015. [https://www.cv-foundation.org/openaccess/content\\_iccv\\_2015/papers/Noh\\_Learning\\_Deconvolution\\_Network\\_ICCV\\_2015\\_paper.pdf](https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Noh_Learning_Deconvolution_Network_ICCV_2015_paper.pdf)

30. Kingma, Diederik P. ir Ba, Jimmy, Adam: A Method for Stochastic Optimization. 2014.  
<https://arxiv.org/abs/1412.6980>