



VILNIAUS GEDIMINO
TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

Artūras JONKUS

OBJEKTŲ SEKIMO VAIZDE ALGORITMŲ SPARTINIMAS
ACCELERATION OF OBJECT TRACKING ALGORITHMS

Magistro baigiamasis darbas

Elektronikos studijų programa, valstybinis kodas 621H61003

Kompiuterizuotų elektroninių sistemų specializacija

Elektronikos inžinerijos studijų kryptis

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
ELEKTRONIKOS FAKULTETAS
ELEKTRONINIŲ SISTEMŲ KATEDRA

TVIRTINU
Katedros vedėjas

(parašas)
Prof. dr. D. Navakauskas
2018 m. _____ mėn. ____ d.

Artūras JONKUS

OBJEKTŲ SEKIMO VAIZDE ALGORITMŲ SPARTINIMAS
ACCELERATION OF OBJECT TRACKING ALGORITHMS

Magistro baigiamasis darbas

Elektronikos studijų programa, valstybinis kodas 621H61003

Kompiuterizuotų elektroninių sistemų specializacija

Elektronikos inžinerijos studijų kryptis

Vadovas

prof. dr. Artūras Serackis

(pedag. vardas, moksl. laipsnis, vardas, pavardė) _____ (parašas) (data)

Lietuvių kalbos konsultantas

lekt. dr. Anželika Gaidienė

(pedag. vardas, moksl. laipsnis, vardas, pavardė) _____ (parašas) (data)

Vilniaus Gedimino technikos universitetas Elektronikos fakultetas Elektroninių sistemų katedra	ISBN ISSN Egz. sk. Data*.....*.....
Antrosios pakopos studijų Elektronikos inžinerijos programos magistro baigiamasis darbas Pavadinimas Objektų sekimo vaizde algoritmų spartinimas Autorius Artūras Jonkus Vadovas Artūras Serackis	
Kalba: lietuvių	
Anotacija Objektų sekimo vaizde algoritmų spartinimas. Magistro baigiamasis darbas elektronikos inžinerijos laipsniui. Vilniaus Gedimino technikos universitetas. Vilnius, 2018, 61 p., 25 iliustr., 7 lent., 37 bibl., 2 priedai. Naujausi objektų sekimo vaizde algoritmai veikia vis tiksliau, tačiau tik nedidelė dalis jų gali veikti realiu laiku. Baigiamojo darbo tikslas - padidinti objektų sekimo algoritmo spartą. Tikslui pasiekti bandyta keisti objektų aptikimo dažnumą bei objektų aptikimo algoritmo architektūrą. Tyrimo objektu pasirinkti pėstieji. Iškelti uždaviniai yra įvykdyti - atlikti eksperimentinius tyrimus ir esamo algoritmo derinimą gauta sistema, galinti sekti pėsčiuosius vaizde realiu laiku (panaudojant GeForce GTX1080Ti grafinį procesorinį įrenginį).	
Prasminiai žodžiai: Pėstieji, MOTA, MOTP, objektų aptikimas	

Vilnius Gediminas Technical University Faculty of Electronics Department of Electronic Systems		ISBN ISSN Copies No. Date*.....*.....	
Master Degree Studies Electronics Engineering study programme Master Graduation Thesis Title Acceleration of the Object Tracking Algorithms Author Artūras Jonkus Academic supervisor Artūras Serackis			
Thesis language: Lithuanian			
Annotation Newest visual object tracking algorithms are getting better every year but only a small fraction of them can work at real time. The goal of this work was to increase a speed of object tracking algorithm. Pedestrians were used as an example. All tasks were accomplished - a group of methods were proposed to enable pedestrian tracking on real time (using GeForce GTX1080Ti GPU). Master Thesis consists of 61 pages without annexes, 25 figures, 7 tables, 37 bibl. entries, 2 annexes			
Keywords: Pedestrians, MOTA, MOTP, object detection			

TURINYS

Žymenys ir santrumpos	6
Įvadas	7
Darbo aktualumas ir tikslas	7
Darbo uždaviniai	7
Naudoti tyrimo ir analizės metodai	7
Darbo naujumas ir praktinė nauda	7
Darbo struktūra	8
1. Objekto vaizde koordinačių adaptyvaus atnaujinimo algoritmų Analitinė apžvalga	9
1.1. Kalmano filtro taikymas objektams vaizde sekti	9
1.1.1. Veikimo principas	9
1.1.2. Problemos ir galimi jų sprendimai sekant objektus vaizde	11
1.2. Dalelių filtro taikymas objektams vaizde sekti	13
1.2.1. Veikimo principas	13
1.2.2. Problemos ir galimi jų sprendimai sekant objektus vaizde	14
2. Objekto koordinačių atnaujinimo algoritmų ir juose naudojamų metodų analizė	17
2.1. Konvoliuciniais dirbtinių neuronų tinklais paremti metodai	17
2.2. YOLOv2 objektų aptikimo algoritmo architektūra.....	18
3. Naujo būdo pasirinktam algoritmui paspartinti sudarymas ir eksperimentinio tyrimo duomenų paruošimas	21
3.1. Objektų sekimo vaizde įvertinimo metrikos	21
3.2. Duomenų paruošimas tinklo mokymui	24
3.2.1. Duomenų rinkinio pasirinkimas	24
3.2.2. Pirminis duomenų apdorojimas	26
3.3. Optimizuojama sistema ir jos parametų įvertinimas	27
3.4. Galimybės paspartinti esamą algoritmą	30
4. Eksperimentinio tyrimo planas ir rezultatai	33
4.1. Objektų sekimo vaizde spartinimas naudojant objektų aptikimą tik kai kuriuose kadruose	33
4.2. Objektų sekimo vaizde spartinimas keičiant objektų aptikimo algoritmo architektūrą.....	37
4.2.1. Objektų aptikimo algoritmo spartinimas keičiant tikslinimo stačiakampių parametrus	38
4.2.2. Objektų aptikimo algoritmo spartinimas atsisakant tam tikrų neuronų tinklo sluoksnių.....	42
4.2.3. Objektų aptikimo algoritmo spartinimas mažinant filtrų skaičių konvoliuciniuose sluoksniuose	45
4.2.4. Objekto aptikimo vaizde spartinimas naudojant <i>SqueezeNet Fire</i> elementus.....	46
Apibendrinimas	49
Literatūra	51
PRIEDAI	54
A priedas. Pasiūlyto algoritmo programos išeities kodas	54
B priedas. Pranešimo XXI-oje Lietuvos jaunųjų mokslininkų konferencijoje medžiaga.....	59

ŽYMENYS IR SANTRUMPOS

Žymenys:

- Kodo kintamųjų pavadinimai žymimi naudojant *Italics* stilių;
- Svetimų kalbų žodžiai, paaiškinti santrumpose arba bet kurioje darbo vietoje, žymimi naudojant *Italics* formatavimo stilių (pvz. *recall*); sutrumpinimai nėra išskiriami kokiais nors formatavimo stiliais.

Santrumpos:

- FPS (angl. *Frames Per Second*) – per vieną sekundę apdorotų kadrų skaičius;
- SLAM (angl. *Simultaneous Localization And Mapping*) – algoritmas žemėlapiui brėžti, kai jį brėžiantis objektas tuo pačiu kiekvienu laiko momentu žino savo poziciją jame;
- YOLO (angl. *You Only Look Once*) – konvoliucinis giluminis dirbtinių neuronų tinklas, kurio veikimas pagrįstas viso vaizdo apdorojimu iš karto;
- mAP (angl. *mean Average Precision*) – suvidurkintas aptikimo tikslumas, įvertis, apibūdinantis, kaip tiksliai vaizde aptinkamas norimas objektas;
- IOU (angl. *Intersection Over Union*) – sankirtos ir sąjungos santykis;
- MOTA (angl. *Multiple Object Tracking Accuracy*) – daugelio objektų sekimo taiklumas. Įvertina trijų rūšių klaidas – sekimo identifikatoriaus praradimo, objekto neaptikimo, kur jis turėtų būti rastas, ir objekto aptikimo ten, kur jo nėra;
- MOTP (angl. *Multiple Object Tracking Precision*) – daugelio objektų sekimo tikslumas (angl., MOTP). Jis vertina, kokia vidutiniška objekto vietos nustatymo (lokalizavimo) paklaida.

ĮVADAS

Darbo aktualumas ir tikslas

Vaizdai mūsų gyvenime turi didelę reikšmę. Pamatytas vaizdas analizuojamas smegenyse, o tokią analizę atlieka milijonai neuronų. Vaizde galima rasti daug informacijos, todėl nesunku rasti vis naujų pritaikymo būdų. Viena iš tokių sričių – pėsčiųjų vaizde sekimas. Vis nauji šią problemą sprendžiantys modeliai ir gerėjantys jų algoritmų rezultatai sukuria galimybių spręsti problemas, kurių nebuvo galima išspręsti anksčiau. Vis dar išlieka problema – tiksliai veikiantys algoritmai tebėra lėti. Pageidautina, kad tokie algoritmai būtų kuo spartesni ir galėtų veikti kuo mažiau resursų ir galios vartojančiuose įtaisuose.

Šio darbo tikslas – ištirti objektų sekimo vaizde algoritmų veikimo principus, pateikti ir eksperimentiškai ištirti galimus algoritmų spartinimo būdus.

Darbo uždaviniai

Baigiamojo darbo tikslui pasiekti keliama trys uždaviniai:

1. Atlikti algoritmų, skirtų adaptyviai atnaujinti objekto koordinates vaizdo sraute, veikimo analizę.
2. Išsiaiškinti pasirinktų algoritmų trūkumus jį taikant specializuotai. Pasiūlyti objekto koordinačių atnaujinimo algoritmo spartinimo būdą.
3. Patikrinti naujai sudaryto algoritmo efektyvumą ir palyginti jo veikimą su kitais algoritmais. Pateikti lyginamojo tyrimo rezultatus.

Naudoti tyrimo ir analizės metodai

Šio baigiamojo darbo tyrimas pagrįstas eksperimentais ir esamų algoritmų analize bei jų parametrų derinimu. Eksperimentiniams tyrimams atlikti naudotas kompiuteris su aštuonių branduolių AMD Ryzen 7 1700 3.7 GHz procesoriumi, 32 GB RAM operatyviosios atminties, ASUS ROG Strix GeForce GTX1080Ti 11 GB RAM GAMING grafiniu procesoriniu įrenginiu. Kodas rašytas *Python* programavimo kalba ir OpenCV bibliotekomis, kai kurios kodo dalys rašytos C ir C++ kalba ir naudotos su papildomais kvietiniais siekiant taip gauti didesnę algoritmo veikimo spartą.

Darbo naujumas ir praktinė nauda

Daugelio objektų sekimas vaizde – vis dar pakankamai sparčiai plėtojama sritis, kurios rezultatų tobulėjimui vis dar yra labai daug erdvės. Praeito dešimtmečio algoritmai greiti, tačiau veikia nepakankamai kokybiškai, naujieji veikia daug geriau, tačiau yra lėtesni ir todėl reikalauja

mokslinių tyrimų, siekiančių šiuos algoritmus optimizuoti. Šio baigiamojo darbo rezultatai ir išvados turėtų pateikti būdų, kurie leistų padidinti algoritmo veikimo spartą.

Spartesni sekimo algoritmai, gauti pritaikius pakeitimus, gali būti panaudoti teikti apčiuopiamą praktinę naudą – juos galima pritaikyti skaičiuoti žmones, apsilankiusius prekybos centre ar esančius jame tam tikru momentu, automatiškai identifikuoti vagis parduotuvėje ar įsilaužėlius į įmonės biurą ar sandėlį, taip pat automatinėms prekybos sistemoms (parduotuvėms be kasų ir kasininkų), savaeigėse mašinose kaip pagalbą siekiant išvengti avarijų ir pan.

Darbo struktūra

Darbo pradžioje reikalinga išsami naudojamų algoritmų analizė, jų palyginimas bei pasirinkimo atliekant tyrimus pagrindimas. Analizė atliekama dviejų rūšių algoritmams – objekto koordinatų atnaujinimo (objekto aptikimo) bei adaptyvių jų atnaujinimo (objekto sekimo).

Tam, kad būtų įmanomas įvairių algoritmų palyginimas, reikia iš anksto aptarti palyginimui tinkamiausius parametrus bei pasirinkti tokius, kurie geriausiai apibūdina objektų sekimo vaizde kokybę. Kadangi šio darbo tikslas yra algoritmų spartinimas, daugiausia dėmesio skiriama būtent sekimo spartai, tačiau dėmesys kreipiamas ir į kitus parametrus.

Įvertinus esamų algoritmų trūkumus galima pasiūlyti įvairias hipotezes bei jas patikrinti eksperimentiškai. Eksperimentai atliekami kuo įvairesniems vaizdams, kad būtų galima tinkamai įvertinti algoritmo kokybę apibūdinančias vertes bei palyginti šias su jau esamų algoritmų rezultatais. Po kiekvieno eksperimento pateikiami grafikai ir lentelės.

Darbo pabaigoje pateikiamas darbe pasiektų rezultatų apibendrinimas.

1. OBJEKTO VAIZDE KOORDINAČIŲ ADAPTYVAUS ATNAUJINIMO ALGORITMŲ ANALITINĖ APŽVALGA

1.1. Kalmano filtro taikymas objektams vaizde sekti

Kalmano filtras – vienas iš dažniausiai praktikoje naudojamų filtrų, turintis daugiau nei 20 skirtingų tipų skirtingiems tikslams pasiekti. Sumažinti klaidas leidžia metodas, veikiantis panašiai kaip mažiausiųjų kvadratų metodas tiesinių lygčių sistemų sprendimui (Chen 2012: 4409–4420). Šis metodas ypatingas tuo, kad išmatuoti duomenų taškai aproksimuojami kreive. Skirtingai nei interpoliacijos metu kreivė neprivalo eiti per visus taškus, todėl matematinė išraiška būna paprastesnė, o neretai labiau atitinka ir realų modelį. Taip nutinka todėl, kad visi matavimai turi tam tikrą triukšmo lygį, kuris paprastai yra atsitiktinis, todėl matavimo taškai yra išsibarstę aplink tikrąsias reikšmes. Kalmano filtro naudojimas gali padėti pasiekti tikslesnius matavimus lokalizavimo (Siekmann *et al.* 2010) ir objekto matavimo (Ahn *et al.* 2009: 4296–4302) uždaviniuose.

Lokalizavimo uždavinys yra vienas iš pagrindinių, darančių įtaką ateities robotų autonomiškumui. Nepaisant naujausių GPS (angl. *Global Positioning System*) technologijos tobulėjimų gaunami rezultatai dar nėra pakankamai tikslūs naudoti be papildomo filtravimo, o savaime greitai prisitaikantys jutikliai nėra prieinami (Chen 2012: 4409–4420) arba ženkliai padidintų įrenginių kainą. Tikslumo didinimas sumažina sistemos efektyvumą, o sparčiai veikianti sistema paprastai nėra pakankamai tiksli (Correa *et al.* 2010: 339–354). Kalmano filtras taip pat tinka automatiniam aptikimui ir sekimui (taikinių, pavyzdžiui, raketų, paieška), prognozavimui ir kt.

Vienas iš svarbių ir pakankamai sudėtingų uždavinių yra trijų ašių vaizdo rekonstrukcija iš dviejų ašių vaizdo. Vaizdas kompiuterio ekrane yra tik dviejų ašių projekcija, todėl tam, kad būtų galima nuspėti, koku atstumu ir kokio dydžio iš tikrųjų yra ieškomas objektas, reikalingi matematiniai modeliai. Šį uždavinį analizuoja SLAM (angl. *Simultaneous Localization And Mapping*) algoritmai. Viena šio algoritmo dalis – išplėstasis Kalmano filtras (angl. *Extended Kalman Filter*), dažnai trumpai vadinamas tiesiog EKF. EKF yra netiesinė Kalmano filtro versija. Dėl savo netiesiškumo šis filtras paprastai reikalauja daug skaičiavimų ir ne visada pateikia tenkinančius rezultatus, todėl skaičiavimus tenka kažkaip supaprastinti pasirenkant paprasčiau suprantamus požymius arba lengvai įvykdomas ir nedaug skaičiavimų reikalaujančias transformacijas. Apie Kalmano filtro veikimo principus, galimas transformacijas bei Kalmano filtro tipus kalbama tolimesniuose poskyriuose.

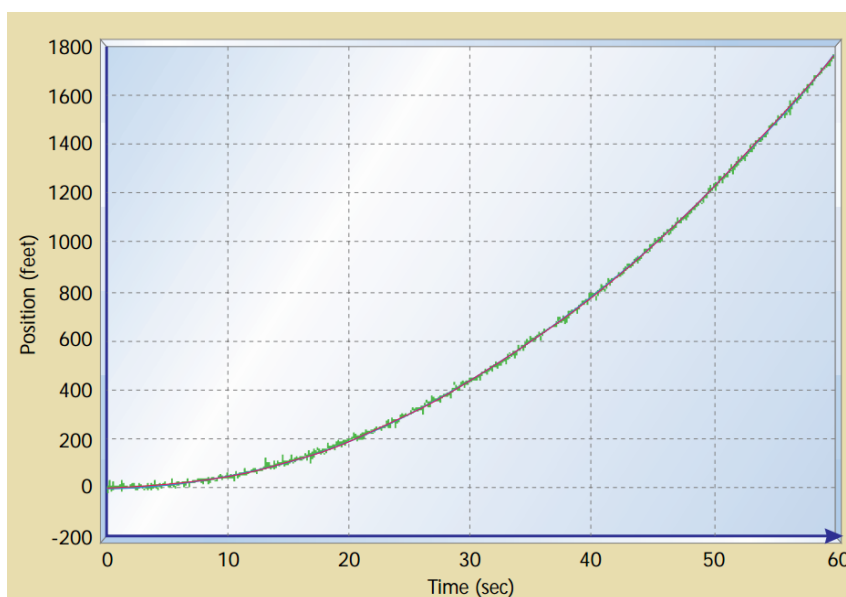
1.1.1. Veikimo principas

Tiesinę sistemą galima aprašyti dviem funkcijomis (Chen 2012: 4409–4420):

$$X_k = f(X_{k-1}, u_{k-1}, v_{k-1});$$

$$Z_k = h(X_k, w_k).$$

Šiose lygtyse daug nežinomųjų, kuriuos reikia išaiškinti. X_k yra būklės (angl. *state*) matrica, į kurią paprastai vaizdų apdorojimo atveju įeina įvairūs skirtingi dydžiai – koordinatės vaizde $x(k)$, $y(k)$, objekto judėjimo greitis įvairiomis koordinatėmis v_x ir v_y , taip pat posūkio kampas $\varphi(k)$ (Chen 2012: 4409–4420). Tam tikrais atvejais aprašomas ir sekimo langas (angl. *tracking window*). X_{k-1} apibūdina paskutinės būklės vertes, u_{k-1} – įėjimo matrica, v_{k-1} – būklės triukšmo matrica, Z_k – išėjimo matrica, w_k – išėjimo vertės triukšmo matrica. Galima sakyti, kad Kalmano filtras vienu metu atlieka ne vieno elemento, o kelių su vaizdų atpažinimu ir sekimu susijusių verčių duomenų filtravimą.



1.1 pav. Transporto priemonės pozicija (tikroji, pamatuota ir įvertinta) (Simon 2001: 72–79)

Kalmano filtro užduotis – įvertinti būklės ir išėjimo matricas. Būklės matrica naudojama tolimesnėms išėjimo vėrtėms apskaičiuoti, todėl galime sakyti, kad šis filtras prognozuoja vertes pagal prieš tai buvusias ir pagal jas derina skaičiavimus. Šioms dviem matricoms skaičiuoti reikalingi papildomi skaičiavimai, o galiausiai viskas suvedama į tris lygtis (Chen 2012: 4409–4420):

$$K_{gk} = P_{k|k-1} H^T (H P_{k|k-1} H^T + R)^{-1};$$

$$\hat{X}_{(K|K)} = \hat{X}_{(K|K-1)} + K_{gk} (Z_{(K)} - H X_{(K|K-1)});$$

$$P_{k|k} = (I - K_{gk} H P_{k|k-1}).$$

K_{gk} yra vadinama Kalmano stiprinimo matrica. Į jo skaičiavimo lygtį įeina buvusių reikšmių kovariacija $P_{k|k-1}$, padedanti įvertinti labiausiai tikėtiną reikšmę. H apibūdinama kaip svorio matrica ir paprastai yra susieta su matavimais z_k , R – matavimo triukšmo kovariacija (Simon 2001:

72–79), I – proceso triukšmo kovariacija (Simon 2001: 72–79), $\hat{X}$ apibūdina optimalią būklės matricą. Kalmano filtro stiprinimo matrica yra keičiama atsižvelgiant į taškų kovariaciją, kuri savo ruožtu keičiama atsižvelgiant į naujai išmatuotas reikšmes. Kitaip sakant, jeigu nauja reikšmė smarkiai skiriasi nuo paskutinių reikšmių, Kalmano filtro stiprinimo koeficientas sumažinamas, kad nebūtų iškreiptas naujų reikšmių skaičiavimas. Esant nedideliame Kalmano stiprinimui nauja reikšmė neturės daug įtakos rezultatams.

Kai procesas, kurį reikia išmatuoti, nėra tiesiškas, išplėstiniame Kalmano filtre (EKF) panaudojama tikslo funkcijos aproksimacija Teiloro eilute. Taip galima įvertinti ir netiesiškas sistemas (Chen *et al.* 2010: 576–581).

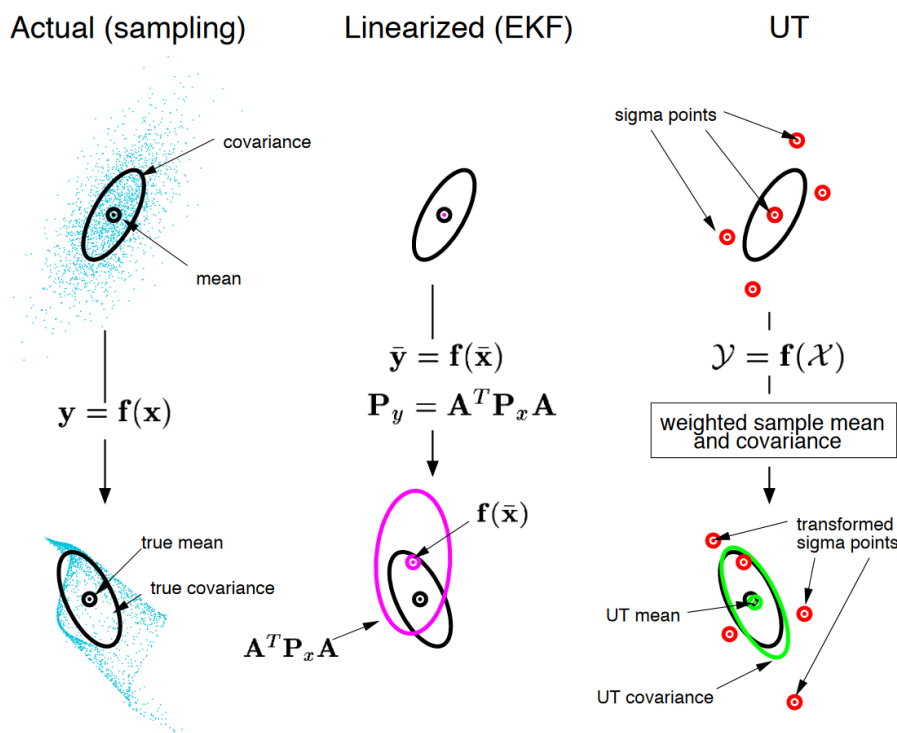
Tinkamas Kalmano filtru aproksimuotos vertės pavyzdys pateiktas 1.1 paveiksle. Jame nurodyti matuotų ir filtruotų reikšmių grafikai. Abscisių ašyje nurodytas laikas sekundėmis, ordinačių – pozicija. Paveiksle galima matyti pradinį signalą su triukšmu, taip pat tikrąjį (etaloninį) ir gautą filtruojant. Galima matyti, kad Kalmano filtru galima gauti reikšmes, kurios beveik neatskiriamos nuo realių.

1.1.2. Problemos ir galimi jų sprendimai sekant objektus vaizde

Nors Kalmano filtrai yra pakankamai ištobulinti matavimo ir prognozavimo užduotims bei leidžia spręsti daug uždavinių, problemų vaizdų sekime išlieka (Chen 2012: 4409–4420). Šias problemas siekia išspręsti tyrėjai, siūlydami pakeitimus tiek skaičiavimo metodikoje, tiek aparatinėje įrangoje.

Viena iš pagrindinių problemų sekime yra objektų aptikimo sparta. Greitai veikiantys algoritmai nepateikia pakankamai tikslų rezultatų, o tikslesni reikalauja skaičiavimų intensyvumo. Vaizduose dažnai tinkama tik netiesinė Kalmano filtro versija, kurios taikymui reikia daugiau skaičiavimų nei tiesinėje versijoje.

Kita problema – patikimumas. Kalmano filtras reikalauja, kad filtro parametrai būtų suderinti. Kadangi procesai didele dalimi yra stochastiniai, jų suderinti gali būti sudėtinga ir tikslumo pagerinimui gali tekti papildomai vertinti įvairius dinامينius bruožus (Zhang *et al.* 2010: 424–437). Šitokiu atveju atpažinimo patikimumas ir stabilumas turėtų padidėti, tačiau algoritmas neišvengiamai tampa sudėtingesnis ir reikalauja daugiau skaičiavimų. Papildomi skaičiavimai turi būti kuo paprastesni.



1.2 pav. UKF naudojamos transformacijos rezultatų pavyzdys. Pirmasis stulpelis – realūs matavimai, antrasis – EKF filtro rezultatai, trečiasis – UKF naudojamos transformacijos rezultatai (Wan *et al.* 2002: 153–158)

Dar viena problema – tikslumo ir efektyvumo suderinimas. Net ir išmintingai parinktas būdas gauti filtravimui naudojamus matavimo taškus ne visada padeda sumažinti skaičiavimo intensyvumą. Siūlomas sprendimas – vietoj EKF naudoti UKF (angl. *Unscented Kalman Filter*), kuris taip pat skirtas darbui su netiesinėmis sistemomis, tačiau smarkiai sumažina aproksimavimo paklaidas. Panaudojant UKF tiek taškų kovariacija, tiek vidurkis paprastai yra artimesni realiam modeliui (Wan *et al.* 2002: 153–158). Rezultatus, kuriuos gavo Wan ir van der Merve, galima matyti 1.2 paveiksle. Pirmasis stulpelis – realiam modeliui pritaikyti rezultatai, antrajame nurodyti EKF gauti rezultatai, trečiame – filtravimu UKF gauti rezultatai.

Tikėtina, kad spartesnį ir kokybiškesnį sekimą galima pasiekti pasitelkus įvairius dirbtinio intelekto metodus. Tarp tokių metodų – apytikslė logika (angl. *fuzzy logic*), kuri leidžia įgyti ne tik dvi reikšmes – loginį vienetą ir loginį nulį – bet ir reikšmes tarp šių dviejų ribinių reikšmių; dirbtinių neuronų tinklai, genetiniai algoritmai, padedantys ieškoti optimizavimo ir paieškos uždavinių sprendimų. Šiuos metodus galima taikyti atskirai arba kartu ir stebėti, ar rezultatai atsveria papildomų skaičiavimų atsiradimą ir algoritmo tapimą sudėtingesniu. Kai kurios Kalmano filtro modifikacijos, pasitelkiančios dirbtinį intelektą, jau yra naudojamos ir pateikia pakankamai smarkiai patobulintus rezultatus. Vienas toks Kalmano filtro modifikacijos pavyzdys – PSO-KF (angl. *Particle Swarm Optimization Kalman Filter*) (Ramakoti *et al.* 2009: 531–533). Spartos didinimui galėtų padėti loginės programuojamos matricos (angl. *Field-Programmable Gate Array*),

nes atlieka skaičiavimus lygiagrečiai visam duomenų blokui, tačiau net ir pakankamai didelėse programuojamose loginėse matricose gali greitai pritrūkti loginių blokų. Su tokiu uždaviniu nesunkiai galėtų susitvarkyti staliniai kompiuteriai, tačiau idealiu atveju norėtusi, kad objekto sekimas būtų pasiekiamas ir planšetiniuose kompiuteriuose ar išmaniuosiuose telefonuose. Spartinti skaičiavimus galima tam tikrą jų dalį perkeltiant loginėms matricoms. Dabartinės loginės matricos (tiek *Xilinx*, tiek *Altera*) jau turi integruotus procesorius (paprastai ARM architektūros), kurie gali didelę dalį užduočių pavesti skaičiuoti pačiai loginei matricai. Skaičiavimai turėtų būti tokie, kad jų nereikėtų atlikti visai duomenų kadru, o tik tam tikram duomenų blokui, o neuronų tinklai ir kiti dirbtinio intelekto metodai būtų kuo paprastesni.

1.2. Dalelių filtro taikymas objektams vaizde sekti

Kitas dažnai naudojamas filtras – dalelių (angl. *particle filter*). Sukurtas 1993 metais kaip Bajeso filtrų aproksimacija ir dabar laikomas pakankamai ištobulintu filtru vaizdų apdorojimo, objektų sekimo, lokalizavimo ir kitoms problemoms spręsti. Jis gali būti nesunkiai palyginamas su jau aptartu išplėstiniu Kalmano filtru. Kalmano filtras ne visada pateikia tenkinančius rezultatus, tad dėl savo konstrukcijos dalelių filtras gali būti naudingas. Šio filtro pagrindu sukonstruotas fastSLAM algoritmas, kuris yra dar viena SLAM uždavinio sprendimo rūšis. Kaip ir Kalmano filtro atveju, trumpai bus aptartas veikimo principas bei galimos problemos ir galimi jų sprendimo būdai bei įvairios galimos modifikacijos, gerinančios galutinį rezultatą.

1.2.1. Veikimo principas

Dalelių filtro esmė – skaičiavimams naudojama tikimybių tankio funkcija. Lėktuvo, skrendančio virš ežero, apaugusio meldais, poziciją žemėlapyje galima aprašyti tikimybių tankio funkcija. Vėliau, atliekant matavimus, pavyzdžiui, aukščio virš ežero, tikimybių tankio funkcija gali būti panaudota tikslesniam lėktuvo lokalizavimui žemėlapyje. Jo algoritmą (Gustafsson 2010: 528–82) iš esmės galima padalyti į penkias dalis, iš kurių paskutinės keturios vykdomos iteratyviai:

Pradžioje vykdomas algoritmo inicijavimas – pasirenkamas pasiskirstymas $q(x_{k+1}|x_{1:k}, y_{k+1})$, perrinkimo (angl. *resampling*) strategija bei dalelių skaičius N . Tada generuojama $x_1^1 \sim p_{x_0}$, kur $i = 1, \dots, N$, taip pat generuojama svorių matrica $w_{1|0}^i = \frac{1}{N}$.

Toliau einančios algoritmo dalys yra pasikartojančios. Pirmoji iš jų – matavimų atnaujinimas (angl. *measurement update*):

$$w_{k|k}^i = \frac{1}{c_k} w_{k|k-1}^i p(y_k | x_k^i).$$

Formulėje c_k - normalizacijos svoris:

$$c_k = \sum_{i=1}^N w_{k|k-1}^i p(y_k | x_k^i).$$

Antroji dalis – įvertinimas. Čia perskaičiuojama tikimybių tankio funkcija ir jos vidurkis:

$$\hat{p}(x_{1:k} | y_{1:k}) = \sum_{i=1}^N w_{k|k}^i \sigma(x_{1:k} - x_{1:k}^i);$$

$$\hat{x} = \sum_{i=1}^N w_{k|k}^i x_{1:k}^i.$$

Trečioji dalis – perrinkimas. Kadangi vieni taškai turi didesnę tikimybę nei kiti, o kai kurie yra per mažai tikėtini ir gali būti atmesti, reikia paaimti N dalelių skaičių, kurių svorių matrica $w_{k|k}^i = \frac{1}{N}$.

Paskutinė, ketvirtoji dalis – generuoti spėjimus ir kompensuoti matavimo svarbos svorį (angl. *importance weight*):

$$x_{k+1}^i \sim q(x_{k+1} | x_k^i, y_{k+1});$$

$$w_{k+1|k}^i = w_{k|k}^i \frac{p(x_{k+1}^i | x_k^i)}{q(x_{k+1}^i | x_k^i, y_{k+1})}.$$

1.2.2. Problemos ir galimi jų sprendimai sekant objektus vaizde

Nors dalelių filtras jau pakankamai ištobulintas, yra neišspręstų problemų, turinčių įtakos algoritmo spartai. Šių problemų įtaką siekiama sumažinti naujausiuose moksliniuose darbuose.

Pirmoji ir turbūt dažniausiai sutinkama problema – taškų nykimas (angl. *sample depletion*), dar vadinamas taškų degeneracija (angl. *sample degeneracy*). Jis apibūdinamas kaip sparti rezultatų divergencija - visų taškų svoriai artėja prie nulio ir tik kurio nors vieno artėja prie vieneto. Iš pirmo žvilgsnio tai atrodytų kaip artėjimas prie idealaus sprendinio, tačiau iš tikrųjų tai reiškia, kad vienas duomenų taškas yra daug daugiau tikėtinas nei kiti. Būtent dėl šios priežasties ir yra vykdomas taškų perrinkimas (angl. *resampling*) (Gustafsson 2010: 528–82). Jis įveda grįžtamąjį ryšį, kitaip sakant, tikėtini taškai vis tiek paprasčiau *atlaikys* perrinkimą. Perrinkimas sprendžią problemą, bet sukuria naują – sunaikinama pradinė informacija ir taip padidėja sprendimo neapibrėžtumas, todėl perrinkimas turi būti atliekamas tik tada, kai tam yra būtinybė. Perrinkimas kartais pakeičiamas naujų duomenų gavimu atlikus pilną dalelių filtro algoritmo iteraciją (po ketvirtosios dalies, aptartos veikimo principo skyrelyje). Pavyzdžiui, Gauso dalelių filtras (angl. *Gaussian Particle Filter*, GPF) naujų dalelių debesiai pritaiko Gauso skirstinį.

Geras indikatorius taškų nykimui apibūdinti yra efektyvus taškų skaičius N_{eff} , kurio aprašymui naudojamas variacijos koeficientas c_v (Gustafsson 2010: 528–82):

$$N_{eff} = \frac{N}{1 + c_v^2(w_{k|k}^i)}.$$

Efektyvus taškų skaičius yra didžiausias ($N_{eff} = N$), kai visų taškų svoriai yra lygūs $w_{k|k}^i = 1/N$, o mažiausias taškų skaičius $N_{eff} = 1$; šis atvejis pasitaiko, kai $w_{k|k}^i = 1$ su tikimybe $1/N$, taip pat mažiausias tada, kai $w_{k|k}^i = 0$ su atsitikimo tikimybe $(N - 1)/N$.

Pastaroji paminėta efektyviųjų taškų skaičiaus formulė nėra naši ir lengvai suskaičiuojama, ją galima pakankamai smarkiai supaprastinti ją aproksimuojant kur kas paprastesne formule:

$$\hat{N}_{eff} = \frac{1}{\sum_i (w_{k|k}^i)^2}.$$

Tokia aproksimacija galima, nes $1 \leq \hat{N}_{eff} \leq N$. Viršutinė padėtis $\hat{N}_{eff} = N$ pasiekama tada, kai visos dalelės turi tą patį svorį, o apatinė padėtis $\hat{N}_{eff} = 1$ – tada, kai visas svoris suteikiamas viso labo vienai dalelei.

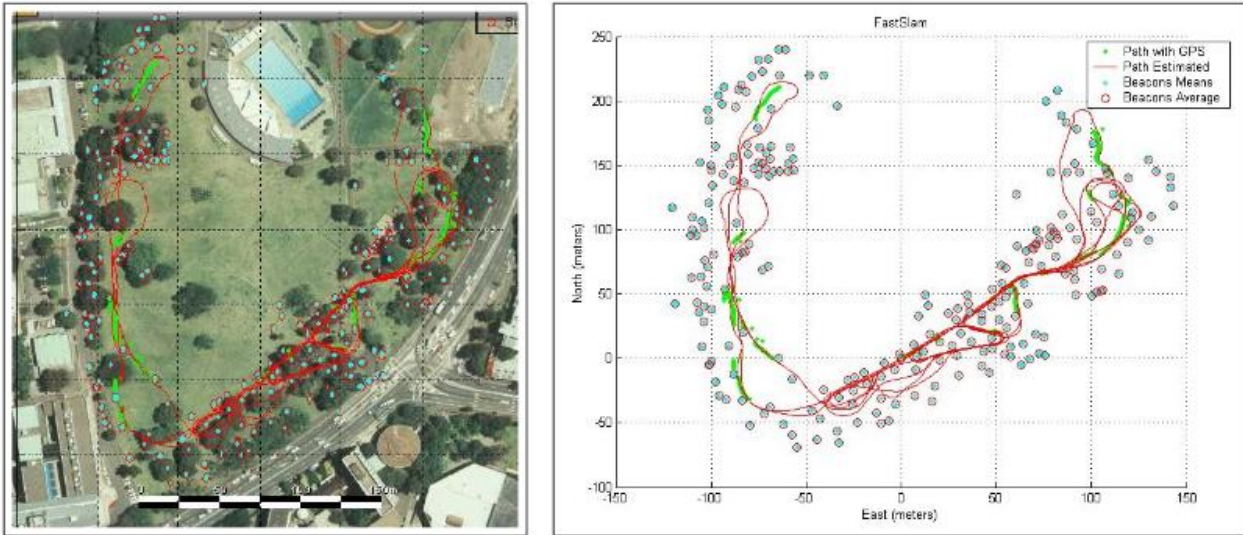
Dar vienas kartais naudojamas būdas siekiant išvengti perrinkimo sukuriama taškų nykimo problemos – prie signalo pridėti papildomo triukšmo (angl. *jitter*) (Fearnhead 1998). Šitaip dalelių filtre bus daugiau triukšmo negu buvo duomenų generavimo procese, o link rezultato bus artėjama lėčiau ir dalelės lengviau *atlaikys* perrinkimo procesą.

Klasikinis perrinkimo algoritmas vadinamas Riplio (angl. *Ripley*) metodu ir atliekama reguliariai, kai efektyvus taškų skaičius tampa per mažas (Myers *et al.* 1995):

```
function [x,w]=resample(x,w)
% Multinomial sampling with Ripley's method
u=cumprod(rand(1,N). A (1.1 [N:-1:1]));
u=flip1r(u);
wc=cumsum(w);
k=1;
for i=1:N
    while (wc(k)<u(i))
        k=k+1;
    end
    ind(i)=k;
end
x=x(ind,:);
w=ones(1,N) • IN;
```

Šioje perrinkimo funkcijoje yra lygiagrečių skaičiavimų, kuriuos aukšto lygio programavimo kalbose, kaip MATLAB programinė įranga, spartina vektorių ir matricų operacijos. Įterptinėse sistemose darbas su matricomis ir lygagregiais skaičiavimais gali būti paspartintas lauku programuojamųjų loginių matricų arba grafinių plokščių pagalba, todėl tikėtina, kad šią skaičiavimų dalį perkėlus skaičiuoti joms algoritmo sparta ir efektyvumas didėja.

Dalelių filtro algoritmas, nors gali pasigirti gerais rezultatais, reikalauja daugiau skaičiavimų nei aptartas išplėstasis Kalmano filtras (EKF), nes turi daugiau žingsnių (pridedamas duomenų perrinkimas) (Gustafsson 2010: 528–82), todėl turėtų būti naudojamas ten, kur yra pakankamai techninių resursų jam išpildyti.



1.3 pav. fastSLAM algoritmo pavyzdys (Thrun 2012)

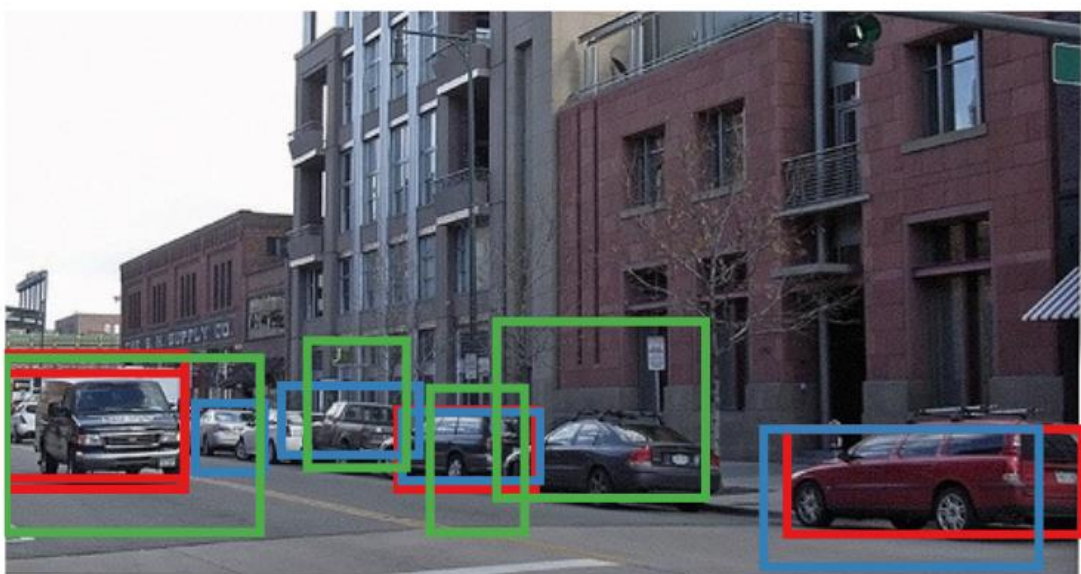
Kaip ir Kalmano filtro atveju, dalelių filtras turi įvairių modifikacijų, kurios išbandomos ir vaizdų atpažinimo ir sekimo srityje. Viena iš jų – *Rao-Blackwellized* dalelių filtras (RBPF), dar kitaip vadinamas *marginalised particle filter* (MPF). Jo tikslas yra išlaikyti filtrą kiek galima mažesnį, kad šis dar galėtų būtų išpildomas. Būtent šis filtras pritaikytas fastSLAM algoritmo atveju, kurio pavyzdį galima matyti 1.3 paveiksle. Keleto objektų sekimui pasiūlytas ir išbandytas ir diskriminatyviai mokytas dalelių filtras (angl. *discriminately trained particle filter*). Šis filtras sėkmingai išpildytas sekant 22 futbolo žaidėjus futbolo aikštėje (Hess *et al.* 2009: 240–247), kur objektai tam tikrais momentais vienas kitą užstoja ir taip apsunkina filtro veikimą.

2. OBJEKTO KOORDINAČIŲ ATNAUJINIMO ALGORITMŲ IR JUOSE NAUDOJAMŲ METODŲ ANALIZĖ

Objektų aptikimo užduoties sprendimai, kurių tikslumas jau viršija eksperto žmogaus tikslumą (He *et al.* 2015: 1026–1034), skatina išbandyti šiuos tinklus ir objektų aptikimui vaizde ir kitoms su vaizdais susijusiose užduotyse, tarp jų ir objektų sekime, kur objekto aptikimas yra viena iš užduoties sudėtinių dalių. Tam pasiūlytos įvairios architektūros, kurios leidžia iš pradžių pasirinkti stačiakampius, kuriuose, tikėtina, pasitaikys objektas, o vėliau tiems stačiakampiams pritaikyti konvoliucinio neuronų tinklo grafu apibūdintas matematinės operacijas. Tarp populiariausių – *Faster R-CNN* (Ren *et al.* 2017: 1137–1149), *Masked R-CNN* (He *et al.* 2017) ir *YOLO* (angl. *You Only Look Once*) (Redmon *et al.* 2015; Redmon *et al.* 2016) architektūros. Būtent apie pastarąją kalbama toliau.

2.1. Konvoliuciniais dirbtinių neuronų tinklais paremti metodai

Konvoliuciniai neuronų tinklai puikiai tinka klasifikavimo uždaviniui spręsti ir savo kokybe lenkia praktiškai bet koki klasifikavimo algoritmą, tačiau reikalauja daugiau skaičiavimų nei paprasti ir netikslūs klasifikatoriai. Konvoliuciniai neuronų tinklai sudaryti iš daugybės filtrų, kurių išėjimams suskaičiuoti naudojama konvoliucijos (sąsukos) operacija, o parametrų skaičius dažnai viršija kelias dešimtis milijonų. Logiška, kad jeigu langų skaičius, kuriuose bandoma klasifikuoti objektus, bus toks pats, kaip tiesinių klasifikatorių atveju, rezultato laukimas prailgs, ir todėl net pačios geriausios grafinės plokštės, kurias galima įsigyti rinkoje, negalės atlikti skaičiavimų realiu laiku.



2.1 pav. Susidomėjimo vertų plotų ieškančių algoritmų rezultatų pavyzdys (Hosang *et al.* 2016: 814–830)

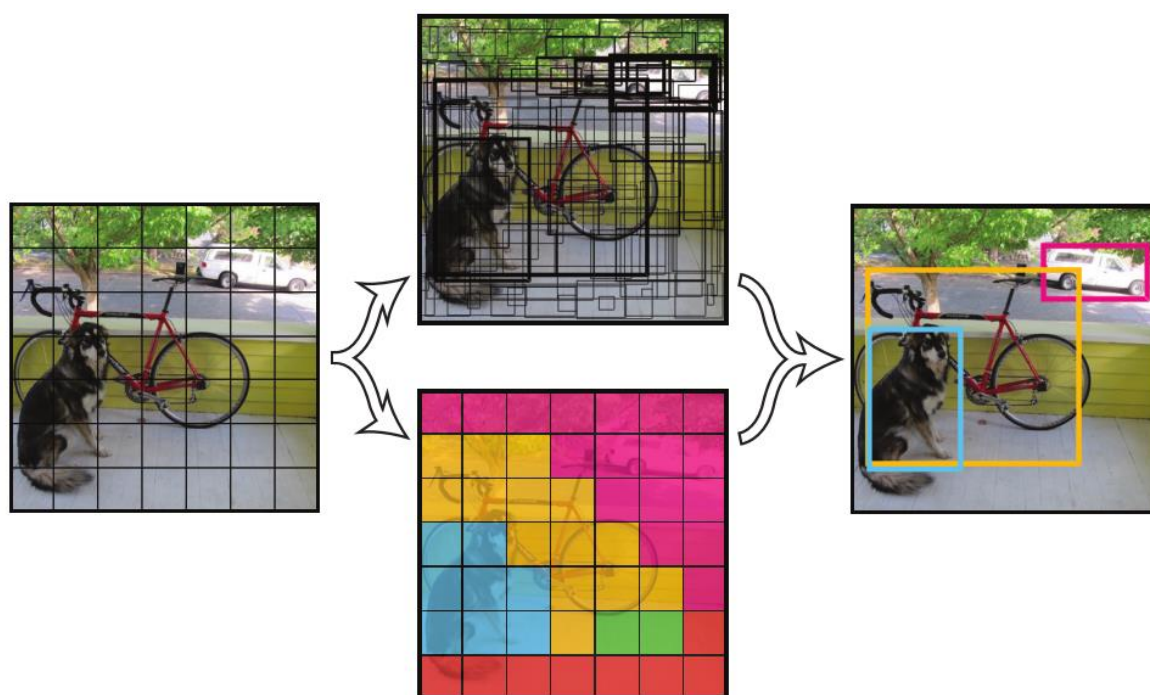
Langų, kuriuose atliekama objekto paieška, kiekio sumažinimas gali būti pasiekiamas algoritmu pasirenkant atitinkamus vaizdo plotus, kuriuose, panašu, kad galima rasti objektą (Hosang *et al.* 2016: 814–830; Chavali *et al.* 2015). Tokių plotų radimas yra greitas panaudojant nesudėtingus klasifikatorius, kurie nekreipdami dėmesio į atskiras klases bando identifikuoti objektus vaizde. Vėliau konvoliucinis neuronų tinklas gali tiksliau įvertinti, ar paprastesnio klasifikatoriaus spėjimas buvo teisingas ir, jeigu taip, įvardinti klasę tiksliau. Šių klasifikatorių yra įvairių, pagrindiniai apžvelgiami kiek vėliau. Dominančių stačiakampių radimo algoritmo rezultatas pateikiamas 2.1 paveiksle (Hosang *et al.* 2016: 814–830). Nesunku pastebėti, kad atmetus stumdomo lango algoritmą, tikrintinų keturkampių skaičius smarkiai sumažėja.

Pastarasis metodas – dominančių vaizdo plotų paieška – panaudotas keliems vaizdo atpažinimo vaizde algoritmams, pavadintiems R-CNN šeima (angl. *Regions with Convolutional Neural Networks features*). Nuo seniausios versijos (R-CNN) atlikta daugybė patobulinimų (Fast R-CNN, Faster R-CNN, taip pat neseniai išleistas Mask R-CNN). Pirmojoje versijoje stačiakampių pasiūlymai vykdomi tiesiai iš pradinio vaizdo (paprastai gaunama apie 2000). Po to viena tinklo dalis apskaičiuoja klases, kita – patikslina stačiakampio koordinates (Zhang *et al.* 2014: 834–849). Tokiam tinklui sunku atlikti mokymą, o pats tinklas veikia itin lėtai, todėl antrojoje versijoje (Fast R-CNN) šie skaičiavimai perkeliama jau už sąsukos operacijų, taip sutaupant skaičiavimų (Girshick 2015: 1440–1448). Galiausiai nuspręsta, kad neverta daryti spėjimų iš pradinio vaizdo – daug sparčiau tai atlikti tiesiai iš rezultatų, gautų po paskutinės sąsukos operacijos tinkle (Ren *et al.* 2017: 1137–1149). Tam specialiai panaudojamas papildomas konvoliucinis tinklas, greitai ir tiksliai sugebantis prognozuoti dominančius vaizdo stačiakampius. Pastaruoju tinklu pasiektas 66,9 % mAP (angl. *mean average precision*, suvidurkintas vidutinis atpažinimo tikslumas, skirtas nurodyti, kaip teisingai atspėjama klasė ir parinktas stačiakampis objektui apibrėžti, apie šį įvertinimo parametą išsamiau kalbama tolimesniuose skyriuose). Palyginus su senosiomis versijomis atpažinimo tikslumas didėjo nesmarkiai (nuo 66 iki 66,9 %), tačiau rezultatų gavimo sparta padidėjo 250 kartų. Paskutinė šio tinklo versija – Mask R-CNN (He *et al.* 2017) – praktiškai nesiskiria nuo Faster R-CNN, tačiau papildomai leidžia vaizdui atlikti ir semantinę segmentaciją.

2.2. YOLOv2 objektų aptikimo algoritmo architektūra

Kai daugelis mokslininkų rinkosi objektų atpažinimą, kuriam susidomėjimo vertus vaizdo plotus siūlė koks nors papildomas algoritmas (pavyzdžiui, aptartos pasirinktinės paieškos), Vašingtono universiteto tyrėjas Josephas Redmonas kartu su kolegomis pasiūlė visiškai kitokį vaizdo apdorojimo principą, išmesdamas susidomėjimo vertų plotų paiešką, taip siekdamas sukurti gilų pilnai konvoliucinį neuronų tinklą, galintį objektus vaizde atpažinti realiu laiku. Iš pradžių galėjęs atpažinti kelias dešimtis skirtingų klasių, vėliau, antrojoje versijoje patobulintas ir dabar gali

atpažinti net 9000 skirtingų klasių su aukštesniu tikslumu bei tai atlikti sparčiai. Pasak tyrėjų, mokymui pritaikius VOC2007+2012 duomenų rinkinį, o testavimui – VOC2007, pateikus 544×544 dydžio vaizdus pasiektas suvidurkintas vidutinis atpažinimo tikslumas (mAP) – 78,6 % (taigi didesnis nei deklaruojama Faster R-CNN straipsnyje). Vienas iš pagrindinių privalumų – *Nvidia GeForce Titan X* grafinę plokštę panaudojant skaičiavimams tokio dydžio paveikslėliams pasiekta 40 kadrų per sekundę sparta, o sumažinus paveiksluką ir prarandant šiek tiek tikslumo – net iki 100 kadrų (Redmon *et al.* 2016). Nors tokia įranga dabar įperkama nedaugeliui, priartėjama prie galimybės tokius objektų atpažinimo modelius sutalpinti į įterptinę sistemą ir juos vykdyti realiu laiku, pavyzdžiui, *Nvidia Jetson TX2*.



2.2 pav. YOLO algoritmo veikimo pavyzdys (Redmon *et al.* 2015)

YOLOv2 atlieka skaičiavimus visam paveikslėliui iš karto, kitaip sakant, nereikalinga pirmiausia išsiskirti susidomėjimo vertų blokų ir kiekvienam iš jų spręsti klasifikavimo uždavinio. Šiuo atveju tereikia praleisti vaizdą pro pilnai konvoliucinį neuronų tinklą bei klasifikatoriaus pateiktus rezultatus, kurių užtikrintumas dėl klasės ir ją ribojančio stačiakampio pasirinkimo yra žemesnis nei koks nors pasirinktas laiptelis, tiesiog išmesti. Tokio algoritmo veikimo pavyzdys pateiktas 2.2 paveiksle (Redmon *et al.* 2015).

YOLO algoritmas sukurtas taip, kad iš karto atliktų skaičiavimus visam paveikslėliui. Iš to ir kilo jo žaismingas pavadinimas (angl. *You Only Look Once*). Tam įėjimo paveikslėlis padalinamas į $S \times S$ dydžio kvadratukus. Jeigu objekto centras pakliūna į atitinkamą ląstelę, klasifikuojamas objektas. Vienas langelis gali būti reikšmingas spėliojant B kiekį objektą apibrėžiančių stačiakampių, bei kiekvienam tokiam spėjimui pateikiamas skaičius, įvertinantis, kaip užtikrintai

tinklas siūlo tokį klasifikavimo rezultatą – objektą viduje ir stačiakampio koordinates. Jeigu nerandamas joks objektas, užtikrintumas dėl klasifikavimo yra lygus nuliui.

Kiekvienas stačiakampis apibūdinamas penkiais parametrais – x , y , w , h ir užtikrintumu (angl. *confidence*). x ir y reiškia stačiakampio centro koordinates palyginus su kvadratuko kraštais. w ir h apibūdina tokio stačiakampio plotį ir aukštį palyginus su visu paveikslėliu. Užtikrintumas apibūdina ne tik tikrumą dėl klasės, bet ir dėl ką tik paminėtų keturių parametru, aprašančių objektą ribojantį stačiakampį. Visa tai atlieka vienas pilnai konvoliucinis neuronų tinklas, neturintis pilnai sujungtų sluoksnių, primenančių klasikinio neuronų tinklo struktūrą. Iš viso toks tinklas kiekvienam mažam vaizdo kvadratu, į kuriuos padalintas vaizdas, išveda $S \times S \times (5B+C)$ matricą. Pavyzdžiui, dalinant paveikslėlį į 7×7 kvadračiukus, vienam kvadratu skiriant du spėjimus bei turint 20-ies klasių duomenų rinkinį išvesties matrica bus $7 \times 7 \times 30$ dydžio. Vėliau dalį spėjimų atmetant dėl per mažo užtikrintumo gaunami galutiniai spėjimai (Redmon *et al.* 2015).

Antroji YOLO versija išlaikė pirmosios versijos sėkmę lėmusius privalumus, tačiau pridėjo daugybę patobulinimų, leidžiančių sėkmingai atskirti didesnę klasių kiekį su truputį didesne sparta ir užtikrintumu. Tarp tokių labiausiai verti paminėjimo duomenų rinkinio normalizavimas (angl. *batch normalization*), didesnės raiškos paveikslėlių naudojimas mokymui, nauja konvoliucinio tinklo versija (pavadinimu *Darknet-19*). Visi atlikti pakeitimai aptarti moksliniame straipsnyje (Redmon *et al.* 2016).

3. NAUJO BŪDO PASIRINKTAM ALGORITMUI PASPARTINTI SUDARYMAS IR EKSPERIMENTINIO TYRIMO DUOMENŲ PARUOŠIMAS

3.1. Objektų sekimo vaizde įvertinimo metrikos

Tvirtinti, kad algoritmas veikia greičiau ir dėl to yra geresnis nei jo analogai, negalima. Pasiiekti didesnę veikimo spartą galima nekreipiant dėmesio į įvairius parametrus, apibūdinančius sekimo kokybę, tačiau algoritmas veikia greičiau dėl to veiks tik blogiau, todėl spartinant algoritmus svarbu sekti ir įvairius jį įvertinančius parametrus. Objekto sekimą apibūdinančių parametrų yra sukurta įvairių, kai kurie iš jų skirti panašioms objektų sekimo aspektams apibūdinti, tad svarbu motyvuotai pasirinkti tinkamiausias magistro darbo užduočiai atlikti. Trumpa jų apžvalga ir pasirinkimas atliksimiesiems moksliniams tyrimams yra šio skyriaus užduotis.

Anot Smito (Smith *et al.* 2005), idealiu atveju algoritmas objektams vaizde sekti turėtų atlikti šešias funkcijas:

1. pradėti sekimą automatiškai;
2. vaizde tinkamose vietose nurodyti tinkamą kiekį objektų;
3. sugebėti nenutrūkstamai sekti objektus vaizde;
4. sekti objektus nepaisant kliūčių (pasikeitus apšvietimui, vienam objektui užstojant kitą ir panašiai);
5. tiksliai išmatuoti objekto parametrus (pavyzdžiui, objekto greitį);
6. būti greitas.

Skirtingų algoritmų greitį galima palyginti skaičiuojant instrukcijų skaičių, tačiau nežymus ir akimi nematomas spartinimas nebūtinai prasmingas, ypač, jeigu dėl to tampa sunkiai paaiškinti algoritmo veikimą ir dėl to kodo palaikymas tampa sudėtingesnis. Be to, toks testavimo procesas būtų labai sudėtingas ir todėl jį taikyti nėra prasminga. Pasirinktas sprendimas – vertinti apdorotus kadrus per sekundę (angl. *frames per second*, FPS). Be abejo, tam, kad būtų galima įvertinti spartos pasikeitimus, testus reikia atlikti skirtingų dydžių vaizdams. Tokie testai parodytų, kaip algoritmo sparta keičiasi nuo įvesties masyvo (vaizdo) dydžio. Paspartinėjimas nedidelės raiškos vaizdams nebūtinai būtų pastebimas apdorojant aukštos raiškos vaizdus.

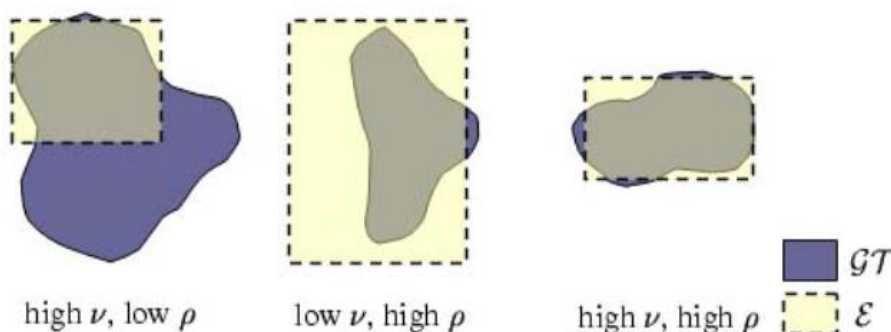
Objektus vaizde galima pažymėti stačiakampiais. Jeigu šis stačiakampis tiksliai apibrėžia objektą, jį galima pavadinti etalonu (angl. *ground truth*, GT). Objektą sekantis algoritmas taip pat pateikia stačiakampį, kurį galima pavadinti tiesiog matavimu (angl. *estimate*, E). Du fundamentalūs parametrai, apibūdinantis atpažinimo kokybę bei nepriklausantys nuo etalono (GT) ir matavimo (E) formos, yra etalono padengimo matavimu tikslumas (angl. *recall*) bei matavimo panaudojimo

etalonui padengti tikslumas (angl. *precision*) (Smith *et al.* 2005). Šie pavadinimai gali atrodyti panašūs, tačiau turi skirtingą prasmę.

Etalono padengimo matavimu tikslumas apibūdina, kiek matavimo stačiakampis padengia etaloninio stačiakampio vieneto dalimis arba procentais. Gali būti, kad nepaisant aukšto koeficiento objekto sekimo rezultatai bus nepatenkinami. Kone visas etaloninis stačiakampis bus uždengtas matavimo stačiakampiu, tačiau etaloniniam stačiakampiui dengti tebus panaudota dalis matavimo stačiakampio. Šį koeficientą $p_{i,j}$ galima apskaičiuoti:

$$p_{i,j} = \frac{|E_i \cap GT_j|}{|GT_j|}$$

čia E_i – objekto matavimo stačiakampio užimamas plotas, GT_j – objekto etaloninio stačiakampio plotas.



3.1 pav. Matavimo panaudojimo etalonui padengti tikslumo (ν) ir etalono padengimo matavimu tikslumo (ρ) parametrų grafinis atvaizdavimas (Smith *et al.* 2005)

Matavimo panaudojimo etalonui padengti tikslumas apibūdina, kokia išmatuoto stačiakampio dalis dengia etaloninį stačiakampį. Labai tikėtina, kad aukštas koeficientas negarantuoja aukštos sekimo kokybės (3.1 pav.). Nors ir visas matavimo stačiakampis panaudotas objektui padengti, tačiau juo padengiamas ne visas objektas. Šį koeficientą $\nu_{i,j}$ galima apskaičiuoti:

$$\nu_{i,j} = \frac{|E_i \cap GT_j|}{|E_i|}$$

Apibendrinti metrikas, naudojamas objektų sekimui vaizde apibūdinti, bandyta *MOTChallenge* kūrėjų (Milan *et al.* 2016: 1–12). Šiuo projektu siekiama standartizuoti įvairių objektų sekimo vaizde algoritmų įvertinimą, pateikiant visiems tyrėjams prieinamą duomenų rinkinį su itin daug mokymo pavyzdžių. Čia išskiriami du pagrindiniai reikalavimai. Pirmasis – kiekvienam atpažintam objektui reikalinga nustatyti, ar atpažintas objektas yra atspėtas teisingai (angl. *true positive*, TP), ar neteisingai (angl. *false positive*, FP). Taip pat pažymima, jeigu objektas vaizde nebuvo atpažintas visai (angl. *false negative*, FN). Antrasis reikalavimas reikalauja, kad praradus objekto sekimą vėliau šis gali būti atnaujintas tik vieną kartą, kitaip sakant, reikia užtikrinti, kad algoritmas

nepriskirtų vienam objektui keleto sekimo identifikatorių, o išlaikytų esamą. Identifikatoriaus praradimas padidintų neteisingai atspėtų (FP) ir visai neatpažintų objektų (FN) skaičių.

Kadangi objektų aptikimo dalis taip pat atlikta šiame darbe bei turi įtakos spartai ir sekimo kokybei, naudinga turėti ir objekto atpažinimo metriką. Čia galima pasinaudoti objektų atpažinimo iškeltais reikalavimais ir darbe naudoti iš pastarųjų parametrų (TP, FP ir FN) išvestą dydį – suvidurkintą aptikimo tikslumą (angl. *mean average precision*, mAP). Siekiama, kad etalono ir matavimo stačiakampių sankirtos ir sąjungos santykis (angl. *intersection over union*, IOU) dydis būtų kuo artimesnis vienetui (tai reikštų, kad spėjiojamas ir tikras stačiakampis atitinka idealiai), kuo labiau nesutampa šie stačiakampiai, tuo mažesnis šis dydis ir tuo prasčiau vertinamas tikslumas. Paprastai laikoma, kad objektą apibūdinantis stačiakampis atspėtas teisingai, jeigu sankirtos ir sąjungos sankirtos santykis yra ne mažesnis už 0,5 arba 50 %. Vėliau šis dydis panaudojamas rasti vidutinį tikslumą, o išvedus vidurkį iš visų klasių gaunamas suvidurkintas vidutinis tikslumas. Kadangi magistriniame darbe klasė tik viena, šie dydžiai sutampa. Tokio dydžio skaičiavimo pseudokodas:

```
Iteravimas per visus paveikslėlius
if IOU > slenkstis
    if objektas dar neaptiktas
        TP++
    else
        FP++
else
    FP++

if joks spėjimas neatliktas
    FN++
```

$VT = TP / (TP + FP)$

Šiame kode VT – vidutinis tikslumas (mAP), TP – tikslus teigiamas spėjimas klasei, FP – netikslus teigiamas spėjimas klasei. FN – neteisingas neigiamas spėjimas klasei..

Panašus į suvidurkintą vidutinį tikslumą parametras, tik apibūdinantis sekimo kokybę, yra daugelio objektų sekimo tikslumas (angl. *multiple object tracking precision*, MOTP)(Milan *et al.* 2016: 1–12; Bernardin *et al.* 2006: 91). Jis vertina, kokia vidutiniška objekto vietos nustatymo (lokalizavimo) paklaida:

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t},$$

čia c_t apibūdina teisingai atspėtų objektų skaičių kadre, $d_{i,t}$ apibrėžia, koks persidengimas yra tarp spėjamo objektą apibrėžiančio stačiakampio ir realaus, pateikto duomenyse.

Daugelio objektų sekimo taiklumas (angl. *multiple object tracking accuracy*, MOTA) pastaruoju metu vadinamas labiausiai paplitusiu sekimo įvertinimo parametru. Jis įvertina trijų rūšių

klaidas ir apskaičiuojamas toliau pateikiama formule (Milan *et al.* 2016: 1–12; Bernardin *et al.* 2006: 91):

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t},$$

čia t – kadro numeris, GT_t – visų atpažintinų objektų skaičius, $IDSW_t$ – sekimo identifikatoriaus numerio pasikeitimų skaičius. Tikėtina, kad klaidų bus daugiau nei atpažintinų objektų, todėl laikoma, procentai šis parametras gali patekti į intervalą $(-\infty, 100]$. Tokio parametro privalumas – gebėjimas grubiai apibūdinti sekimo kokybę, nes įvertinamos įvairių rūšių klaidos bei naudojami santykiniai vienetai.

Iš aptartų parametrų galime spręsti, kad sekimo kokybę darbe tinkamiausia apibūdinti atpažinimo ir sekimo sparta kadrais per sekundę, taip pat įvertinti aptikimo kokybę suvidurkintu vidutiniu tikslumu (mAP), apskaičiuoti vidutinę objekto lokalizavimo tikslumą (MOTP) ir daugelio objektų sekimo taiklumą (MOTA). Pastarieji du parametrai naudojami ir Multiple Object Tracking Benchmark (Milan *et al.* 2016: 1–12) duomenims vertinti, kurie bus naudojami testavimo metu objektų sekimo kokybei tirti.

3.2. Duomenų paruošimas tinklo mokymui

Nors dauguma naujausių mašininio mokymo modelių prieinami nemokamai, paprastai jie būna tik generalizuotos modelio galimybių demonstracijos ir tikrasis darbas prasideda siekiant šį modelį suderinti ir pritaikyti kokiai nors naujai užduočiai. Tam, kad užduotis galėtų būti įvykdyta, labai svarbūs mokymo duomenys. Duomenų rinkimas ir apdorojimas paprastai užima daug laiko, laimei, tam tikroms užduotims, kaip pėsčiųjų aptikimas vaizde, sukurta didelių duomenų rinkinių, kuriais galima naudotis be apribojimų. Duomenų rinkinio pasirinkimo ir pirminio duomenų apdorojimo aprašymas yra šio skyriaus tikslas.

3.2.1. Duomenų rinkinio pasirinkimas

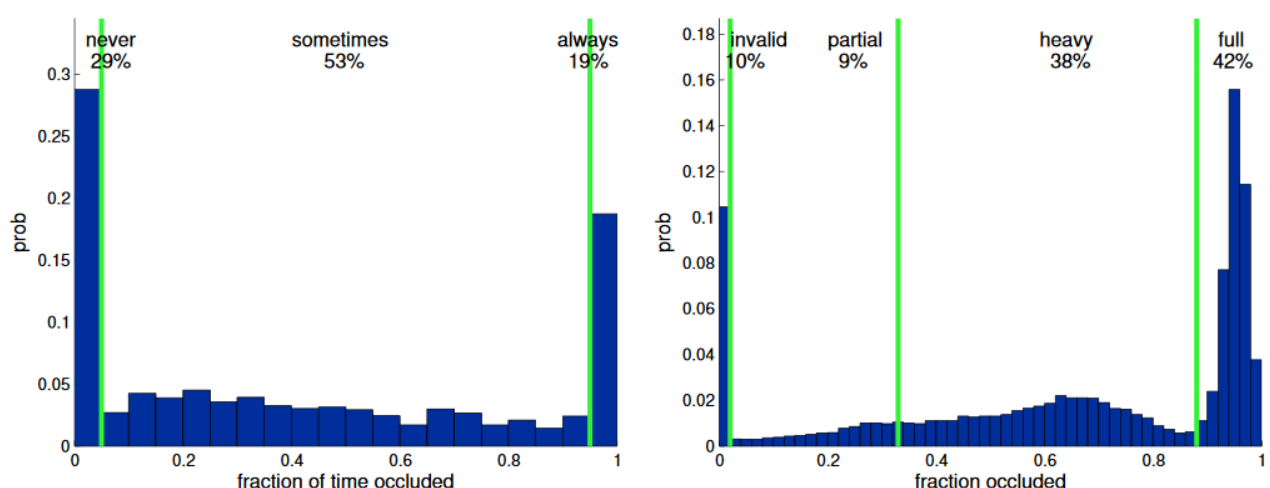
Kaip jau kalbėta skyriaus įžangoje, duomenų pasirinkimas yra viena svarbiausių dalių galutinėje sistemoje. Prastas duomenų pasirinkimas gali tapti viena iš priežasčių, kodėl gauti rezultatai yra nepatenkinami.

Vienas iš svarbiausių reikalavimų duomenų rinkiniui – duomenų įvairovė. Mašininio mokymo rezultatai priklauso nuo to, kokie duomenys jam pateikiami. Pavyzdžiui, jeigu mokymo duomenyse pateikiami tik žmogaus atvaizdai iš priekio, tikriausiai nebus išmokta atpažinti žmogaus iš šono arba nugaros, jeigu bus pateikiamas tik vieno žmogaus atvaizdas, nebus jokių objektų, užstojančių žmogų, jeigu nebus objektų saveikos, mažai tikėtina, kad algoritmas bus atsparus tokiomis sąlygomis. Lygiai taip pat svarbus ir objekto dydis vaizde. Jeigu visi vaizdai vienodo dydžio,

išmokstami būtent tam tikri bruožai, tam tikri vaizdo taškų sąryšiai, kurie nebūtinai tinkami skirtingų dydžių vaizdams. Algoritmai gali neveikti tinkamai ir tada, jeigu mokymo duomenys surenkami esant vienokiam apšvietimui, o naudojant sistemą objektą siekama aptikti šviesesnėje arba tamsesnėje aplinkoje.

Giluminiuose dirbtinių neuronų tinkluose svarbus ir didelis duomenų kiekis apskritai. Paprastai šie tinklai turi gausybę parametrų. Tai reiškia, kad esant nedideliui duomenų skaičiui bus labai gerai prisitaikyta prie mokymo duomenų, tačiau atliekant testus rezultatai nebus ekvivalentiški (angl. *overfitting*). Dažnai parametrų būna tiek daug, kad mokymui reikia milijonų paveikslukų. Tiesa, toks kiekis nėra būtinas, jeigu iš esamų duomenų generuojami papildomi duomenys (angl. *data augmentation*) pridedant triukšmo ar atliekant įvairias transformacijas, taip pat tada, jeigu sistema mokoma ne visa, o tik tam tikra jos dalis, o likusi dalis iš esmės veikia kaip bruožų išskyrimo blokas. Duomenų trūkumo problemą spręsti padeda ir įvairūs regularizacijos metodai. Tačiau net ir nepaisant šių funkcijų, kurias pasiūlo naujausi mašininio mokymo modeliai, reikia pakankamai daug duomenų. Tai atrodo suprantama jau dėl minėtų priežasčių – daugeliu atveju siekiami atpažinti objektai yra labai įvairūs nepaisant priklausymo vienai klasei. Galima tvirtinti, kad norint pasiekti pakankamai gerų rezultatų reikia turėti dešimtis tūkstančių norimų rasti objektų (pėsčiųjų) pavyzdžių.

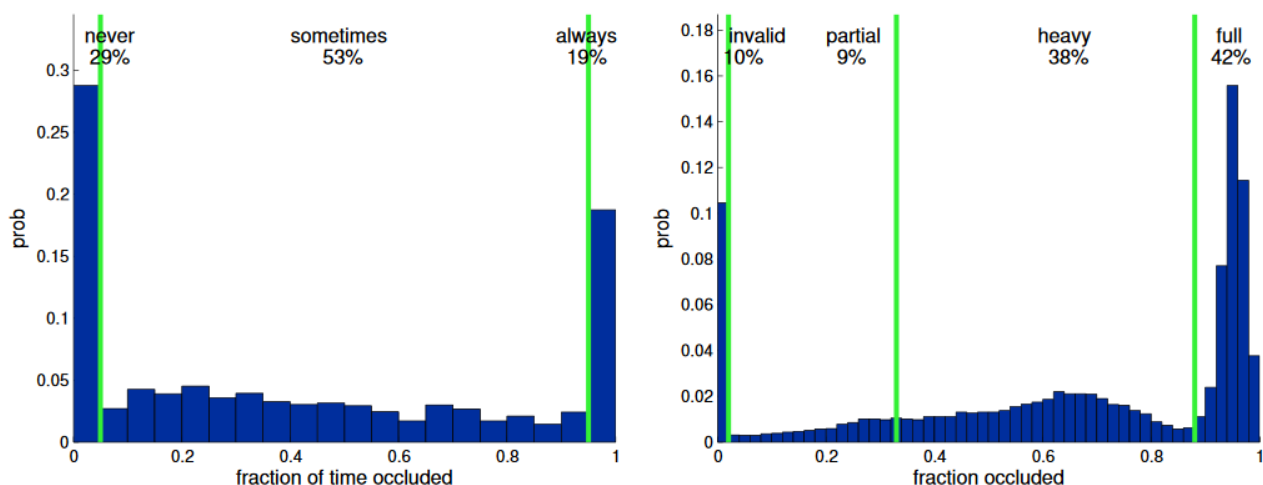
Įvardinus reikalavimus duomenų rinkiniui galima jį kurti arba pasirinkti iš platinamų laisvai. Kadangi reikalingas duomenų pavyzdžių kiekis yra didelis, rinkti duomenis pačiam bei vėliau juos sužymėti užtruktų ilgai, o baigiamojo darbo tikslas nėra duomenų analizė, pasirinkta naudoti jau sukurta apibrėžtus reikalavimus atitinkanti duomenų rinkinį. Pasirinktas duomenų rinkinys – *Caltech Pedestrian Dataset* (Dollár et al. 2009: 304–311; Dollár et al. 2012: 743–761).



3.2 pav. Objekto aukščio vaizde pasiskirstymas

Visą duomenų rinkinį sudaro net 10 valandų filmuotos, 640×480 vaizdinės skyros medžiagos. Per vieną sekundę surinkti 30 kadrų. Visas duomenų rinkinys yra sudalintas į 137-is maždaug

minutės ilgio segmentus, o visą duomenų rinkinį sudaro net 250 tūkst. paveikslų, kuriuose pažymėta 350 tūkst. pėsčiųjų, iš kurių unikalūs 2300. Taigi šis duomenų rinkinys su kaupu turėtų atitikti dydžio reikalavimus. Visgi magistro darbo metu nuspręsta naudoti ne visą duomenų rinkinį, atsižvelgiant į įrangos mokymui techninį ribotumą.



3.3 pav. Objektų užtvaros statistika

Duomenys taip pat pakankamai įvairūs. Maždaug 50 % kadrų neturi nė vieno pėsčiojo, 30 % pastebimi daugiau nei du. Vienas pėstysis vaizdo įrašė matomas vidutiniškai penkias sekundes. Pėsčiojo dydis vaizde taip pat įvairus, o pačių kūrėjų (Dollár *et al.* 2009: 304–311) skiriamas į tris rūšis atsižvelgiant į šio aukštį vaizde pikseliais – artimas objektas (>80 pikselių), vidutinio lauko (30–80 pikselių), tolimas objektas (30 pikselių ir mažiau). Pėsčiųjų vaizde aukščio pasiskirstymas pateiktas 3.2 paveiksle. Galima matyti, kad šis išsibarstymas iš esmės atitinka Gauso kreivę. 3.3 paveiksle pateikiama statistika, kaip dažnai ir kaip smarkiai vienas objektas užtveria kitą (angl. *occlusion*). Galima spėti, kad dėl tokios statistikos algoritmas bus atsparesnis dviejų objektų susidūrimo atveju.

Paskutinis šio duomenų rinkinio plusas – platus naudojimas moksliniuose tyrimuose pastarąjį dešimtmetį, todėl nesudėtinga palyginti nagrinėjamo mašininio mokymo modelio rezultatus su kitų ir padaryti išvadas, ar ir kaip šis yra pranašesnis už kitus modelius.

Apibendrinant galima teigti, kad duomenys turi atitikti tokius reikalavimus:

- paveikslų mokymui kiekis turi būti dešimties tūkstančių eilės;
- objektai vaizde turi būti įvairaus dydžio, apšvietimo, tam tikroje dalyje vaizdų turi sąveikauti vienas su kitu.

3.2.2. Pirminis duomenų apdorojimas

Duomenų radimas yra tik pirmoji duomenų paruošimo dalis. Jie gali būti pateikti įvairiais formatais, todėl gali būti nesuvokiami mokymo algoritmui. Tokiu atveju duomenis pakeisti į kitokį

formatą. Paprastai konvertavimas atliekamas ne patiems vaizdams, paprastai saugomiems JPEG formatu, o anotacijoms – failams, kuriuose pažymėti stačiakampių, žyminčių objektus, duomenys bei klasės, kuriai objektas priklauso. Toks duomenų žymėjimas leidžia panaudoti tuos pačius paveikslėlius atliekant mokymą tik kokiai nors klasei, jeigu, aišku, paveikslėliuose galima rasti pakankamai mokymui reikalingų objektų.

Pageidaujamas formatas – tam tikrame aplanke sukelti vaizdai su anotacijomis. Kaip jau minėta, vaizdai paprasta būna JPEG formato, bet gali būti GIF, PNG ar bet kokio kito, kurį palaiko Python programavimo biblioteka *Pillow*, skirta vaizdų apdorojimui. Anotacijose kiekvienam kadrai aprašyti skiriamas vienas failas, o šiame faile kiekvienam objektui skiriama eilutė su penkiais parametrais. Pirmasis skaičius – klasė, užkoduota skaičiumi nuo 0 iki $n-1$, kur n – klasių skaičius. Kadangi domina tik viena klasė, šioje vietoje visada įrašomas nulis. Toliau pateikiamos objekto centro koordinatės bei objekto plotis ir aukštis ($\langle \text{klasė} \rangle \langle x \rangle \langle y \rangle \langle \text{plotis} \rangle \langle \text{aukštis} \rangle$).

Caltech duomenų rinkinyje pateikiami *.seq* ir *.vbb* plėtinius turintys failai. Pirmuosiuose sudėta informacija apie vaizdus, antruosiuose – šių vaizdų anotacijos. Nors pirmuoju atveju toks formatas atrodo gan nesuprantamas ir sudėtingas, nereikalauja didelių skaičiavimų įvairioms konversijoms – juose jau sudėti vaizdo failai su papildoma tarnybine informacija. Tuo tarpu *.vbb* failai atitinka MATLAB skirtą anotacijų formatą, kurio konvertavimui tinka įvairūs tam specialiai sukurti įrankiai *Python* programavimo kalba.

Galiausiai esamus vaizdus reikia išdalinti į dalis, vieną iš jų skiriant mokymui, kitą validavimui, jeigu pageidaujama, trečiąją dalį galima paskirti testavimui. Paprastai laikoma, kad validavimui ir testavimui skiriama bent po 20 procentų duomenų pavyzdžių. Tam į atskirus tekstinius failus (pavadintais kad ir *train.txt* ir *test.txt*) pakanka sukelti pilną nuorodą į kiekvieno failo buvimo vietą diske (pavyzdžiui, */home/arturas/Downloads/train/paveiksliukas.jpeg* Linux operacinėje sistemoje). Visgi tam, kad būtų galima tinkamai atlikti pačio sekimo algoritmo testavimą, pravartu panaudoti visai nesusijusius vaizdus. Tam panaudojami *Multiple Object Detection Benchmark* (2D MOT-15) pateikiami vaizdo failai (Milan *et al.* 2016: 1–12), nufilmuoti įvairiose realaus pasaulio situacijose.

3.3. Optimizuojama sistema ir jos parametrų įvertinimas

Kaip jau minėta, Kalmano filtras skirtas prognozuoti, šiuo atveju – objekto vietą vaizde. Kadangi algoritmai netobuli, kartais objektas visai nerandamas, o net ir gautas rezultatas paprastai turi aukšto dažnio triukšmo bei gali skirtis tarp dviejų kadru, reikalingas algoritmas, kuris šį triukšmą pašalintų. Tai jis atlieka prognozuodamas objekto vietą kitame kadre. Tam, kad būtų ką sekti, svarbu pasirinkti tinkamą vaizdo aptikimo algoritmą. Skirtingi pasirinkimai lems skirtingą

tikslumą ir greitį. Tikslumą ir greitį taip pat lemia tinkamos Kalmano filtro modifikacijos pasirinkimas.

Tam, kad būtų galima palyginti, ar pakeitimai davė apčiuopiamos naudos, reikalingas etalonas, kuris įvertintas 3.1 skyriuje apibūdintomis metrikomis. Tokiu atveju kiekvienas algoritmo pakeitimas gali būti įvertintas kiek įmanoma labiau apibrėžtai.

Daugelio objektų vaizde sekimo sistemą iš esmės galime sudaryti iš trijų dalių – objekto aptikimo (detektoriaus), filtro, dinamiškai atnaujinančio sekimo koordinatės, bei sekimo koordinačių priskyrimo atitinkamiems objektams. Kiekvieną iš šių trijų dalių gali sudaryti skirtingi algoritmai, kartu leidžiantys visai sistemai pasiekti norimą rezultatą. Visgi optimizuojamai sistema pasirinkti būtent tam tikri trys. Detekcijos uždaviniui naudojamas kiek anksčiau aptartas YOLOv2 mašininio mokymo modelis (Redmon *et al.* 2016; Redmon *et al.* 2015), sekimui naudojamas taip pat jau anksčiau išplėstinai aptartas Kalmano filtras, o priskyrimo problemą sprendžia Kuhno-Munkreso algoritmas (skaičiavimų sudėtingumas $O(n^3)$), paprasčiau žinomas vengriško algoritmo vardu (Munkres 1957: 32–38). Taigi detektorius nurodo, kokiose koordinatėse rastas objektas, Kalmano filtras užtikrina, kad tarp kadrų atrastas objektas būtų atsparesnis aukšto dažnio triukšmui, o vengriškas algoritmas stengiasi užtikrinti, kad tam tikras objektas išlaikytų jam nuo pradžių priskirtą identifikatorių. Tai šis atlieka sprendžiamas priskyrimo užduotį – kiekvienas objektas gauna identifikatorių pagal tai, kiek arti yra atpažintas objektas dviejuose skirtinguose kadruose. Mažai tikėtina, jog objektas taip greitai pakeis savo poziciją, kad keli objektai susikeis vietomis. Kitaip sakant, tas pats identifikatorius priskiriamas objektui, kurio pozicija per du kadrus pasikeitė mažiausiai.

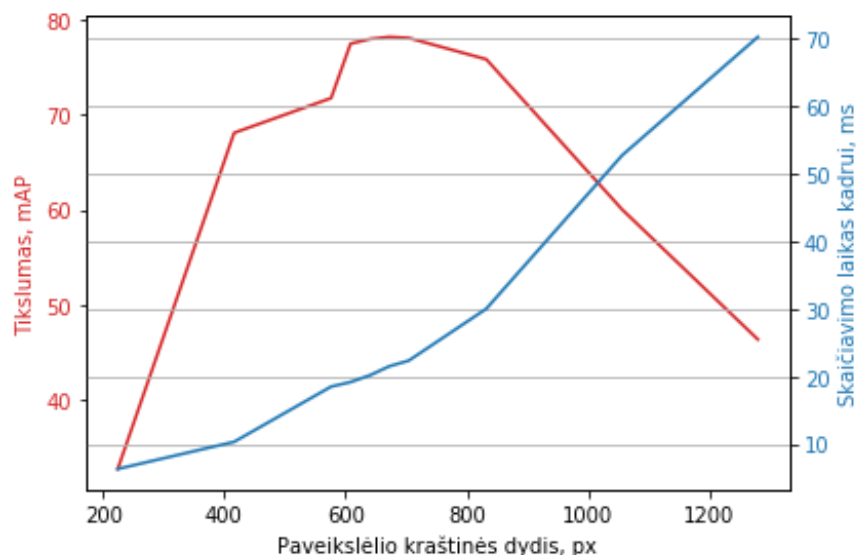
Kalmano filtro ir sekimo koordinačių priskyrimo algoritmo derinys (pavadintas *SORT: Simple Online Realtime Tracking*) neseniai aptartas ir parodė aukštus rezultatus (Bewley *et al.* 2016: 3464–3468). Šio derinio privalumas yra aukšta sparta – sekimas neapkraunamas bandymais spręsti kraštutinius atvejus, kurie sulėtina algoritmą, nors padidina jo atsparumą klaidoms. Laikoma, kad prisitaikymas prie ribinių atvejų nebeteri prasmės, nes vis gerėja atpažinimo algoritmų kokybė bei jų sparta. Tam, kad objektas būtų atsparesnis triukšmui, šio sekimo rezultatai saugomi dar kurį laiką prieš juos ištrinant.

Python programavimo kalba parašytas SORT algoritmas platinamas nemokamai, tad norint gauti pradinius rezultatus, kuriuos būtų galima naudoti palyginimui, reikia apmokėti atpažinimo algoritmą savais duomenimis. Tam panaudota *Caltech Pedestrian Dataset* duomenų bazė. Mokymas nutraukiamas po 45 tūkst. iteracijų, atrenkamas geriausias rezultatą validavimo duomenims grąžinantys svoriai. Mokymas atliekamas 416×416 dydžio vaizdams, po to paveikslai testams padidinami iki 608×608 . Tai leidžia geriau aptikti mažus objektus ir pasiekti aukštesnį

aptikimo rezultata. Pėsčiųjų aptikimo algoritmo tikslumas 416×416 vaizdams – 68,05 %, 608×608 vaizdams – 77,81 %.

Objektų aptikimo algoritmas, parenkant didesnius vaizdus įėjime, leidžia kokybiškiau aptikti objektus vaizde. Tam, kad būtų galima įvertinti, ar toks pakeitimas vertas papildomo skaičiavimų sudėtingumo, reikia nubrėžti laiko vienam vaizdo kadrai apdoroti priklausomybę nuo įėjimo vaizdo dydžio (3.4 paveikslas). Tam panaudojami validavimo duomenys. Svarbu paminėti tai, jog algoritmui pateikiami vaizdai gali būti daug didesni – visi vaizdai sumažinami ir apkarpomi taip, kad algoritmo įėjime būtų kvadratiniai ir tam tikro nustatyto dydžio. Be abejo, didelių vaizdų paruošimas algoritmui gali turėti įtakos objektų aptikimo spartai bei kokybei.

Iš 3.4 paveikslo galima pastebėti, kad sprendimas šiek tiek kiltelti įėjimo vaizdo dydį yra pagrįstas. Tolimesnis vaizdo didinimas nebetenka prasmės – prastėja objekto aptikimo kokybė ir ilgėja tam reikalingas laiko tarpas. Tam įtakos gali turėti tai, kad mokymui naudoti duomenys nėra HD raiškos, todėl algoritmui daug sunkiau atlikti skaičiavimus su dideliais vaizdais objektų aptikimo algoritmo įėjime.



3.4 pav. Laiko vienam vaizdui apdoroti priklausomybę nuo įėjimo vaizdo dydžio

Paskutinis uždavinys, kurį reikia atlikti prieš įvertinant galimybes spartinti objektų sekimo vaizde algoritmą, yra sekimą apibūdinančių metrikų fiksavimas, bei kiekvienai sistemos daliai vykdyti reikalingas laikas. Pastarasis apibūdinimas padės tinkamai pasirinkti optimizuotinus algoritmo elementus. Kaip minėta, sekimo įvertinimui naudoti *Multiple Object Tracking Benchmark* (2D MOT15) duomenys. Testavimo rezultatai pateikti 3.1 lentelėje. *Recall* – etalono padengimo matavimu tikslumas, *Precision* – matavimo panaudojimo etalonui padengti tikslumas, FPS – kadrų per sekundę skaičius (angl. *Frames Per Second*).

3.1 lentelė. Pradinio algoritmo įvertinimo rezultatai

Vaizdo įrašo pavadinimas	Kadro raiška	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	<i>Recall</i>	<i>Precision</i>	MOTA	MOTP
PETS09-S2L1	768 × 576	15,4	733	15,1	25,7	71,2	13,2	66,1
TUD-Campus	640 × 480	16	707	15,6	57,4	81,2	42,6	64,4
TUD-Stadtmitte	640 × 480	16,3	612	15,9	67,3	94,3	62,1	63
ETH-Sunnyday	640 × 480	16,2	539	15,7	58,6	76,5	38,3	71,7
ETH-Pedcross2	640 × 480	16,3	838	16	23,1	92,7	20,3	73,5
KITTI-17	1224 × 370	20,5	538	19,8	41,9	57,3	6,9	67,7
ADL-Rundle-6	1920 × 1080	10,3	484	10,1	45,3	78	30,7	70,9
Venice-2	1920 × 1080	10,3	424	10,1	34,3	60	10,2	69
Iš viso:		15,2	609	14,8	44,2	76,4	28	68,3

Iš 3.1 lentelės galima matyti, kad didžiausią skaičiavimų dalį algoritmas skiria būtent objekto aptikimui, o skirtumas tarp jo ir kitų dalių kartų sudėjus (kartu vykdomas objekto sekimas ir unikalios identifikatoriaus priskyrimas) matuojamas keliomis dešimtimis kartų. Galima daryti išvadą, kad objekto sekimo algoritmo spartinimas (atmetant objekto aptikimą) turi prasmės tik tada, jeigu jo laikas sutrumpėja bent keliais kartais.

Taip pat iš 3.1 lentelės galima matyti, kad gana aukšta sekimo kokybė pasiekama nedideliems vaizdams (640 × 480). Prastesnė kokybė gaunama sekant objektus dideliuose vaizdo kadruose arba tokiuose, kur viena vaizdo kadro kraštinė daug ilgesnė už kitą (KITTI-17). Taip pat galima matyti (atsižvelgus į MOTP), kad tarp teisingai atspėtų stačiakampių ir tikrųjų objekto koordinatų paklaida nėra didesnė ir svyruoja apie 70 %, tačiau pasitaiko gana didelis klaidų skaičius, kurį įvertina MOTA parametras.

3.4. Galimybės paspartinti esamą algoritmą

Kadangi, kaip minėta, algoritmą sudaro trys dalys, pasiekti spartesnį rezultatą galima stengiantis keisti bet kurią iš šių dalių. Be abejo, svarbu įvertinti, kuri dalis užima daugiausia laiko

ir stengtis optimizuoti būtent ją. Iš jau atliktos algoritmo analizės galima matyti, kad lėčiausiai pateikiami objekto aptikimo rezultatai.

Atvaizdavimas kompiuterio ekrane vyksta realiu laiku, kai per sekundę apdorojami bent 25 kadrai. Toks kadrų skaičius gali atrodyti didelis, juk informacija tarp kadrų paprastai kinta nežymiai, todėl tikėtina, kad atliekant objekto aptikimą ne kiekviename kadre galima sutaupyti nemažai skaičiavimų, reikalingų būtent objekto aptikimui. Panašią idėją turėjo ir *Nvidia* kompanijos, tyrėjai, vystantys bepiločio automobilio valdymo mašininio mokymo modelį (Bojarski *et al.* 2016: 1–9). Tiesa, jie tokį metodą taikė mokymo duomenims, tačiau atsižvelgiant į tai, kad šiuo atveju svarbiau algoritmo sparta nei atsparumas, toks metodas gali būti vertas susidomėjimo.

Objektams vaizde aptikti naudojamų dirbtinių neuronų tinklų įėjime paprastai paduodami nedideli, paprastai kvadratiniai vaizdai. Esamo algoritmo (YOLOv2) įėjime paduodami 608×608 raiškos vaizdai. Visi kiti vaizdai yra prieš tai sumažinami iki šio dydžio. Kadangi dideliuose vaizduose bet koku atveju šiuos mažinant išmetama daug informacijos, galima spėti, kad įėjime iš karto užtikrinant mažesnius vaizdus neturėtų būti prarasta daug aptikimo tikslumo, tačiau galima gauti daug spartesnį algoritmą. Pavyzdžiui, du kartus sumažintas vaizdas atmintyje užima keturis kartus mažiau vietos, tad tikėtina, kad paspartėjimo priklausomybė keičiant kadro dydį būtų kvadratinė.

Viena iš problemų siekiant sparčiai įvertinti algoritmą – didelis kiekis parametrų, kurių kiekvienas paprastai užima po 32 bitus. Pavyzdžiui, YOLOv2 atveju parametrams saugoti naudojama net 256 MB atminties. Tiesa, tai nėra daug, jeigu lygintume su pirmąja algoritmo versija, kurios parametrams saugoti naudota net 1 GB atminties (Redmon *et al.* 2015). Jeigu vietoj 32-jų bitų slankaus kablelio (angl. *float*) saugotume 16-os bitų slankaus kablelių skaičius, būtų sutaupyta ne tik parametrų saugojimui skirtos vietos, bet potencialiai ir padidėtų objektų aptikimo sparta (16-os bitų matematika vaizdams paskutinėse GPI ir CPI atliekama sparčiau nei 32-jų bitų). Tiesa, nėra iki galo aišku, kokie rezultatai pasiekiami sumažinus neuronų tinklo koeficientų tikslumą. Visgi pastarųjų metų rezultatai (Park *et al.* 2017: 5456–5464; Hubara *et al.* 2016: 1–29), kuriuose neuronų tinklai mokosi itin mažo tikslumo skaičiais, bei tai, kad ši tema nagrinėjama jau gana seniai (Lundin *et al.*), leidžia tikėtis, kad kvantavimas leistų sutaupyti skaičiavimų prarandant sąlyginai nedaug tikslumo.

Parametrų užimamos atminties problemą galima spręsti ne vien neuronų tinklo svorių kvantavimu. Galima bandyti tiesiog sumažinti parametrų skaičių keičiant dirbtinių neuronų tinklo architektūrą – pakeičiant atitinkamus konvoliucinius sluoksnius kitais arba jų visai atsisakant. Nedidelis objektų aptikimo tikslumo sumažėjimas galimai neturėtų didelės reikšmės sekimo stabilumui, tačiau galėtų pridėti daug spartos algoritmui. Be abejo, svarbu pasirinkti tokius sluoksnius, kuriuose mokomas kuo didesnis svorių kiekis. Konvoliucinių sluoksnių kitimas aktyviai

naudojamas siekiant sukurti nedidelius, telpančius į mobiliuosius įrenginius, tačiau tik nežymiai prasčiau veikiančius konvoliucinius neuronų tinklus, tokius, kaip *SqueezeNet* (Iandola *et al.* 2016: 1–13). Šio straipsnio kūrėjai siūlo tris strategijas – keisti konvoliucijos (sąsukos) operacijų branduolius iš 3×3 į 1×1 ir taip naudoti devynis kartus mažiau parametrų, taip pat mažinti konvoliucijų įėjimo dydį panaudojant jų pasiūlytus *fire* modulius bei mažinti tinklo sluoksnius tik pačiame neuronų tinklo gale, kad paskutiniuose (aktyvacijos) sluoksniuose (angl. *activation maps*) būtų pakankamai parametrų tam, jog tinklas neprarastų tikslumo. Viena arba kelios strategijos gali būti panaudotos ir siekiant optimizuoti jau esamo algoritmo objektų aptikimo architektūrą.

4. EKSPERIMENTINIO TYRIMO PLANAS IR REZULTATAI

Šiame skyriuje atliekami eksperimentiniai tyrimai bei algoritmo derinimas ir išsiaiškinama, ar siūlomi sekimo algoritmo pakeitimai leidžia taip pat kokybiškai ar net kokybiškiau atlikti objekto sekimą. Tam, kad būtų patogų sekti kiekvieną iš siūlymų, atskirai tyrimo daliai paskiriamas atskiras poskyris, kuris baigiamas gautų rezultatų įvertinimu trumpomis išvadomis. Kiekviename skyriuje pateikiamos lentelės ir grafikai, leidžiantys įvertinti kiekybiškai algoritmo spartinimą bei sekimo kokybės pokyčius, lyginant su etaloniniais rezultatais, aptartais praeitame skyriuje.

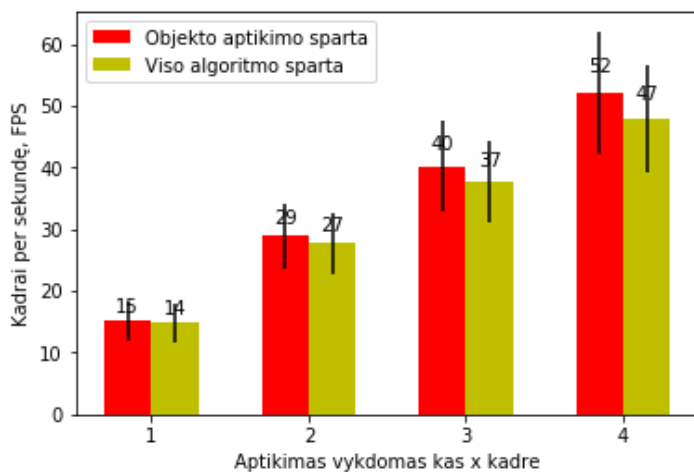
4.1. Objektų sekimo vaizde spartinimas naudojant objektų aptikimą tik kai kuriuose kadruose

Kadangi daugiausia skaičiavimų reikia skirti objektams aptikti, pasirinkta optimizuoti būtent šią dalį. Manoma, kad objektų paieška tik kas kelintame kadre leis be didelių pakeitimų sutaupyti skaičiavimų. Tuose kadruose, kuriuose aptikimas praleidžiamas, naudojami paskutinio objektų aptikimo rezultatai. Kadangi aptikimui skiriama keliomis dešimt kartų daugiau laiko nei sekimui, tikėtina, kad atliekant objektą tik kas antram kadrai galima pasiekti maždaug du kartus didesnę spartą. Iš viso eksperimentai atlikti atmetant kas antrą, trečią ar ketvirtą kadrus. Parametrų pasikeitimai lyginant su pradiniu algoritmu pateikiami 4.1 lentelėje, o suvidurkinti parametrai bei jų priklausomybė nuo aptikimo algoritmo pritaikymo dažnumo pateikiami 4.1 ir 4.2 paveiksluose.

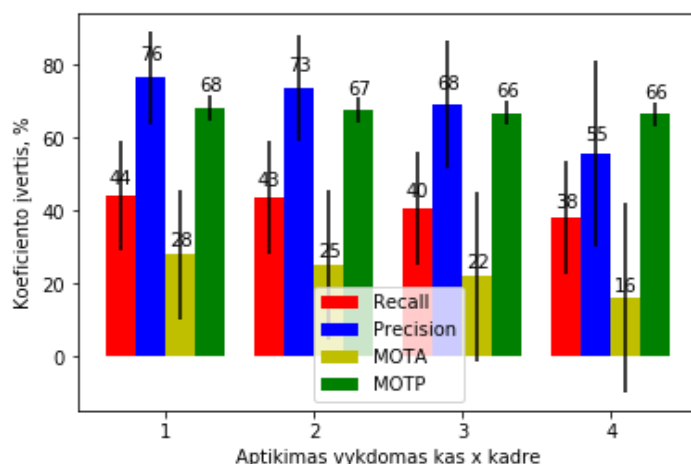
4.1 lentelė. Daugelio objektų sekimo rezultatai aptikimą vykdant tik kas kelintame kadre

Vaizdo įrašo pavadinimas	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	<i>Recall</i>	<i>Precision</i>	MOTA	MOTP
Aptikimas vykdomas kas antrame kadre							
PETS09-S2L1	29,7	756	28,5	24	65,5	9,2	65,3
TUD-Campus	30,6	742	29,4	55,7	79,7	40,1	64,2
TUD-Stadtmitte	30,8	621	29,3	68,3	94,6	63	62,5
ETH-Sunnyday	30,5	573	29	59,1	73,8	35,8	71,2
ETH-Pedcross2	30,8	875	29,7	23,4	89,9	19,6	72,5
KITTI-17	37,8	560	35,4	38,4	48,6	-6,1	65,7
ADL-Rundle-6	20,8	526	20	46,1	78	31,3	70,1
Venice-2	21	463	20,1	34,4	59,2	9,3	69,4
Iš viso:	29	630	27,8	43,7	73,7	25,3	67,6
Aptikimas vykdomas kas trečiame kadre							
PETS09-S2L1	42,5	733	40,2	19,7	56,9	2,6	64

TUD-Campus	44,2	746	41,7	56,3	79,2	40,4	63,2
TUD-Stadtmitte	44,2	616	41,3	58,8	94,6	63,8	62,2
ETH-Sunnyday	42,7	529	39,5	59,4	59,1	29,6	70,5
ETH-Pedcross2	41,4	812	39,4	23,4	88,7	19,5	71,5
KITTI-17	50	548	45,8	26,6	39,1	-17,9	65,2
ADL-Rundle-6	28,3	517	26,8	45,6	75,9	29,4	69,5
Venice-2	28,2	442	26,5	34,9	58	9,3	69
Iš viso:	40,2	618	37,7	40,6	68,9	22,1	66,9
Aptikimas vykdomas kas ketvirtame kadre							
PETS09-S2L1	54,3	716	50,5	14,5	41,6	-7,5	64
TUD-Campus	56,7	677	52,3	54,6	74,5	34,5	63,2
TUD-Stadtmitte	56,2	584	51,3	57,6	92	60,7	62
ETH-Sunnyday	55,3	547	50,2	52,7	64,3	21,3	70,2
ETH-Pedcross2	56,1	857	52,7	22,4	8,1	16	70,8
KITTI-17	66,4	538	59,1	22,9	30,7	-31,5	64,8
ADL-Rundle-6	36,1	514	33,7	45,4	74,6	28,1	69,4
Venice-2	36,1	450	33,5	34,6	57,6	7,8	68,4
Iš viso:	52,2	610	47,9	38,1	55,4	16,2	66,6



4.1 pav. Apdorojimo spartos palyginimas, kai aptikimui naudojamas kas x kadras



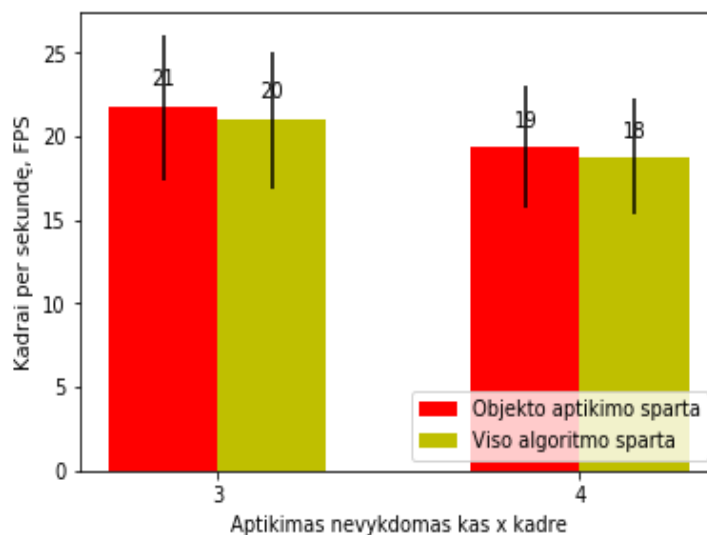
4.2 pav. Daugelio objektų sekimo parametrų palyginimas, kai aptikimui naudojamas kas x kadras

Iš pateiktų 4.1 ir 4.2 paveikslų ir 4.1 lentelės galima daryti išvadą, kad kaip ir tikėtasi, objekto aptikimą atliekant tik kas kelintame kadre vidutinis laikas vienam kadru apdoroti sumažėja. Apdorojimo spartos didėjimas yra tiesinis, kaip matoma 4.1 paveiksle. Visgi nepaisant spartos padidėjimo, galinčio viršyti tris kartus, atliekant objektų paiešką tik dalyje kadrų prarandamas tikslumas (4.2 paveikslas). Nors daugelio objektų sekimo tikslumas (MOTP) išlieka praktiškai toks pats (objektas lokalizuojamas panašiai kokybiškai), padidėja identifikavimo klaidų bei neteisingai atpažintų objektų skaičius, tai liudija vis mažėjantis daugelio objektų sekimo taiklumas (MOTA). Taip pat galima pastebėti, kad šio parametro mažėjimas yra vis spartesnis retinant aptikimo dažnį. Tikėtina, kad pasirenkant kiek kitokią kadrų atmetimo taktiką sparta galėtų kažkiek padidėti išlaikant aukštą objektų sekimo taiklumą. Galima išbandyti atvirkštinį atvejį – būtent kas kelintam kadru nevykdyti aptikimo ir pabandyti įvertinti gautus rezultatus. Tokių eksperimentų rezultatai pateikti 4.2 lentelėje ir 4.3 bei 4.4 paveiksluose.

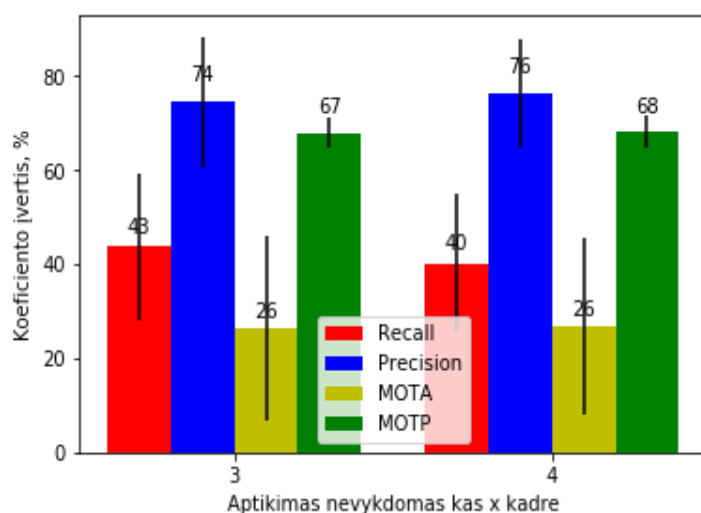
4.2 lentelė. Daugelio objektų sekimo rezultatai aptikimo nevykdant kas kelintame kadre

Vaizdo įrašo pavadinimas	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	<i>Recall</i>	<i>Precision</i>	MOTA	MOTP
Aptikimas nevykdomas kas trečiame kadre							
PETS09-S2L1	22,3	737	21,6	24,3	67	10,3	65,7
TUD-Campus	22,7	718	22,0	56,8	80,3	41,5	64,1
TUD-Stadtmitte	23,4	607	22,5	67,5	94,1	62,3	62,9
ETH-Sunnyday	23	546	22,1	59,8	75,1	37,8	71,3
ETH-Pedcross2	23,2	847	22,6	23,2	91,4	20,1	72,8
KITTI-17	29,2	556	27,7	37,6	51,1	-2	67,1

ADL-Rundle-6	15	494	14,6	45,7	77,5	30,7	70,5
Venice-2	15,2	438	14,7	34,5	59,9	10,1	69,2
Iš viso:	21,8	618	21	43,7	74,6	26,4	68
Aptikimas nevykdomas kas ketvirtame kadre							
PETS09-S2L1	19,8	729	19,3	25	68,5	11,3	65,8
TUD-Campus	20,8	725	20,2	56,3	80,5	41,2	64,7
TUD-Stadtmitte	21	625	20,3	67,5	94,4	62,3	62,7
ETH-Sunnyday	20,9	550	20,1	28,9	74,8	36,9	71,5
ETH-Pedcross2	21,1	836	20,6	23,1	91	19,8	72,9
KITTI-17	24,5	536	23,5	40,3	53,4	1,2	67,1
ADL-Rundle-6	13,2	491	12,9	45,8	77,9	31,1	70,5
Venice-2	13,9	440	13,5	34,2	59,6	9,7	69,4
Iš viso:	19,4	617	18,8	40,1	76,3	26,7	68



4.3 pav. Apdorojimo spartos palyginimas, kai objekto aptikimui nenaudojamas kas x kadras

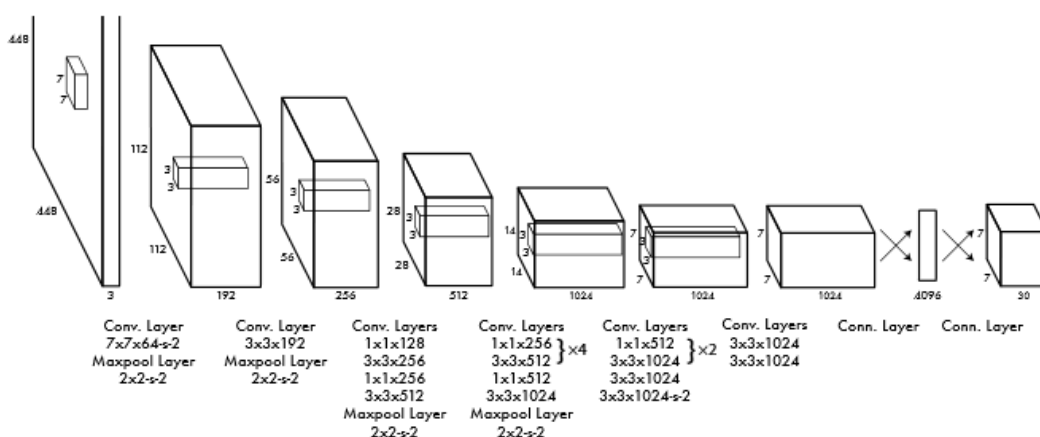


4.4 pav. Daugelio objektų sekimo parametrų palyginimas, kai objekto aptikimui nenaudojamas kas x kadras

Iš 4.2 lentelėje ir 4.3-4.4 paveiksluose pateiktų rezultatų galima matyti, kad atmetant kas tam tikrą kadrą rezultatai keičiasi nežymiai, tačiau gana smarkiai krenta apdorojimo sparta. Kritimas nėra tiesinis, tačiau tai lengvai paaiškinama tuo, kad skiriasi nenaudojamų kadrų kiekis pirmojoje ir antrojoje eksperimento dalyje. Pavyzdžiui, jeigu atmetame kas trečią kadrą, naudojame dvi trečiąsias galimų kadrų. Eksperimente praleistas kas antro kadro atmetimas, nes jis ekvivalentiškas kas antro kadro naudojimui.

Iš visų šiame poskyryje atliktų eksperimentų galima daryti išvadą, kad norint paspartinti daugelio objektų sekimą vaizde ir prarasti pakankamai nedaug tikslumo tinkamiausia aptikimo algoritmą naudoti kas antram kadrui. Nežymus parametų suprastėjimas leidžia pasiekti maždaug 89 % didesnę viso objektų sekimo algoritmo spartą.

4.2. Objektų sekimo vaizde spartinimas keičiant objektų aptikimo algoritmo architektūrą



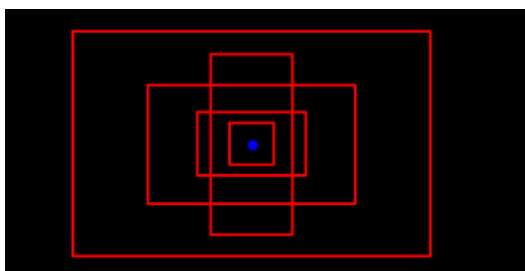
4.5 pav. Giluminio dirbtinių neuronų tinklo YOLO architektūra (Redmon *et al.* 2015)

YOLOv2 algoritmas pagal nutylėjimą sukurtas taip, kad gerai veiktų daugelio objektų, kurie priklauso ne vienai klasei, atpažinime. Toks algoritmas rodo gerus rezultatus aptikant įvairius objektus, kurie skiriasi savo spalva ar forma. Visgi magistro darbo atveju klasė yra tik viena, o pageidaujamo stačiakampio aukštis kone visada yra daug didesnis nei plotis. Tikėtina, kad algoritmo derinimas ir pačios architektūros keitimas leis tiksliau aptikti pėsčiuosius vaizde bei tai atlikti su mažiau skaičiavimų nei reikia pradiniam algoritmui.

Tam, kad būtų galima apibūdinti galimus pakeitimus, reikia nurodyti algoritmo architektūrą. Ji pateikta 4.5 paveiksle. Tiesa, čia pateikta pirmosios modelio versijos architektūra, straipsnio autorius, pateikdamas antrosios versijos rezultatus, nurodė padarytus pakeitimus, o didžioji tinklo dalis išliko tokia pati (Redmon *et al.* 2016). Toliau, siūlant pakeitimus ir pateikiant eksperimentinių tyrimų rezultatus, kurie patvirtina arba paneigia šias hipotezes, bus naudojamosi būtent šiuo paveikslu bei nurodoma, kokie pakeitimai atlikti (antrojoje versijoje) ir šiame baigiamajame darbe.

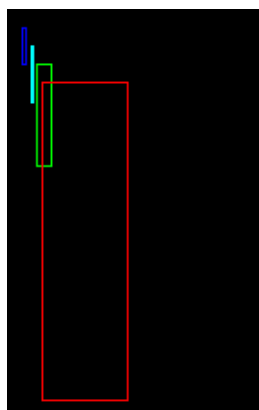
4.2.1. Objektų aptikimo algoritmo spartinimas keičiant tikslinimo stačiakampių parametrus

Pirmojoje YOLO algoritmo versijoje klases, stačiakampio, kuris apibrėžia objektą, kraštinių skaičiavimą atlikdavo pilnai sujungti neuronų sluoksniai (angl. *fully-connected layers*, žr. 4.5 paveiksle), kuriuos iš esmės galime laikyti daugiasluoksniu perceptronu. Toks perceptronas giluminiuose neuronų tinkluose nėra pageidaujamas – dėl didžiulio tarpusavio jungčių tarp neuronų skaičiaus čia saugojamas itin didelis tinklo svorių skaičius. Didelis parametrų skaičius apsunkina mokymą bei reikalauja daugiau laiko atlikti skaičiavimus su vienu vaizdo kadru, todėl antrojoje dalyje buvo atsisakyta šių sluoksnių ir jie pakeisti tikslinimo stačiakampiais (angl. *anchor boxes*). Pagal nutylėjimą išskirti penki tikslinimo stačiakampiai (pateikti 4.6 paveiksle). Jų paskirtis – padėti algoritmui atlikti objektų aptikimą, kai vienas objektas dalinai uždengtas kito. Kadangi visas vaizdas suskirstytas į stačiakampių matricą ir kiekvienam stačiakampiui siūlomi penki galimi spėjimai, skirtingos formos tikslinimo stačiakampiai padeda spėlioti skirtingos formos objektus, net jeigu vienas iš jų dalinai dengiamas kito.



4.6 pav. Pagal nutylėjimą objektų aptikimui naudojami tikslinimo stačiakampiai

Šie stačiakampiai gaunami atsižvelgiant į mokymo duomenų rinkinyje randamų etaloninių stačiakampių (angl. *ground truth*). Kadangi nežinoma, kokiam tikslinimo stačiakampiui reikėtų priskirti atitinkamą objektą, atliekamas klasterizavimas. Tam patogus paprastas k-vidurkių algoritmas. Jo esmė yra iteratyviai atnaujinti stačiakampių centrus ir koordinates bei atitinkamų objektą priskyrimą vienai ar kitai stačiakampių grupei. Skaičiavimai atliekami iki konvergavimo. Pritaikius šį algoritmą galima leisti atlikti algoritmo mokymą turint ir mažiau tikslinimo stačiakampių, pavyzdžiui, keturis (4.7 paveikslas). Tikėtina, kad naudojant šiuos tikslinimo stačiakampius algoritmas turėtų pateikti tikslesnius pėsčiųjų aptikimo rezultatus, nes atsižvelgiama, kad žmogaus aukštis yra smarkiai didesnis nei plotis.



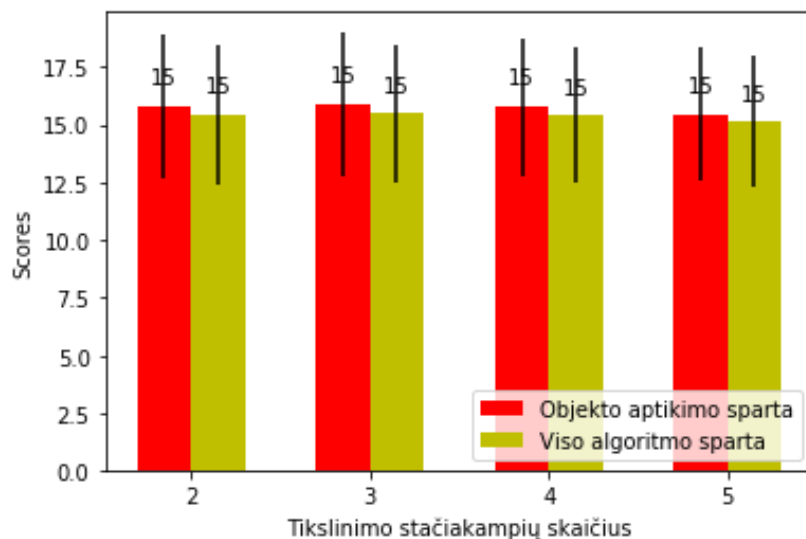
4.7 pav. K-vidurkių algoritmu gauti tikslinimo stačiakampiai, kai jų kiekis – keturi

Šiame poskyryje siekiama ištirti, kokią įtaką algoritmo veikimui turi tikslinimo stačiakampių kiekio ir formos pakeitimas. 4.3 lentelėje pateikiami algoritmo veikimo rezultatai, kai tikslinimo stačiakampių formos radimui pagal nurodytą jų kiekį atliekamas k-vidurkių algoritmo pagalba. Kadangi kiekvienu atveju šį mašininio mokymo modelį reikėjo apmokyti iš naujo, lentelėje taip pat pateikiamas ir objektų aptikimo suvidurkintas tikslumas (mAP).

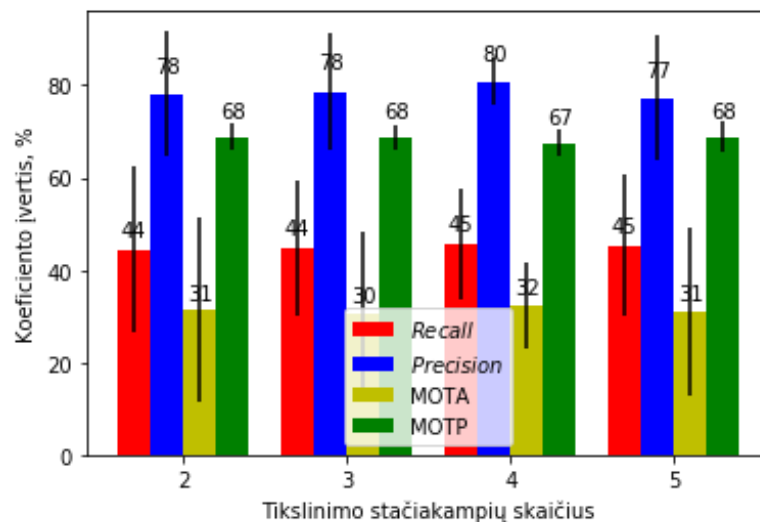
4.3 lentelė. Daugelio objektų sekimo rezultatai keičiant tikslinimo stačiakampių skaičių

Vaizdo įrašo pavadinimas	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	<i>Recall</i>	<i>Precision</i>	MOTA	MOTP
Tikslinimo stačiakampių skaičius – 5, mAP – 78,87 %							
PETS09-S2L1	15,6	757	15,3	18	46,2	-4,5	63,6
TUD-Campus	16,5	750	16,2	59,1	82,2	44,6	63,9
TUD-Stadtmitte	16,5	645	16,1	67,7	92,6	60,5	67,3
ETH-Sunnyday	16,5	636	16,1	57,3	83,4	42,9	71,8
ETH-Pedcross2	16,4	781	16,1	30,5	89,7	26	73,2
KITTI-17	20,2	747	19,7	44,2	79,7	30,4	70
ADL-Rundle-6	11	455	10,7	47,8	72,7	27,4	70,2

Venice-2	10,9	484	10,7	37,5	71,1	21,1	69,8
Iš viso:	15,5	657	15,1	45,3	77,2	31	68,7
Tikslinimo stačiakampių skaičius – 4, mAP – 78,65 %							
PETS09-S2L1	15,9	703	15,6	34	72,8	18,7	63,6
TUD-Campus	17	742	16,6	61,3	84,9	48,5	65
TUD-Stadtmitte	16,9	619	16,5	57,6	73,3	35,5	63,4
ETH-Sunnyday	16,8	615	16,4	56,1	81	40,2	69,9
ETH-Pedcross2	16,7	834	16,4	26,8	87,7	22	71,6
KITTI-17	20,6	781	20,1	44,2	84,4	33,6	68,4
ADL-Rundle-6	11,1	465	10,8	50,6	78,1	34	69
Venice-2	11,1	549	10,8	34,7	82,6	26	69,3
Iš viso:	15,8	663	15,4	45,7	80,6	32,3	67,5
Tikslinimo stačiakampių skaičius – 3, mAP – 78,68 %							
PETS09-S2L1	16,5	717	16,1	25,5	62,2	7,4	64,5
TUD-Campus	16,8	781	16,4	60,2	88,5	50,7	64,5
TUD-Stadtmitte	16,9	656	16,4	69,5	92,8	63	68,5
ETH-Sunnyday	16,9	631	16,5	51,3	79,3	36,1	71,6
ETH-Pedcross2	16,8	856	16,5	26,7	93,5	23,7	72,5
KITTI-17	20,9	780	20,4	42,6	83,9	30,9	68,1
ADL-Rundle-6	11	458	10,8	47,8	68,8	23,7	70,1
Venice-2	11	468	10,8	34,2	59,6	9,7	69,4
Iš viso:	15,9	668	15,5	44,7	78,6	30,7	68,7
Tikslinimo stačiakampių skaičius – 2, mAP – 77,62 %							
PETS09-S2L1	16,1	933	15,8	14,7	54,2	0,9	63,8
TUD-Campus	16,8	795	16,5	60,4	92,7	54	66,4
TUD-Stadtmitte	16,8	610	16,3	73,	92,1	65,9	69
ETH-Sunnyday	16,9	644	16,4	56,6	82,6	42,6	72,8
ETH-Pedcross2	16,8	857	16,5	27,2	92,2	24,1	71,2
KITTI-17	21	750	20,4	37,3	76,8	23,5	66,8
ADL-Rundle-6	11	426	10,7	49,1	66,5	22,3	69,5
Venice-2	11	459	10,8	36,2	67,4	17,5	71,1
Iš viso:	15,8	684	15,4	44,4	78,1	31,4	68,8



4.8 pav. Daugelio objektų sekimo spartos priklausomybė nuo tikslinimo stačiakampių skaičiaus



4.9 pav. Daugelio objektų sekimo parametrų priklausomybė nuo tikslinimo stačiakampių skaičiaus

4.3 lentelėje nurodyti parametrai leidžia daryti keletą išvadų. Pirmiausia, mažėjantis tikslinimo stačiakampių skaičius neturi didelės įtakos sekimo tikslumui. Tikėtina, kad taip nutinka dėl to, kad stačiakampio, apibrėžiančio pėsčiąjį, kraštinės visada yra panašaus santykio. Visgi nebuvo pastebėtas ryškus algoritmo paspartėjimas, paprastai vos kelios dešimtosios kadro per sekundę, t. y. tik keliais procentais.

Tikslinimo stačiakampių skaičiaus ir formos derinimas k-vidurkių algoritmu leido pasiekti nežymiai tikslesnį pėsčiųjų aptikimo tikslumą. Dirbtinių neuronų tinklo mokymo metu taip pat pastebėta, kad tinklas sparčiau konverguoja bei mažiau osciliuoja apie globalų minimumą. Taip pat reiktų paminėti tai, kad taikant penkis tikslinamuosius stačiakampius dalis parametrų buvo prastesni nei naudojant keturis, todėl siekiant optimalaus rezultato gali tekti derinti šį parametą

pagal poreikius. Toks keitimas galimas naudojant kokius nors validavimo duomenis, kad ir tokius, kurie pateikiami objektų sekimui skirtuose duomenų rinkiniuose.

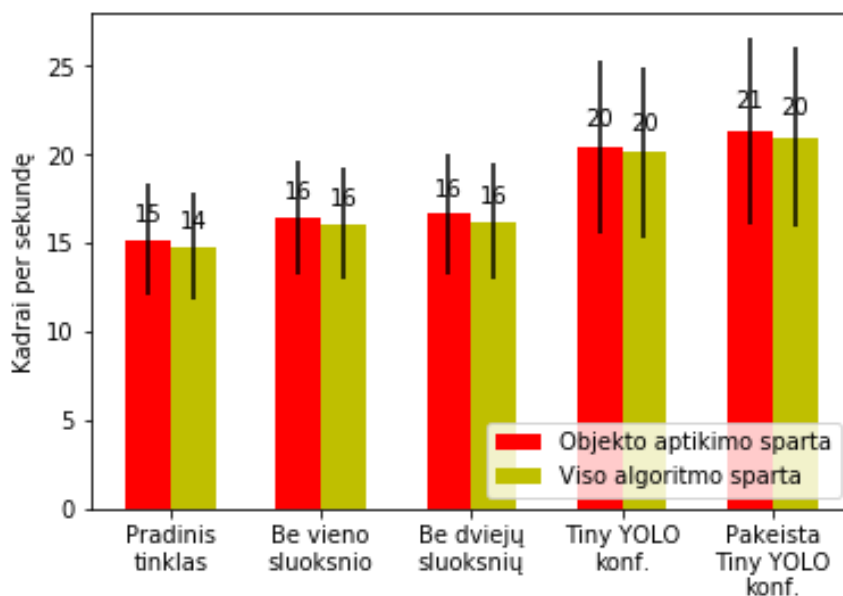
4.2.2. Objektų aptikimo algoritmo spartinimas atsisakant tam tikrų neuronų tinklo sluoksnių

Tam, kad būtų pateiktas objekto aptikimo rezultatas, konvoliucinis neuronų tinklas turi atlikti daugybę konvoliucijos (sąsukos) operacijų. Šios operacijos iš esmės atlieka pradinio vaizdo filtravimą. Pradiniai sluoksniai padeda atpažinti paprastus objekto požymius – kampus, spalvas, toliau einantys – sudėtingesnius. Siekiant objektų aptikimą pritaikyti dideliame klasių skaičiui, daugiau sluoksnių turintis tinklas leidžia šiam atskirti daugiau unikalių objekto požymių. Bandant vaizde pažymėti tik vieno tipo objektus tokių sluoksnių turėtų užtekti mažiau. Atlikti eksperimentiniai tyrimai siekia ištirti tinklo sluoksnių kiekio įtaką, kai vaizde sekami pėstieji. Iš viso atlikti keturi skirtingi bandymai. Pirmuoju atveju išmetamas priešpaskutinis konvoliucinis sluoksnis, turintis 1024-is filtrus su 3×3 branduoliu (paskutiniojo išmesti negalima dėl tinklo struktūros), antruoju – du priešpaskutiniai, turintys po 1024-is filtrus su 3×3 branduoliu. Trečiuoju atveju išbandomas *Tiny YOLO* tinklas (Redmon *et al.* 2016), turintis kompaktiškesnę YOLO tinklo struktūrą ir keturis kartus mažiau mokymo parametrų (64 MB). Skyrius baigiamas pastarojo tinklo struktūros tolimesniu sumažinimu išmetant dar tris sluoksnius ir taip užtikrinant, kad mokymo parametrų sumažėja iki 6,8 MB. Šių bandymų rezultatai pateikti 4.4 lentelėje, o spartos ir parametrų grafinis palyginimas pateiktas 4.8-4.9 paveiksluose.

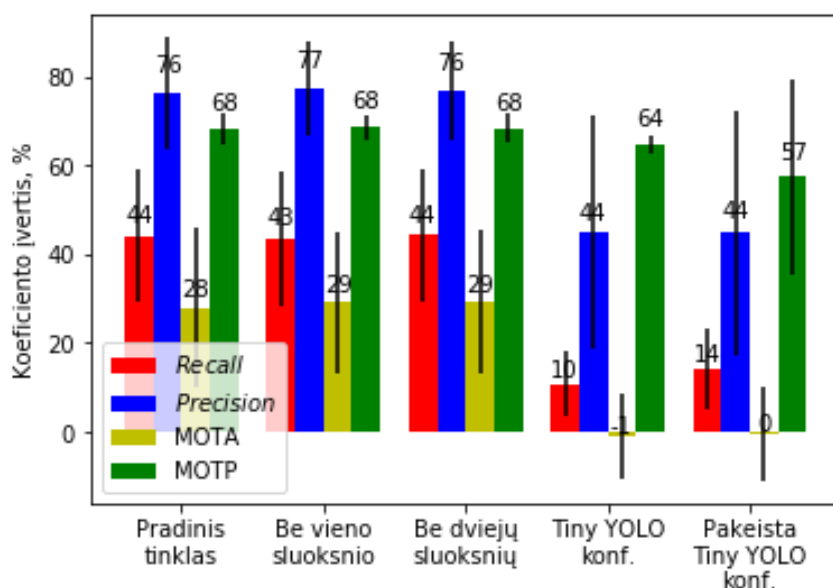
4.4 lentelė. Daugelio objektų sekimo rezultatai

Vaizdo įrašo pavadinimas	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	Recall	Precision	MOTA	MOTP
Atsisakius vieno konvoliucinio sluoksnio, mAP – 71,69 %							
PETS09-S2L1	16,9	889	16,6	19,4	65,5	7,2	66
TUD-Campus	17,4	677	17	59,6	79,9	43,5	63,8
TUD-Stadtmitte	17,3	654	16,9	66,3	87,3	55,4	65,8
ETH-Sunnyday	17,6	625	17,1	56,4	82,4	42,1	71,6
ETH-Pedcross2	17,5	852	17,1	28,2	94,3	25,6	71,9
KITTI-17	21,9	815	21,4	40,8	79,2	26,3	68,8
ADL-Rundle-6	11,5	498	11,3	43,9	70	23	70,5
Venice-2	11,4	478	11,2	32,8	60	9,6	70,8
Iš viso:	16,4	686	16,1	43,4	77,3	29,1	68,7
Atsisakius dviejų konvoliucinių sluoksnių, mAP – 71,62 %							

PETS09-S2L1	17	886	16,7	21,1	66,8	8,9	64,5
TUD-Campus	17,7	693	17,2	64,3	81,6	47,6	65,5
TUD-Stadtmitte	17,8	666	17,3	63,8	86	52,1	63
ETH-Sunnyday	17,5	635	17	54,6	82,9	40,8	71,8
ETH-Pedcross2	17,7	813	17,3	28,5	89,1	24	72
KITTI-17	22,3	804	21,7	42,2	83,1	31,1	69,9
ADL-Rundle-6	11,3	489	11,1	45,1	71,6	25,3	70,9
Venice-2	11,5	420	11,2	35	54,4	4,2	69,2
Iš viso:	16,6	676	16,2	44,3	76,9	29,3	68,4
<i>Tiny YOLO konfigūracija, mAP – 66,46 %</i>							
PETS09-S2L1	20,2	1050	19,8	9,8	43,8	-3,8	65,6
TUD-Campus	21,8	1050	21,3	18,4	75,9	11,1	62,6
TUD-Stadtmitte	21,8	1070	21,4	15,7	61,1	4,1	65,5
ETH-Sunnyday	22,1	2800	21,9	1,2	6	-17	66,1
ETH-Pedcross2	22,1	4500	22	0	0,4	-12,3	60,1
KITTI-17	29	1290	28,4	8,8	42,3	-4,2	65,7
ADL-Rundle-6	13	714	12,8	22,8	69,7	11	64,7
Venice-2	13,1	1010	13	10	59,2	2,5	67,4
Iš viso:	20,4	1690	20,1	10,8	44,8	-1,1	64,7
<i>Atsisakius trijų konvoliucinių sluoksnių Tiny YOLO konfigūracijoje, mAP – 53,26 %</i>							
PETS09-S2L1	21,4	733	20,8	25,3	60,8	6,9	62,5
TUD-Campus	23	999	22,5	13,6	47,6	-2,5	63,5
TUD-Stadtmitte	22,6	863	22,1	16,7	46	-4,3	63,9
ETH-Sunnyday	23,2	3120	23	0,2	0,8	-18,3	70,3
ETH-Pedcross2	22,8	4180	22,7	0	0	-12,4	0
KITTI-17	30,8	1005	29,9	23,8	77,5	15	67,6
ADL-Rundle-6	13,5	736	13,3	20,2	71	9,8	64,3
Venice-2	13,5	762	13,3	14	55,6	1,9	68,4
Iš viso:	21,4	1550	21	14,2	44,9	-0,49	57,6



4.10 pav. Daugelio objektų sekimo spartos priklausomybė nuo atmetamų sluoksnių skaičiaus



4.11 pav. Daugelio objektų sekimo parametrų priklausomybė nuo atmetamų sluoksnių skaičiaus

Iš 4.10 paveikslą galima matyti, kad atsisakant vis daugiau konvoliucinių sluoksnių algoritmo sparta vis auga. 4.11 paveikslas parodo, kad norint paspartinus pėsčiųjų sekimo algoritmą išlaikyti ir sekimo tikslumą galima atsisakyti tik pirmųjų dviejų sluoksnių. Tai galima aiškinti tuo, jog paskutiniuose sluoksniuose gaunami pėsčiųjų išskyrimui naudojami požymiai nėra tokie svarbūs, o atsisakius daugiau sluoksnių prarandama jau ir itin svarbi informacija. MOTA parametras *Tiny-YOLO* konfigūracijos atveju sumažėja iki nulio, tai parodo, kad objekto sekime klaidų jau yra tiek

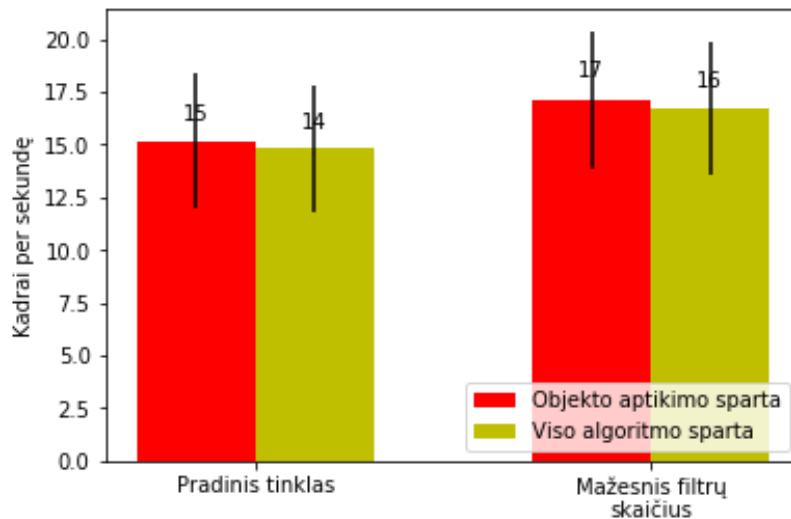
pat, kiek turėtų būti aptikta objektų. Smarkiai sumažėja ir kiti parametrai, taip pat padidėja šių parametru standartinis nuokrypis.

4.2.3. Objektų aptikimo algoritmo spartinimas mažinant filtrų skaičių konvoliuciniuose sluoksniuose

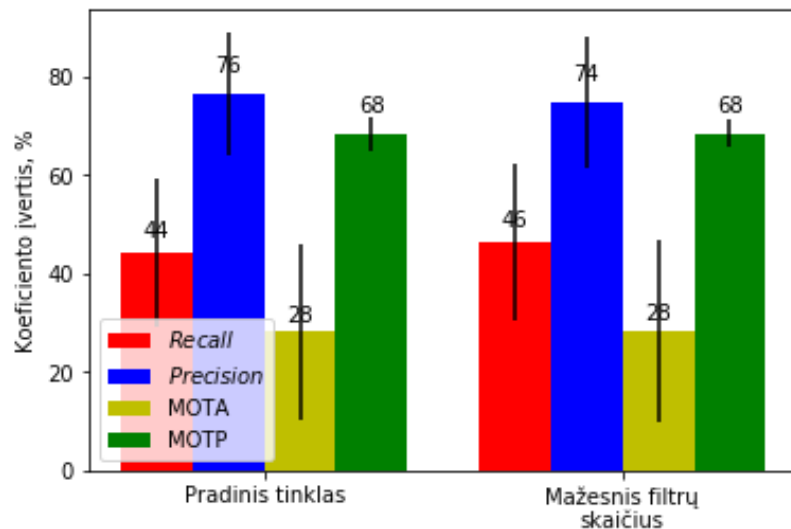
Sumažinti objekto aptikimo algoritmo parametru skaičių ir taip paspartinti objektų paiešką galima ne tik atsisakant ištisų konvoliucinių sluoksnių. Lygiai taip pat skaičiavimų sumažėjimo galima tikėtis sumažinant filtrų skaičių konvoliuciniuose sluoksniuose. Tam, kad būtų galima patikrinti objektų sekimo spartos ir tikslumo priklausomybę nuo filtrų skaičiaus konvoliuciniame sluoksnyje, atliktas eksperimentas – trijuose sluoksniuose tinklo pabaigoje (neskaitant paskutiniojo) filtrų kiekis sumažintas nuo 1024-ių iki 512-os. Tokio eksperimento rezultatai pateikti 4.5 lentelėje. Tam, kad gautus rezultatus būtų galima grafiškai palyginti su pradinės pėsčiųjų sekimo sistemos rezultatai, pateikiami 4.12 ir 4.13 paveikslai.

4.5 lentelė. Daugelio objektų sekimo rezultatai mažinant filtrų skaičių konvoliuciniuose sluoksniuose

Vaizdo įrašo pavadinimas	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	<i>Recall</i>	<i>Precision</i>	MOTA	MOTP
mAP – 77,31 %							
PETS09-S2L1	17,4	835,6	17	21	64,8	7,5	63,8
TUD-Campus	18,3	675,9	17,9	63,2	77,5	43,2	64,8
TUD-Stadtmitte	18,6	724,1	18,1	66,8	96,5	63	69,1
ETH-Sunnyday	18,2	595,8	17,6	61,1	77,4	40,6	72,1
ETH-Pedcross2	18,2	908,1	17,8	27	91,3	23,7	72
KITTI-17	22,3	639,3	21,6	45,5	58,9	11,1	67,5
ADL-Rundle-6	12,1	490,5	11,8	47,1	72,8	27,3	68,8
Venice-2	12	427,5	11,7	38,3	58,1	9,4	69,1
Iš viso:	17,1	661	16,7	46,3	74,7	28,2	68,4



4.12 pav. Daugelio objektų sekimo spartos palyginimas, kai naudojamas mažesnis filtrų skaičius konvoliuciniame sluoksnyje



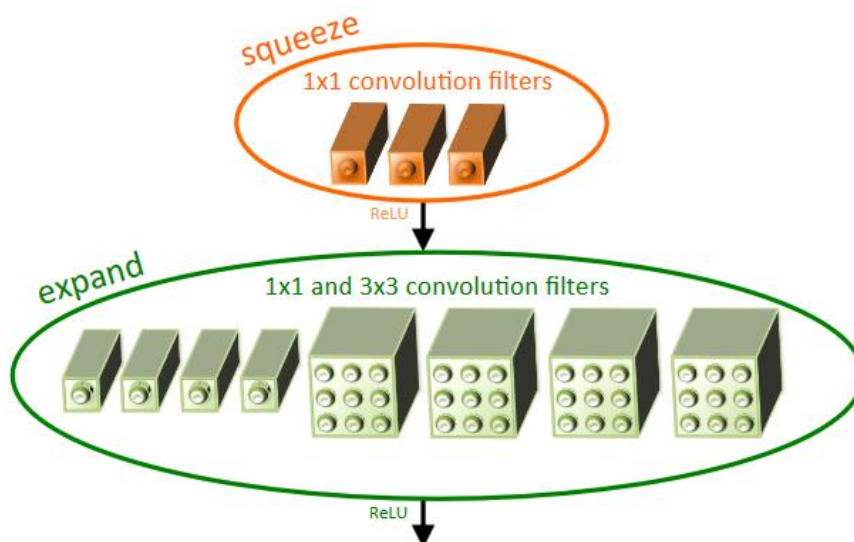
4.13 pav. Daugelio objektų sekimo parametrų palyginimas, kai naudojamas mažesnis filtrų skaičius konvoliuciniame sluoksnyje

Kaip ir atsisakant atskirų sluoksnių, taip ir sumažinant filtrų skaičių viename konvoliuciniame sluoksnyje galima sutaupyti skaičiavimų ir taip paspartinti pėsčiųjų sekimo algoritmą. Iš 4.12 paveikslo galima matyti, kad objekto sekimo sparta nežymiai padidėjo, o iš 4.13 paveikslo galima daryti išvadą, kad sekimo tikslumą apibūdinantys parametrai iš esmės nesuprastėja.

4.2.4. Objekto aptikimo vaizde spartinimas naudojant *SqueezeNet Fire* elementus

Populiarėjant konvoliuciniams dirbtinių neuronų tinklams ir augant tyrimų, skirtų jiems, skaičiui, ieškota, kaip mažinti tinklo parametrų skaičių keičiant pačių konvoliucinių sluoksnių išdėstymą ir jų struktūrą. Tai lėmė naujų konvoliucinių neuronų tinklų atsiradimą, tokių, kaip

Inception (Szegedy *et al.* 2016) arba *Squeezenet* (Iandola *et al.* 2016: 1–13). Pastarojo pagrindinis konstrukcinis blokas pavadintas *Fire* moduliu. Jo struktūrinė schema pateikta 4.14 paveiksle. Elemento įėjime – 1×1 konvoliucijų sluoksnis, po kurio seka 1×1 ir 3×3 konvoliucijų sluoksnis. Konvoliucijos (sąsukos) su 1×1 branduoliu sumažina filtrų skaičių išlaikydamos didžiąją dalį reikalingos informacijos, todėl tolimesnis signalo filtravimas taikant didesnių branduolių (3×3 , 5×5 , 7×7) sąsukas atliekamas su mažesniu įėjimo parametrų skaičiumi, todėl mažiau užtrunka ir skaičiavimai (Szegedy *et al.*). Naudos aktyvacijos funkcijos – išlyginto tiesinio vieneto (angl. *rectified linear unit*, ReLU). Tinklo svorių regularizacijai *SqueezeNet* kūrėjai naudoja *dropout* techniką, magistro darbe ji pakeista į rinkinio normalizacijos (angl. *batch normalization*) techniką. Toks sprendimas pasirinktas todėl, kad pastaroji technika naudojama objektų aptikimo algoritme YOLOv2.

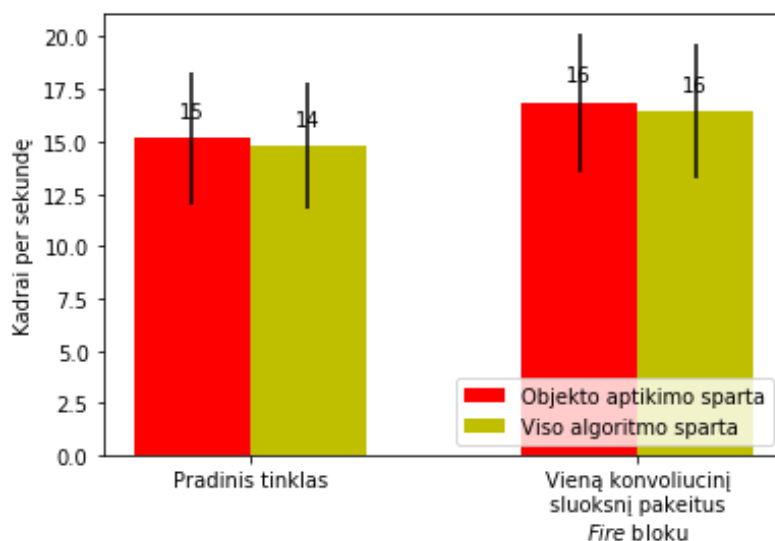


4.14 pav. *Squeezenet* tinklo *Fire* bloko struktūrinė schema (Iandola *et al.* 2016: 1–13)

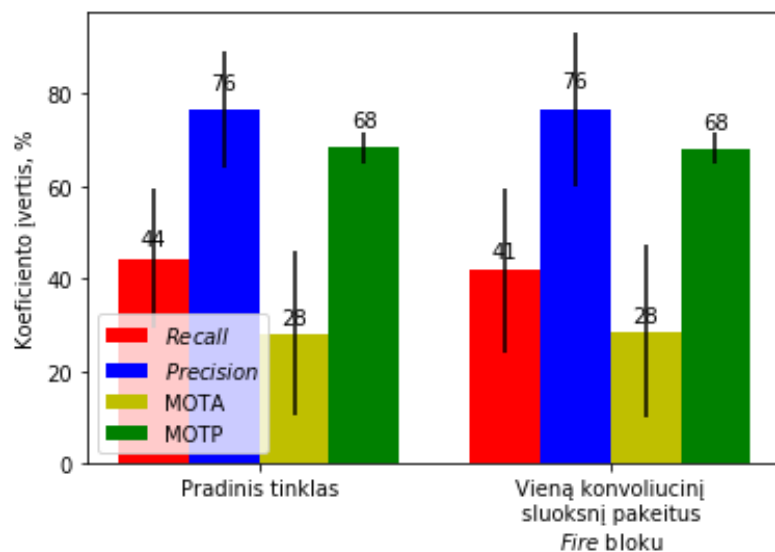
4.6 lentelė. Daugelio objektų sekimo rezultatai pakeitus konvoliucinius sluoksnius į *SqueezeNet Fire* blokus

Vaizdo įrašo pavadinimas	Objekto aptikimo sparta, FPS	Sekimo sparta, FPS	Viso algoritmo sparta, FPS	<i>Recall</i>	<i>Precision</i>	MOTA	MOTP
mAP – 71,7 %							
PETS09-S2L1	17,4	1190	17,2	9,2	43,8	-3,5	64,2
TUD-Campus	17,7	749	17,3	63,5	88,4	53,8	65,6
TUD-Stadtmitte	17,6	743	17,2	55,4	93,9	50,8	62,4
ETH-Sunnyday	17,9	574	17,3	60,2	75,7	38	72,1
ETH-Pedcross2	17,9	952	17,6	23	92,2	20,2	70,7
KITTI-17	22,5	825	21,9	41,2	84,3	31,1	69,7

ADL-Rundle-6	11,7	498	11,5	46,3	76,1	29,6	70,9
Venice-2	11,8	451	11,5	34,4	57,2	7,4	69,4
Iš viso:	16,8	748	16,4	41,7	76,5	28,4	68,1



4.15 pav. Daugelio objektų sekimo spartos palyginimas, kai vienas konvoliucinis sluoksnis keičiamas *SqueezeNet Fire* bloku



4.16 pav. Daugelio objektų sekimo tikslumo parametrų palyginimas, kai vienas konvoliucinis sluoksnis keičiamas *SqueezeNet Fire* bloku

Iš 4.15 paveikslo galima matyti, kad *SqueezeNet Fire* blokas paspartina sekimo algoritmo veikimą, o 4.16 paveikslas įrodo, kad toks būdas tik nežymiai sumažina sekimo algoritmo tikslumą. Taigi galima daryti išvadą, kad sąsuka su 1×1 branduoliu, kuri sumažina filtrų skaičių, leidžia prarasti tik nedidelę dalį objekto aptikimą vykdančio algoritmo kokybiškam veikimui reikalingos informacijos.

APIBENDRINIMAS

Pradinė sistema sudaryta iš YOLOv2 (angl. *You Only Look Once*) algoritmo, skirto pėstiesiems aptikti, bei SORT (angl. *Simple Online and Realtime Tracking*) algoritmo, skirto aptiktiems objektams sekti. Iš šių elementų sudaryto algoritmo mokymui naudotas pėsčiųjų fotografijų duomenų rinkinys *Caltech Pedestrian Dataset*.

Objektų sekimo tikslumui apibūdinti pasirinkti penki įverčiai – apdorotų kadrų per sekundę skaičius, etalono padengimo matavimu tikslumas (angl. *recall*), matavimo panaudojimo etalonui padengti tikslumas (angl. *precision*), daugelio objektų sekimo tikslumas (MOTP) ir daugelio objektų sekimo taiklumas (MOTA). Objektų aptikimo tikslumui įvertinti naudotas suvidurkintas vidutinis aptikimo tikslumas (angl. *mean average precision*).

Atlikta pradinio objektų sekimo algoritmo analizė rodo, kad daugiausia laiko sekimo algoritmas turi skirti objekto aptikimui, kitos objektų sekimo algoritmo dalys atliekamos dešimtimis kartų greičiau, todėl pasirinkta optimizuoti būtent objektų aptikimo dalį.

Magistro darbo rengimo metu atlikti dviejų rūšių eksperimentiniai tyrimai. Vienu atveju tirta objekto sekimo tikslumo priklausomybė nuo objektų paieškos dažnumo, antru atveju siekta išsiaiškinti, kaip objekto sekimo tikslumą veikia objekto aptikimo architektūros pokyčiai – keistas tiek neuronų tinklo dydis sluoksniais, tiek vieno sluoksnio plotis, tiek tikslinamųjų stačiakampių skaičius atskirame vaizdo stačiakampyje, taip pat vietoje priešpaskutinio konvoliucinio sluoksnio išbandytas *SqueezeNet* (Iandola *et al.* 2016: 1–13) giluminiame neuronų tinkle naudotas *Fire* modulis.

Objekto (pėsčiojo) paieškos vykdymas ne kiekviename kadre leidžia smarkiai paspartinti vaizdo sekimą prarandant nedidelę dalį sekimo tikslumo. Objekto paiešką atliekant kas antrame kadre MOTA parametras krenta maždaug trimis procentiniais punktais, tačiau algoritmas paspartėja maždaug 90 %. Toliau didinant objekto paieškos intervalą objekto lokalizavimas vykdomas praktiškai taip pat tiksliai (kaip matoma 4.2 ir 4.4 paveiksluose), tačiau vis retesnė objekto paieška didina sekimo klaidų skaičių – tai rodo vis po kelis procentus krentantis MOTA parametras.

Nedidelė dalis objektų sekimo tikslumo prarandama mažinant tikslinimo stačiakampių kiekį YOLOv2 tinkle. Tokį rezultatą galima pagrįsti tuo, kad pėsčiųjų ūgis beveik be išimčių yra daug didesnis nei plotis, todėl galima atlikti mažiau spėjimų kiekvienam paveikslėlio kvadratui. Nors toks rezultatas leidžia atsisakyti kai kurių tikslinimo stačiakampių, objekto aptikimo sparta nuo to praktiškai nepasikeitė – spartos augimas matuojamas tik dešimtosiomis kadro per sekundę dalimis. Tikslinimo stačiakampių formos keitimas pritaikant k-vidurkių algoritmą naudotam duomenų rinkiniui nežymiai (maždaug 1 %) padidina objektų aptikimo tikslumą.

Sumažinti aptinkant objektus naudojamo konvoliucinio dirbtinių neuronų tinklo parametrų skaičių ir taip paspartinti algoritmo veikimą galima tiek mažinant tinklo sluoksnių skaičių, tiek mažinant viename sluoksnyje naudojamų filtrų skaičių. Atlikus nežymius pakeitimus galutiniams neuronų tinklo sluoksniams galima apdoroti vidutiniškai du papildomus kadrus per sekundę praktiškai tokiu pačiu tikslumu. Didesni pokyčiai siekiant toliau spartinti objektų aptikimą turi neigiamų padarinių. Tai galima matyti iš *Tiny YOLO* modifikacijų – nors spartos augimas ženklus, atpažinimo kokybė suprastėja tiek, kad MOTA parametras tampa neigiamas. Tai reiškia, kad aptinkant objektus klaidų yra daugiau nei tikslių spėjimų.

Tinklo parametrų skaičius mažėja ir vietoj konvoliucinių sluoksnių naudojant *SqueezeNet* tinklo kūrėjų pasiūlytus *Fire* blokus (Iandola *et al.* 2016: 1–13). Filtrų skaičiaus sumažinimas konvoliucijomis su 1×1 branduoliu padidina algoritmo spartą ir todėl apdorojama vidutiniškai 1,6 papildomo kadro per sekundę.

Iš visų atliktų eksperimentų sėkmingiausias sekimo algoritmo spartos didinimas objektų paiešką atliekant ne visuose kadruose, kitais atvejais siekiant panašaus spartos padidėjimo prarandama per daug sekimo tikslumo (tai įrodo eksperimentai su *Tiny YOLO* konfigūracija). Tikėtina, kad tam tikri pakeitimai, pavyzdžiui, nežymus tinklo sluoksnių skaičiaus mažinimas, viename sluoksnyje naudojamų filtrų skaičius ar *Fire* blokai, gali būti naudojami kartu su retesne objektų paieška arba jungiami tarpusavyje. Tai įrodyti arba paneigti galėtų papildomi eksperimentai.

LITERATŪRA

- Ahn, H.S., Ko, K. H. 2009. Simple pedestrian localization algorithms based on distributed wireless sensor networks, *IEEE Transactions on Industrial Electronics* 56(10): 4296–4302. DOI: 10.1109/TIE.2009.2017097.
- Bernardin K., Elbs A., Stiefelhagen R. 2006. Multiple object tracking performance metrics and evaluation in a smart room environment, *Sixth IEEE International Workshop on Visual Surveillance, in conjunction with ECCV* [interaktyvus] 90: 91 [žiūrėta 2018 m. kovo 28 d.]. Prieiga per internetą: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.7070&rep=rep1&type=pdf>.
- Bewley, A., Ge, Z., Ott, L., Ramos, F., Upcroft, B. 2016. Simple Online and Realtime Tracking, *Proceedings - International Conference on Image Processing, ICIP 2016–August*: 3464–3468. DOI: 10.1109/ICIP.2016.7533003.
- Bojarski, M. et al. 2016. End to End Learning for Self-Driving Cars [interaktyvus]: 1–9 [žiūrėta 2018 m. kovo 29 d.]. Prieiga per internetą: <http://arxiv.org/abs/1604.07316>.
- Chavali, N., Agrawal, H., Mahendru, A., Batra, D. 2015. Object-Proposal Evaluation Protocol is „Gameable“. DOI: 10.1109/CVPR.2016.97.
- Chen, C., Cheng, Y. 2010. MATLAB-based simulators for mobile robot Simultaneous Localization and Mapping, *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings* 2: 576–581. DOI: 10.1109/ICACTE.2010.5579471.
- Chen, S. Y. 2012. Kalman filter for robot vision: A survey, *IEEE Transactions on Industrial Electronics* 59(11): 4409–4420. DOI: 10.1109/TIE.2011.2162714.
- Correa, J., Soto, A. 2010. Active Visual Perception for Mobile Robot Localization, *Journal of Intelligent and Robotic Systems* 58(3–4): 339–354. DOI: 10.1007/s10846-009-9348-4.
- Dollár, P., Wojek, C., Schiele, B., Perona, P. 2009. Pedestrian detection: A benchmark, *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*: 304–311. DOI: 10.1109/CVPRW.2009.5206631.
- Dollár, P., Wojek, C., Schiele, B., Perona, P. 2012. Pedestrian detection: An evaluation of the state of the art, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34(4): 743–761. DOI: 10.1109/TPAMI.2011.155.
- Fearnhead, P. 1998. Sequential Monte Carlo methods in lter theory Trinity Term 1998, *University of Oxford*.
- Girshick, R. 2015. Fast R-CNN, *Proceedings of the IEEE International Conference on Computer Vision 2015 Inter*: 1440–1448. DOI: 10.1109/ICCV.2015.169.
- Gustafsson, F. 2010. Applications Practice with Positioning Particle Filter Theory and, *IEEE*

- Aerospace and Electronic Systems Magazine* 25(7): 528–82. DOI: 10.1109/MAES.2010.5546308.
- He, K., Zhang, X., Ren, S., Sun, J. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *Proceedings of the IEEE International Conference on Computer Vision* 2015 Inter: 1026–1034. DOI: 10.1109/ICCV.2015.123.
- He, K., Gkioxari, G., Dollár, P., Girshick, R. 2017. Mask R-CNN [interaktyvus][žiūrėta 2017 m. lapkričio 17 d.]. Prieiga per internetą: <http://arxiv.org/abs/1703.06870>.
- Hess, R., Fern, A. 2009. Discriminatively trained particle filters for complex multi-object tracking, *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*: 240–247. DOI: 10.1109/CVPRW.2009.5206801.
- Hosang, J., Benenson, R., Dollar, P., Schiele, B. 2016. What Makes for Effective Detection Proposals?, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38(4): 814–830. DOI: 10.1109/TPAMI.2015.2465908.
- Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., Bengio, Y. 2016. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations: 1–29. DOI: 10.1109/CVPR.2016.90.
- Iandola, F. N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W. J., Keutzer, K. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size: 1–13. DOI: 10.1007/978-3-319-24553-9.
- Lundin T., Moerland P. Quantization and Pruning of Multilayer Perceptrons: Towards Compact Neural Networks [interaktyvus][žiūrėta 2018 m. balandžio 2 d.]. Prieiga per internetą: <http://publications.idiap.ch/downloads/reports/1997/com97-02.pdf>.
- Milan A., Leal-Taixe L., Reid I., Roth S., Schindler K. 2016. MOT16: A Benchmark for Multi-Object Tracking [interaktyvus]: 1–12 [žiūrėta 2018 m. balandžio 4 d.]. Prieiga per internetą: <http://arxiv.org/abs/1603.00831>.
- Munkres J. 1957. Algorithms for the Assignment and Transportation Problems, *Journal of the Society for Industrial and Applied Mathematics* 5(1): 32–38. DOI: 10.1137/0105003.
- Park, E., Ahn, J., Yoo, S. 2017. Weighted-Entropy-Based Quantization for Deep Neural Networks, *Cvpr*: 5456–5464. DOI: 10.1109/CVPR.2017.761.
- Ramakoti, N., Vinay, A., Jatoth, R. K. 2009. Particle Swarm Optimization Aided Kalman Filter for Object Tracking, *2009 International Conference on Advances in Computing, Control, and Telecommunication Technologies*: 531–533. DOI: 10.1109/ACT.2009.135.
- Redmon, J. Darknet: Open Source Neural Networks in C.
- Redmon, J., Divvala, S., Girshick, R., Farhadi, A. 2015. You Only Look Once: Unified, Real-Time Object Detection. DOI: 10.1109/CVPR.2016.91.

- Redmon, J., Farhadi, A. 2016. YOLO9000: Better, Faster, Stronger. DOI: 10.1142/9789812771728_0012.
- Ren, S., He, K., Girshick, R., Sun, J. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(6): 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.
- Siekmann, J., Wahlster, W. 2010. *RoboCup 2009: Robot Soccer World Cup XIII*. DOI: 10.1007/978-3-642-11876-0.
- Simon, D. 2001. Kalman Filtering, *Embedded System Programming* (June): 72–79. DOI: 10.1016/j.automatica.2005.03.018.
- Smith, K., Gatica-Perez, D., Odobez, J., Ba S. 2005. Evaluating Multi-Object Tracking.
- Szegedy, C., Reed, S., Sermanet, P., Vanhoucke, V., Rabinovich, A. 2014. Going deeper with convolutions: 1–12.
- Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A. 2016. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. DOI: 10.1016/j.patrec.2014.01.008.
- Thrun, S. 2012. Particle Filters in Robotics [interaktyvus][žiūrėta 2016 m. lapkričio 10 d.]. Prieiga per internetą: <http://arxiv.org/abs/1301.0607>.
- Wan, E. A., Van Der Merwe, R. 2002. The unscented Kalman filter for nonlinear estimation, *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*: 153–158. DOI: 10.1109/ASSPCC.2000.882463.
- Zhang, H., Negahdaripour, S. 2010. EKF-based recursive dual estimation of structure and motion from stereo data, *IEEE Journal of Oceanic Engineering* 35(2): 424–437. DOI: 10.1109/JOE.2010.2041573.
- Zhang, N., Donahue, J., Girshick, R., Darrell, T. 2014. Part-based R-CNNs for fine-grained category detection, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 8689 LNCS(1): 834–849. DOI: 10.1007/978-3-319-10590-1_54.

PRIEDAI

A priedas. Pasiūlyto algoritmo programos išeities kodas

```

1  from ctypes import *
2  from sort import *
3  import math
4  import random
5  import numpy as np
6  import matplotlib.pyplot as plt
7  from PIL import Image
8  import cv2
9
10 colors = tuple([ ( random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255)) for k in range(32) ])
11
12 def sample(probs):
13     s = sum(probs)
14     probs = [a/s for a in probs]
15     r = random.uniform(0, 1)
16     for i in range(len(probs)):
17         r = r - probs[i]
18         if r <= 0:
19             return i
20     return len(probs)-1
21
22 def c_array(ctype, values):
23     arr = (ctype*len(values))()
24     arr[:] = values
25     return arr
26
27 class BOX(Structure):
28     _fields_ = [("x", c_float),
29                 ("y", c_float),
30                 ("w", c_float),
31                 ("h", c_float)]
32
33 class IMAGE(Structure):
34     _fields_ = [("w", c_int),
35                 ("h", c_int),
36                 ("c", c_int),
37                 ("data", POINTER(c_float))]
38
39 class METADATA(Structure):
40     _fields_ = [("classes", c_int),
41                 ("names", POINTER(c_char_p))]
42
43
44
45 #lib = CDLL("/home/pjreddie/documents/darknet/libdarknet.so", RTLD_GLOBAL)
46 lib = CDLL("libdarknet.so", RTLD_GLOBAL)
47 lib.network_width.argtypes = [c_void_p]
48 lib.network_width.restype = c_int
49 lib.network_height.argtypes = [c_void_p]
50 lib.network_height.restype = c_int
51
52 predict = lib.network_predict
53 predict.argtypes = [c_void_p, POINTER(c_float)]
54 predict.restype = POINTER(c_float)
55
56 make_boxes = lib.make_boxes
57 make_boxes.argtypes = [c_void_p]

```

```

58 make_boxes.restype = POINTER(BOX)
59
60 free_ptrs = lib.free_ptrs
61 free_ptrs.argtypes = [POINTER(c_void_p), c_int]
62
63 num_boxes = lib.num_boxes
64 num_boxes.argtypes = [c_void_p]
65 num_boxes.restype = c_int
66
67 make_probs = lib.make_probs
68 make_probs.argtypes = [c_void_p]
69 make_probs.restype = POINTER(POINTER(c_float))
70
71 detect = lib.network_predict
72 detect.argtypes = [c_void_p, IMAGE, c_float, c_float, c_float, POINTER(BOX),
POINTER(POINTER(c_float))]
73
74 reset_rnn = lib.reset_rnn
75 reset_rnn.argtypes = [c_void_p]
76
77 load_net = lib.load_network
78 load_net.argtypes = [c_char_p, c_char_p, c_int]
79 load_net.restype = c_void_p
80
81 free_image = lib.free_image
82 free_image.argtypes = [IMAGE]
83
84 letterbox_image = lib.letterbox_image
85 letterbox_image.argtypes = [IMAGE, c_int, c_int]
86 letterbox_image.restype = IMAGE
87
88 load_meta = lib.get_metadata
89 lib.get_metadata.argtypes = [c_char_p]
90 lib.get_metadata.restype = METADATA
91
92 load_image = lib.load_image_color
93 load_image.argtypes = [c_char_p, c_int, c_int]
94 load_image.restype = IMAGE
95
96 ndarray_image = lib.ndarray_to_image
97 ndarray_image.argtypes = [POINTER(c_ubyte), POINTER(c_long),
POINTER(c_long)]
98 ndarray_image.restype = IMAGE
99
100 predict_image = lib.network_predict_image
101 predict_image.argtypes = [c_void_p, IMAGE]
102 predict_image.restype = POINTER(c_float)
103
104 network_detect = lib.network_detect
105 network_detect.argtypes = [c_void_p, IMAGE, c_float, c_float, c_float,
POINTER(BOX), POINTER(POINTER(c_float))]
106
107 def classify(net, meta, im):
108     out = predict_image(net, im)
109     res = []
110     for i in range(meta.classes):
111         res.append((meta.names[i], out[i]))
112     res = sorted(res, key=lambda x: -x[1])
113     return res
114
115 def detect(net, meta, im, thresh=.24, hier_thresh=.24, nms=.45):
116     #im = load_image(image, 0, 0)
117     boxes = make_boxes(net)
118     probs = make_probs(net)

```

```

119     num = num_boxes(net)
120     network_detect(net, im, thresh, hier_thresh, nms, boxes, probs)
121     res = []
122     for j in range(num):
123         for i in range(meta.classes):
124             if probs[j][i] > 0:
125                 res.append((meta.names[i], probs[j][i], (boxes[j].x,
boxes[j].y, boxes[j].w, boxes[j].h)))
126     res = sorted(res, key=lambda x: -x[1])
127     free_image(im)
128     free_ptrs(cast(probs, POINTER(c_void_p)), num)
129     return res
130
131 def detect_im(net, meta, im, thresh=.24, hier_thresh=.24, nms=.45):
132     boxes = make_boxes(net)
133     probs = make_probs(net)
134     num = num_boxes(net)
135     network_detect(net, im, thresh, hier_thresh, nms, boxes, probs)
136     res = []
137     for j in range(num):
138         for i in range(meta.classes):
139             if probs[j][i] > 0:
140                 res.append((meta.names[i], probs[j][i], (boxes[j].x,
boxes[j].y, boxes[j].w, boxes[j].h)))
141     res = sorted(res, key=lambda x: -x[1])
142     free_image(im)
143     free_ptrs(cast(probs, POINTER(c_void_p)), num)
144     return res
145
146
147 def detect_np(net, meta, np_img, thresh=.24, hier_thresh=.24, nms=.45):
148     im = nparray_to_image(np_img)
149     boxes = make_boxes(net)
150     probs = make_probs(net)
151     num = num_boxes(net)
152     network_detect(net, im, thresh, hier_thresh, nms, boxes, probs)
153     res = []
154     for j in range(num):
155         for i in range(meta.classes):
156             if probs[j][i] > 0:
157                 res.append((meta.names[i], probs[j][i], (boxes[j].x,
boxes[j].y, boxes[j].w, boxes[j].h)))
158     res = sorted(res, key=lambda x: -x[1])
159     free_image(im)
160     free_ptrs(cast(probs, POINTER(c_void_p)), num)
161     return res
162
163
164 def nparray_to_image(img):
165     data = img.ctypes.data_as(POINTER(c_ubyte))
166     image = ndarray_image(data, img.ctypes.shape, img.ctypes.strides)
167
168     return image
169
170 if __name__ == "__main__":
171     sequences = ['PETS09-S2L1', 'TUD-Campus', 'TUD-Stadtmitte', 'ETH-
Sunnyday', 'ETH-Pedcross2', 'KITTI-17', 'ADL-Rundle-6', 'Venice-2']
172     phase = 'train'
173     net = load_net(b"cfg/yolo-obj-less-slight-mod.cfg", b"yolo-obj-less-
slight-mod_45000.weights", 0)
174     meta = load_meta(b"cfg/obj.data")
175     #vidcap = cv2.VideoCapture('moskva.mov')
176     det_time = 0
177     tracking_time = 0

```

```

178     id_time = 0
179     success = True
180     frame_count = 0
181     skipRate = 1
182     cv2.namedWindow("Image", cv2.WINDOW_NORMAL)
183
184     # Place to store results after tracking
185     if not os.path.exists('output'):
186         os.makedirs('output')
187
188     for seq in sequences:
189         directory='2DMOT2015/%s/%s/img1/'%(phase,seq)
190         mot_tracker = Sort()
191         seq_det_time = 0
192         seq_tracking_time = 0
193         seq_frame_count = 0
194         with open('output/%s.txt'%(seq),'w') as out_file:
195             for file in sorted(os.listdir(directory)):
196                 if file.endswith(".jpg"):
197                     image = cv2.imread('{}{}'.format(directory,file))
198                     start_time = time.time()
199                     seq_frame_count += 1
200                     if skipRate == 0 or skipRate == 1:
201                         r = detect_np(net, meta, image)
202                     elif (seq_frame_count%skipRate == 1):
203                         r = detect_np(net, meta, image)
204                     #print (r)
205                     dets = []
206                     for i in range(0,len(r)):
207                         dets.append(r[i][2])
208                     detections = np.asarray(dets)
209
210                     for i in range(0,len(r)):
211                         w_half= detections[i, 2]/2.0
212                         h_half= detections[i, 3]/2.0
213                         x_mid= detections[i, 0]
214                         y_mid= detections[i, 1]
215                         detections[i, 0] = x_mid - w_half
216                         detections[i, 2] = x_mid + w_half
217                         detections[i, 1] = y_mid - h_half
218                         detections[i, 3] = y_mid + h_half
219
220                     tracking_start_time = time.time()
221                     trackers = mot_tracker.update(detections)
222                     tracking_cycle_time = time.time() - tracking_start_time
223                     seq_tracking_time += tracking_cycle_time
224
225                     if(len(trackers)==0): trackers= [[0, 0, 0, 0, 0]]
226
227                     d= trackers[0]
228                     for i in range(0, 4):
229                         if np.isnan(d[i]):
230                             d[i] = 0
231
232                     for d in trackers:
233                         print('%d,%d,%.2f,%.2f,%.2f,%.2f,1,-1,-1,-
1'%(seq_frame_count,d[4],d[0],d[1],d[2]-d[0],d[3]-d[1]),file=out_file)
234                         pt1 = (int(d[0]), int(d[3]))
235                         pt2 = (int(d[2]), int(d[1]))
236                         cv2.rectangle(image, pt1, pt2, colors[int(d[4]%32)],
3)
237                     cv2.imshow("Image", image)
238                     if cv2.waitKey(1) == 27:
239                         break

```

```

240         cycle_time = time.time() - start_time
241         seq_det_time += (cycle_time - tracking_cycle_time)
242         print("\r\nDetection time for {}: {:.3f} for {} frames or {:.1f}
FPS".format(seq,seq_det_time, seq_frame_count, seq_frame_count/seq_det_time))
243         print("Tracking time for {}: {:.3f} for {} frames or {:.1f}
FPS".format(seq,seq_tracking_time, seq_frame_count,
seq_frame_count/seq_tracking_time))
244         print("Total time for {}: {:.3f} for {} frames or {:.1f}
FPS".format(seq,seq_tracking_time+seq_det_time, seq_frame_count,
seq_frame_count/(seq_tracking_time+seq_det_time)))
245         frame_count += seq_frame_count
246         det_time += seq_det_time
247         tracking_time += seq_tracking_time
248
249         print("\r\nDetection took: {:.3f} for {} frames or {:.1f}
FPS".format(det_time,frame_count,frame_count/det_time))
250         print("Tracking took: {:.3f} for {} frames or {:.1f}
FPS".format(tracking_time,frame_count,frame_count/tracking_time))
251         print("Combined took: {:.3f} for {} frames or {:.1f}
FPS".format(tracking_time+det_time,frame_count,frame_count/(tracking_time+det_t
ime)))
252

```

B priedas. Pranešimo XXI-oje Lietuvos jaunųjų mokslininkų konferencijoje medžiaga



PĖSČIŲŲ ATPAŽINIMAS NAUDOJANT GILUMINĮ DIRBTINIŲ NEURONŲ TINKLĄ YOLO V2

21-OJI JAUNŲJŲ MOKSLININKŲ KONFERENCIJA
„MOKSLAS – LIETUVOS ATEITIS: ELEKTRONIKA IR ELEKTROTECHNIKA“

Artūras JONKUS, arturas.jonkus@stud.vgtu.lt

Vadovas: prof. dr. Artūras SERACKIS

Elektroninių sistemų katedra

Vilnius, 2018



Artūras JONKUS

2018-03-16

Jaunųjų mokslininkų konferencija
„Mokslas – Lietuvos ateitis: Elektronika ir elektrotechnika“

Motyvacija



▶ Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2

2 9



Artūras JONKUS

2018-03-16

Jaunųjų mokslininkų konferencija
„Mokslas – Lietuvos ateitis: Elektronika ir elektrotechnika“

Tikslas ir uždaviniai

Tikslas – panaudoti YOLOv2 dirbtinių neuronų tinklą kiek įmanoma geresniems rezultatams pasiekti pėsčiųjų atpažinime

Uždaviniai:

- Paleisti esamą modelį
- Apmokyti modelį naujiems duomenims
- Optimizuoti parametrus

▶ Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2

3 9

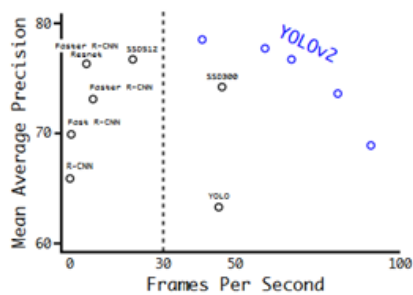
Žinomi sprendimai

- Orientuotų gradientų histograma (HoG) + atraminių vektorių mašinos (SVM) (Dalal et al., 2005 „Histograms of oriented gradients for human detection“)
- Ensemble inference networks (Fukui et al. 2015. „Pedestrian Detection Based on Deep Convolutional Neural Network with Ensemble Inference Network“)
- Faster R-CNN ir Boosted Forest (Zhang et al. 2016. „Is Faster R-CNN Doing Well for Pedestrian Detection?“)

► Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2

4 9

Kodėl YOLOv2?



► Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2

5 9

Tinklo struktūra



► Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2

6 9

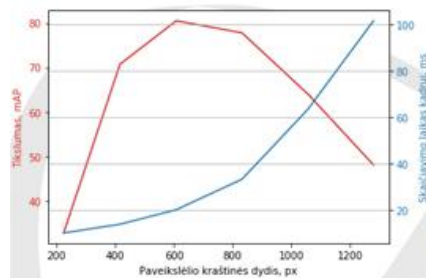
Rezultatai

- Caltech pėsčiųjų duomenų rinkinys
- 16757 paveikslėlių mokymui
- 1592 paveikslėliai testavimui
- Dvi dienos mokymui Nvidia GTX 1080Ti



► Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2 7 9

Rezultatai



► Pėsčiųjų atpažinimas naudojant giluminį dirbtinių neuronų tinklą YOLOv2 8 9

Ačiū už dėmesį