Vilniaus Universitetas

Fizikos fakultetas

Taikomosios elektrodinamikos ir telekomunikacijų institutas

Daniel Zakševski

DIRBTINIŲ NEURO TINKLŲ TAIKYMAS VAIZDO ATPAŽINIMUI BEI MECHANIZMŲ VALDYMUI

Magistro studijų baigiamasis darbas

Elektronikos ir telekomunikacijų technologijų
studijų programa

Studentas Daniel Zakševski

Leista ginti 2022-05-25

Darbo vadovas doc. Vytautas Jonkus

Recenzentas dr. Rimvydas Aleksiejūnas

Instituto vadovas prof. Robertas Grigalaitis

Vilnius 2022

TURINYS

ĮVADAS

Dirbtinis intelektas yra per paskutinius dešimtmečius išpopuliarėjusi mokslo sritis. Jos tikslas yra suprasti ir atkurti intelektą. Intelekto sąvoka yra labai plati, tad ir dirbtinio intelekto sąvokų yra daug. Viena iš jų apibrėžia dirbtinį intelektą kaip sistemą, kuri priima racionalius sprendimus. Mašininis mokymas yra viena iš dirbtinio intelekto mokslo srities posričių, kurioje sistemų kūrimas vyksta ne konkrečių instrukcijų programavimu, o mokymo būdu, kurio metu algoritmas automatiškai tobulėja „mokymosi“ iš duomenų būdu. Duomenų kokybė bei jų kiekis šioje srityje yra ypatingai svarbūs.

Vienas iš perspektyviausių algoritmų, naudojamų mašininio mokymo srityje yra dirbtiniai neuroniniai tinklai, kurių architektūra yra įkvėpta biologinių neurologinių tinklų. Šie tinklai yra sudaryti iš dirbtinių neuronų, o šiuolaikiniai neuro tinklai turi didelį kiekį įvairių tipų sluoksnių ir yra vadinami multisluoksniais. Vienas tokių neuro tinklų yra vadinamas konvoliuciniu neuro tinklu, kuriame bent vienas iš sluoksnių yra konvoliucinis. Šių tinklų išskirtinumas slypi tame, kad jie puikiai tinka darbui su atvaizdais bei taikymais, kur reikalingas milžiniškas kiekis duomenų.

Prižiūrimasis mokymas bei skatinamasis mokymas, mašininio mokymo šakos, yra du skirtingi algoritmų tipai naudojami mokant agentus. Abiejose šakose taikomi neruroniniai tinklai, tačiau užduotys, kurias bando išspręsti šios mokslo šakos yra skirtingos. Vaizdų atpažinime, konvoliucinių neuro tinklų architektūrą radikaliai pakeitė YOLOv3 (angl. *You Only Look Once*) algoritmas. Jo išskirtinumas yra tame, kad bruožams tinkle gauti yra naudojamas milžiniškas konvoliucinis tinklas, vadinamas DarkNet-53, kuris leidžia YOLOv3 architektūrai, maža tikslumo sumažėjimo kaina, būti ypač greitam. Ši architektūra buvo toliau tobūlinama iki YOLOv5, kurios architektūra paremta CSPDarkNet neuro tinklu. Skatinamasis mokymas, tuo tarpu, yra dažnai naudojamas robotinių mechanizmų valdyme. Mechanizmas, algoritmo pagalba, tokio kaip PPO (angl. *proximap policy optimization*), atlieka veiksmus ir gauna atlygį, kuris parodo, ar atliktas veiksmas yra geras. Pagal šį atlygį algoritmas nusprendžia kuriuos veiksmus kartoti, o kuriuos ne. Fiziškas tokių sistemų treniravimas yra sudėtingas, tad šiam tikslui pasiekti kartais naudojamos simuliacijos.

Išmokius modelį atpažinti objektus, jis turi būti išbandomas praktiškai. Vienas iš lengvai prieinamų būdų tai padaryti yra pasinaudoti OpenCV biblioteka ir reikiamą objektą detektuojantį dirbtinį intelektą.

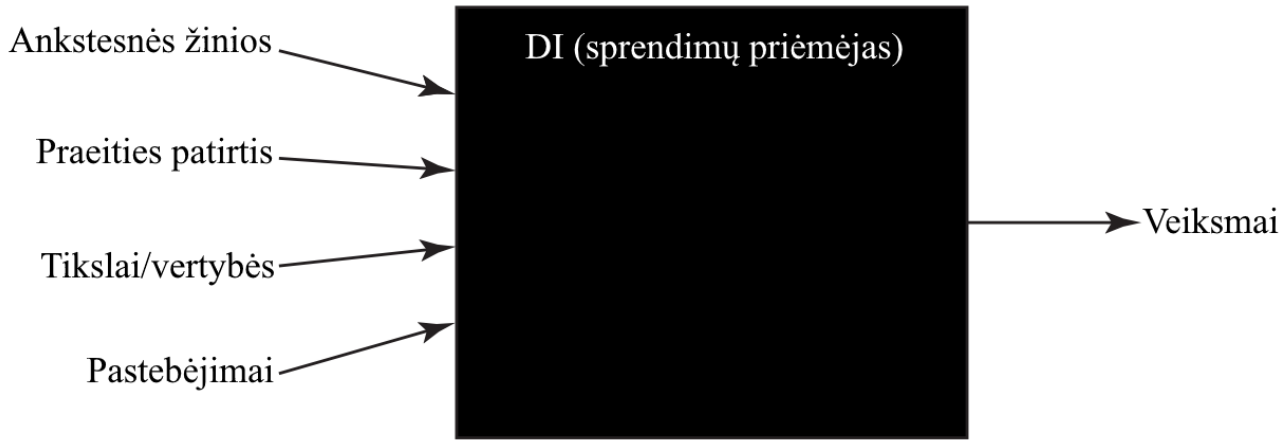
Šio darbo tikslas yra panaudoti dirbtinį neuroninį tinklą vaizdo atpažinimui bei išmokyti robotą surasti ir paliesti objektą virtualioje aplinkoje.

1. Teorijos apžvalga

1.1. Dirbtinis intelektas

Viena perspektyviausių šių laikų mokslinių disciplinų – dirbtinis intelektas – jau domino žmoniją nuo pat Senovės Graikijos civilizacijos laikų. Viename iš mitų yra pasakojama apie bronzinį milžiną Talosą, kurio užduotis buvo ginti Kretos salą. Šio milžino kūrėjas buvo Hefaistas, ugnies dievas, o savo kūrinį jis padovanojo Kretos karaliui. Žinoma, tai yra tik mitai ir tokių pasakojimų apie vienokį ar kitokį dirbtinį intelektą per žmonijos istoriją galima rasti daug.

Dirbtinis intelektas kaip mokslinė disciplina prasidėjo XX a. viduryje, o pati sąvoka „dirbtinis intelektas“ pirmą kartą pavartota John McCarthy 1956 metais [1, 2]. Ši mokslinė sritis bando ne tik suprasti žmogaus intelektą, bet ir jį atkurti. Apibrėžti ši daug emocijų kelianti žodžių junginį galima dvipusiškai – galvoti ir daryti žmoniškai arba galvoti ir daryti racionaliai. Bandymai atkurti intelektą buvo daromi iš abiejų šių pusių. Šiame darbe dirbtinio intelekto patikimumo sąvoka yra sukonkretinama – dirbtinis intelektas veikia korektiškai jeigu jis elgiasi racionaliai, t. y. elgiasi taip, kad būtų pasiektas geriausias rezultatas, arba, esant neapibrėžtumams, tikėtinai geriausias rezultatas atsižvelgiant į visus šiuo metu turimus duomenis [3]. Dirbtinis intelektas charakterizuojamas sistemos sugebėjimu teisingai interpretuoti išorinius duomenis, iš jų mokytis, ir, pasinaudojant šiomis žiniomis, pasiekti konkrečius tikslus, geriausią rezultatą [4]. Tačiau kas yra geriausias rezultatas arba teisingas sprendimas? Tai vertinti galime atsižvelgiant į pasekmes. Kai sprendimų priėmėjas (pvz.: žmogus arba robotas) atsiranda kažkokioje aplinkoje, jis generuoja veiksmų seką remiantis šiuo metu gaunamais suvokimais. Ši veiksmų seka kažkokiu būdu paveikia aplinką. Jeigu sprendimų priėmėjo sugeneruota veiksmų seka pakeitė aplinką, mūsų požiūriu, pageidautinai, sakome, kad jis atliko savo darbą gerai. Žinoma, kiekvienoje situacijoje pageidautini veiksmai skiriasi, todėl, dažniausiai, vertintojas turės aprašytas pageidautinas pasekmes kievienai sprendimų priėmėjo situacijai. Galima



1 pav. Sprendimų priėmėjas – juoda dėžė [5].

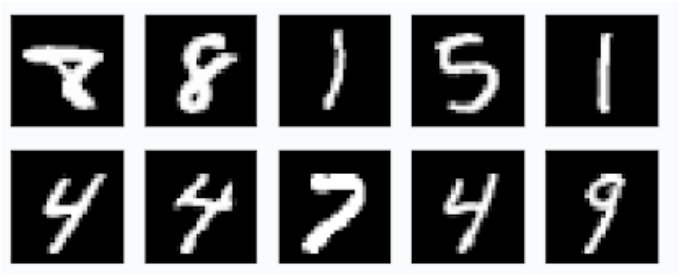
pastebėti, kad užduotys ir vertinimo kriterijai yra sukonkretinami, kitaip tariant, dirbtinio intelekto taikymas vyksta konkrečioje srityje, konkrečių užduočių vykdime. Tai yra vadinama siauru dirbtiniu intelektu. Būtent siauras dirbtinis intelektas yra naudojamas Facebook veidų atpažinime, Siri balso atpažinime arba Tesla savarankiško vairavimo sistemose. Šio darbo rašymo metu tai dirbtinio intelekto viršūnė.

Norint, kad dirbtinis intelektas (toliau DI) priimtų bet kokius sprendimus, reikia kažkokiu būdu paaiškinti, kas yra gerai, o kas blogai – apibrėžiamas tikslas (arba naudingumo funkcija) [5]. Naudingumo funkcija gali būti labai paprasta (pvz.: „jeigu DI laimi šaškių partiją, tai 1, kitais atvejais 0“) arba labai sudėtinga (pvz.: „daryk matematinius veiksmus panašius į tuos, kurie buvo veiksmingi praeityje“). Tikslas gali būti aiškiai apibrėžtas arba sumestas (angl. *induced*). Jeigu DI sistema yra suprogramuojama skatinamajam mokymui, tikslai yra netiesioginiu būdu nustatomi apdovanojant ir baudžiant tam tikrą elgėsį [6]. Kitose DI sistemose tikslai nėra jokiu būdu parodomi, o tik netiesiogiai atvaizduojami treniravimo duomenyse. Tokį sudėtingą DI elgėsį ir sugebėjimą mokytis galima sukurti algoritmų pagalba. Algoritmas yra baigtinė, gerai apibrėžtų žingsnių seka, skirta išspręsti problemą arba atlikti skaičiavimams [6, 7]. Įprastai, ne DI srityje, jeigu turime išspręsti dvi problemas, mums reikės parašyti dvi skirtingas programas pasinaudojant tam tikrais algoritmais. Šios programos gali turėti kažką bendrą, pavyzdžiui, programavimo kalbą arba tą pačią duomenų bazės struktūrą, tačiau programa, skirta šachmatų žaidimui, jokiu būdu negalės apdoroti kreditinių kortelių duomenų. DI srityje tokia dilema nebūtinai galioja – tas pats algoritmas gali būti panaudojamas niekuo nesusijusioms problemoms spręsti.

1.2. Mašininis mokymas

Mašininis mokymas [8] yra viena iš dirbtinio intelekto posričių. Ši mokslo sritis kuria ir tiria algoritmus, kurie automatiškai tobulėja vykstant „mokymosi“ iš duomenų procesui [9, 10]. Uždaviniai, kuriuos žmogus daro kiekvieną dieną įprastai, tokie kaip vairavimas, kalbos arba vaizdų supratimas, negali būti programuojami įprastu būdu. Nors dažnai atliekami, neturime pakankamų įžvalgų į šių uždavinių atlikimą, kad galėtume parašyti algoritmus jų sprendimui. Tačiau pasinaudojant mašininiu mokymusi šie uždaviniai yra įveikiami su pakankamai gerais rezultatais, su prielaida, kad buvo panaudotas didelis kiekis treniravimo duomenų. Kita uždavinių šeima, kurių sprendime panaudojamas mašininis mokimasis yra tiesiog per sudėtingi žmonėms atlikti dėl milžiniškų duomenų rinkinių dydžių ir sudėtingumo: astronominių duomenų apdorojimas, orų prognozė, žiniatinklių paieškos sistemų veikimas. Dar vienas mašininio mokymo privalumas, paminėtas anksčiau, yra algoritmų panaudojamumas įvairių problemų sprendimui [6, 10].

Tradiciškai, išskiriami trys mašininio mokymo implementavimo metodai: prižiūrimas mokymas (angl. *supervised learning*), neprižiūrimas mokymas (angl. *unsupervised learning*), skatinamasis mokymas (angl. *reinforcement learning*) [6, 8, 9, 10, 11]. Skirtumai tarp šių metodų ir mašininio mokymo ypatumai gali būti parodomi populiaraus pavyzdžio, MNIST duomenų rinkinio (2 pav.),



2 pav. MNIST duomenų rinkinyje esančių skaičių atvaizdų pavyzdys [13].

pagalba [11, 12].

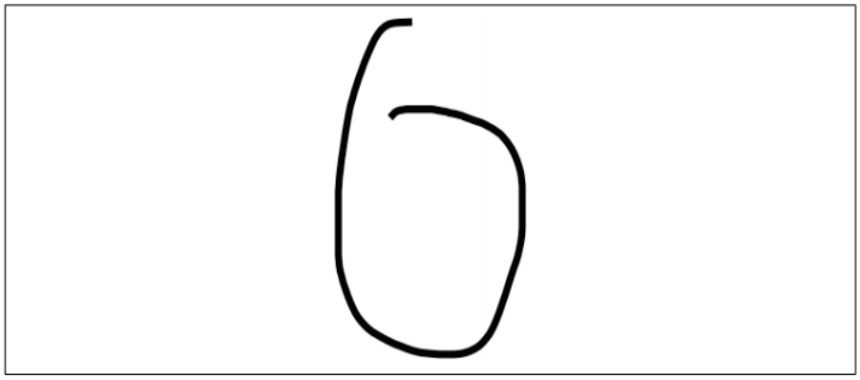
Šiame rinkinyje yra milžiniškas skaičius 28×28 ranka rašytų skaitmenų atvaizdų. Tai reiškia, kad vienas rinkinio skaitmuo gali būti užrašomas 784 realių skaitmenų vektoriumi $\boldsymbol{x}$. Tikslas šiuo atveju yra sukurti mašiną, kuri įvestyje priimtų $\boldsymbol{x}$, o išvestyje parodytų identifikuotą skaitmenį – 0, 1, …, 9. Ši užduotis yra sudėtinga dėl to, kad kiekvieno asmens raštas yra skirtingas. Šią problemą būtų įmanoma išspręsti aprašant kokias nors taisykles šių skaitmenų atpažinimui, tačiau tyrimai rodo, kad toks būdas neduoda gerų rezultatų [11]. Geresnis šios problemos sprendimas gaunamas naudojant mašininio mokymo metodą, kurio metu N ilgio skaitmenų rinkinys $\{x_1, x_2, \ldots, x_N\}$, vadinamas treniravimo rinkiniu, yra naudojamas adaptyvios funkcijos svertinių verčių koregavimui. Treniravimo rinkinio skaitmenų kategorijos (t. y. koks skaičius – 0, 1, …, 9 – yra atvaizde) yra nustatomos iš anksto. Tokiu atveju, kai treniravimosi rinkinys yra iš anksto teisingai kategorizuotas (t. y. kiekvienas skaitmens vektorius įvestyje jau turi su savimi susietą teisingą išvesties vektorių), mokymas yra vadinamas prižiūrimu [2, 6, 10, 11]. Jeigu, atvirkščiai, įvesties vektoriai nėra susiejami su jokiu teisingu atsakymu, mokymas vadinamas neprižiūrimu.

Tokio mašininio mokymo algoritmo rezultatas gali būti išreikštas funkcija $y(x)$, kuri priima atvaizdo pikselių verčių vektorių $\boldsymbol{x}$ ir parodo skaičiaus kategoriją vektoriaus pavidalu, $\boldsymbol{y}$. Konkretus funkcijos pavidalas yra nustatomas treniravimo, arba mokymo, metu iš treniravimo rinkinio. Kai šis algoritmas, dažniau vadinamas modeliu, yra išmokytas, jis gali apibrėžti jam prieš tai nematytus skaičių atvaizdus. Modelio gebėjimas taisyklingai kategorizuoti naujus pavyzdžius yra vadinamas apibendrinimu (angl. *generalization*) [6, 10, 11]. Praktikoje, treniravimo rinkinys yra sudarytas iš baigtinio skaičiaus skirtingų įvesties vektorių, o tai reiškia, kad modelis jokiu būdu nebus matęs visus įmanomus vektorius. Dėl to apibendrinimas mašininio mokymo modeliams yra labai svarbus.

Daugumoje taikymų įvesties duomenys būna iš anksto apdorojami, su tikslu, kad bus išskirti duomenyse esantys bruožai. Pavyzdžiui, grįžtant prie skaitmenų atpažinimo, skaitmenų atvaizdai dažniausiai būna verčiami bei suvienodinami jų dydžiai. Tai padeda tuo, kad patys skaičiai atvaizde būna toje pačioje vietoje ir tokio pat dydžio, o tai reiškia, kad algortimui bus lengviau atpažinti išskirtinius bruožus kategorije. Reikia pažymėti, kad, jeigu išankstinis apdorojimas yra daromas treniravimo rinkiniui, tai tie patys žingsniai turi būti daromi praktiškai taikant modelį. Galima taip pat pabrėžti, kad išankstinis apdorojimas padeda su duomenų apdorojimo greičiu [8, 11]. Tai yra ypač svarbu jeigu modelis turi veikti realiu laiku. Pavyzdžiui, jeigu tikslas yra atpažinti veidus realiu laiku aukštos kokybės video sraute, kompiuteris turi pereiti per milžinišką pikselių skaičių per sekundę bei juos praleisti pro sudėtingą bruožų atpažinimo modelį. Toks uždavinys yra įmanomas tik su ypač galingais kompiuteriais. Tokiu atveju geresnė išeitis yra surasti naudingus bruožus, kurie yra apdorojami greitai, bet tuo pačiu turi savyje reikalingą informaciją, leidžiančią veidą atskirti nuo ne veido. Būtent tie surasti bruožai yra naudojami kaip įvestis veidų atpažinimo algoritmui.

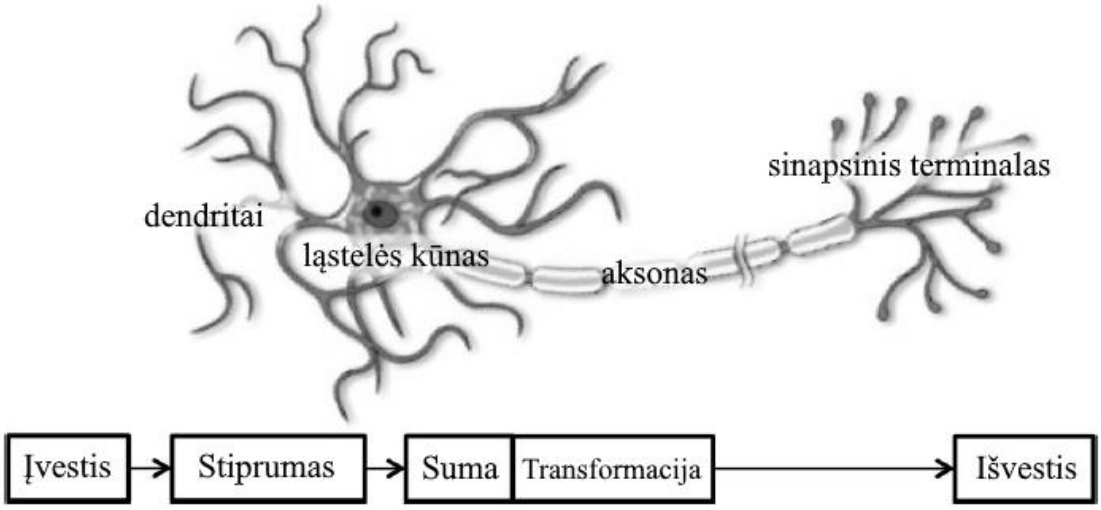
1.3. Dirbtiniai neuro tinklai

Prieš tai aptartas mašininis mokymas ir paprastesni algoritmai naudojami jame veikia gerai ir pastumėjo tokių uždavinių kaip rašto supratimas, vaizdo atpažinimas sprendimą į priekį. Tačiau žvilgtelėjus į 3 pav. pavaizduotą skaitmenį, net mokytojui būtų sudėtinga suprasti, ar ten yra 0, ar 6. Toks uždavinys yra tuo labiau sudėtingas kokiam nors algoritmui. Todėl buvo ieškota kitų požiūrių į šią problemą. Daug dalykų kuriuos žmogus išmoksta mokykloje turi nemažai bendro su kompiuteriais. Mokomės daugybos, lygčių sprendimo ir diferencijavimo panaudojant tam tikrą taisyklių rinkinį. Tačiau dalykai, kuriuos išmokstame labai ankstyvame amžiuje, dalykai, kurie mums atrodo labiausiai naturalūs ir nereikalaujantys pastangų, yra išmokstami pavyzdžių pagalba, ne formulių. Tėvai neaiškina savo dvimečiui vaikui kaip atpažinti šunį išmatuojant jo kūno kontūrą arba nosies ilgį. Vaikas išmoksta atpažinti šunį iš jam parodomo gausaus kiekio pavyzdžių ir būdamas pataisytas, kai spėja neteisingai. Kitaip tariant, kai gimstame, smegenyse egzistuoja modelis, kurio įvestyje yra mūsų sensorių (akių, ausų itt.) gaunami duomenys ir kuris spėja apie tai, ką mes šiuo



3 pav. 0 ar 6? Ypač sudėtingas yždavinys algoritmui [14].

metu patiriame. Jei mūsų spėjimas paneigiamas kaip neteisingas, mūsų modelis yra pakoreguojamas taip, kad atsižvelgtų į šią informaciją. Gyvenimo eigoje (ir evoliucijos metu) šis modelis tampa vis tikslesnis. Žinoma, visą tai vyksta pasamoniškai, tačiau būtent tai mokslininkai nusprendė panaudoti gerinant mašininio mokymo algoritmus [14, 15]. Dirbtiniai neuro tinklai (arba tiesiog neuro tinklai) buvo sukurti žmogaus nervų sistemos simuliacijai mašininio mokymo uždaviniams, skačiavimo vienetus modelyje traktuojant kaip žmogaus neuronus [16]. Galutinis neuro tinklų rezultatas yra žmogaus nervų sistemos atliekamus skaičiavimus simuliuojantis dirbtinis intelektas. Įkvėpimas tokio modelio sukūrimui, žinoma, buvo žmogaus nervų sistemos dalis, neuronas. Neuronas yra optimizuotas tam, kad priimtų informaciją iš kitų neuronų, ją apdorotų ir išsiųstų rezultatą kitiems neuronams. Šis procesas ir biologinio neurono struktūra yra parodyta 4 pav. Neuronas gauna signalus dendrituose (įvestis). Kiekviena iš dendritų šakų yra dinamiškai sustiprinama arba susilpninama, priklausomai nuo to kaip dažnai ji yra naudojama ir būtent kiekvieno sujungimo stiprumas lemia signalo šakoje indėlį į galutinę neurono išvestį. Ląstelės kūne signalai yra susumuojami atsižvelgus į



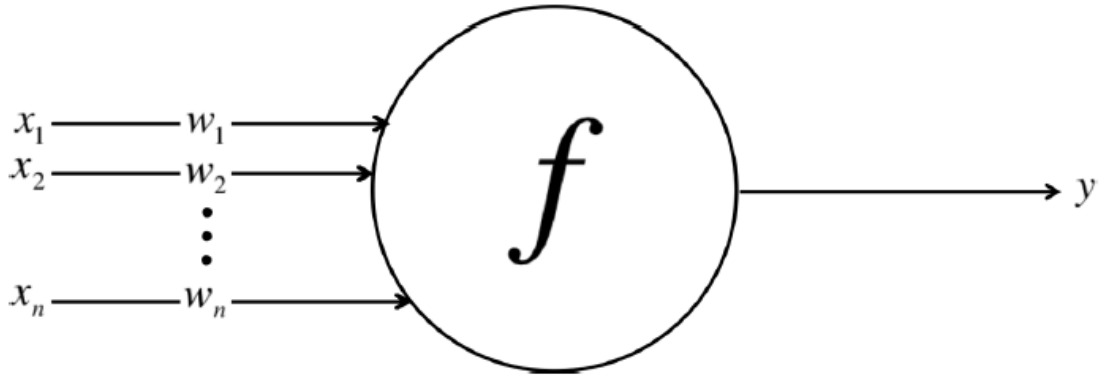
4 pav. Supaprastinta funkcinė biologinio neurono struktūra [14].

jų šakos svertines vertes. Ši suma yra transformuojama į naująjį signalą, kuris toliau sklinda aksonu, kol nepasiekia kitų neuronų.

Šį biologinių neuronų funkcijų supratimą yra įmanoma paversti į dirbtinį modelį, kurį naudotų kompiuteris. Tokio modelio architektūra yra parodyta 5 pav. Dirbtinio neurono įvestyje yra tam tikras kiekis įvadų, x_1, x_2, ..., x_n, kurių kiekvienas yra paveiktas svoriniu daugikliu w_1, w_2, ..., w_n. Vėlgi, visi įvadai yra susumuojami:

$$z = \sum_{i=0}^{n} w_i x_i \qquad (1)$$

Šis rezultatas yra paveikiamas funkcija *f* ir gaunama dirbtinio neurono išvestis $y = f(z)$. Tokiuose modeliuose funkcija yra vadinama aktyvacijos funkcija [17, 18]. Ji lemia, ar neuronas yra „aktyvuotas“ ar ne. Ši funkcija yra būtina, kadangi be jos dirbtinis neuro tinklas supaprastėja į tiesinę regresiją.



5 pav. Dirbtinio neurono struktūra [14].

Verta pažymėti, kad dirbtinių neuro tinklų išraiškos dažnai būna užrašomos vektorine forma. Perrašant įvesti kaip vektorių $\boldsymbol{x} = [x_1\ x_2\ \dots\ x_n]$ ir svorinius daugiklius kaip $\boldsymbol{w} = [w_1\ w_2\ \dots\ w_n]$, galime perrašyti išvestį kaip [14]:

$$y = f(\boldsymbol{x} \cdot \boldsymbol{w} + b) \tag{2}$$

kur *b* yra tam tikra slenkstinė vertė. Tai yra svarbu dėl to, kad šiuos modelius yra lengviau implementuoti programiškai apie juos galvojant kaip apie vektorių manipuliacijas.

Paprasčiausias dirbtinis neuro tinklas yra vadinamas perceptronu (angl. *perceptron*) [8, 14, 15, 16]. Šis neuro tinklas turi vieną įvesties sluoksnį (įvestyje priimamos kelios vertės, 0 arba 1) ir vieną išvestį (taip pat 0 arba 1). Visos įvesties vertės yra paveikiamos svoriniais koeficientais ir susumuojami. Ši suma yra paveikiama Heaviside funkcija – jeigu suma yra daugiau nei kažkokia, iš anksto nustatyta, slenkstinė vertė, perceptrono išvestyje turėsime 1, jei mažiau, turėsime 0. Matematiškai tai galime aprašyti taip [14, 15, 18]:

$$y = \begin{cases} 0, & jei \sum_j w_j x_j \leq b \\ 1, & jei \sum_j w_j x_j > b \end{cases} \tag{3}$$

Kur *y* yra išvestis, *w* yra svoriniai koeficientai, *x* yra įvestis, *b* yra slenkstinė vertė.

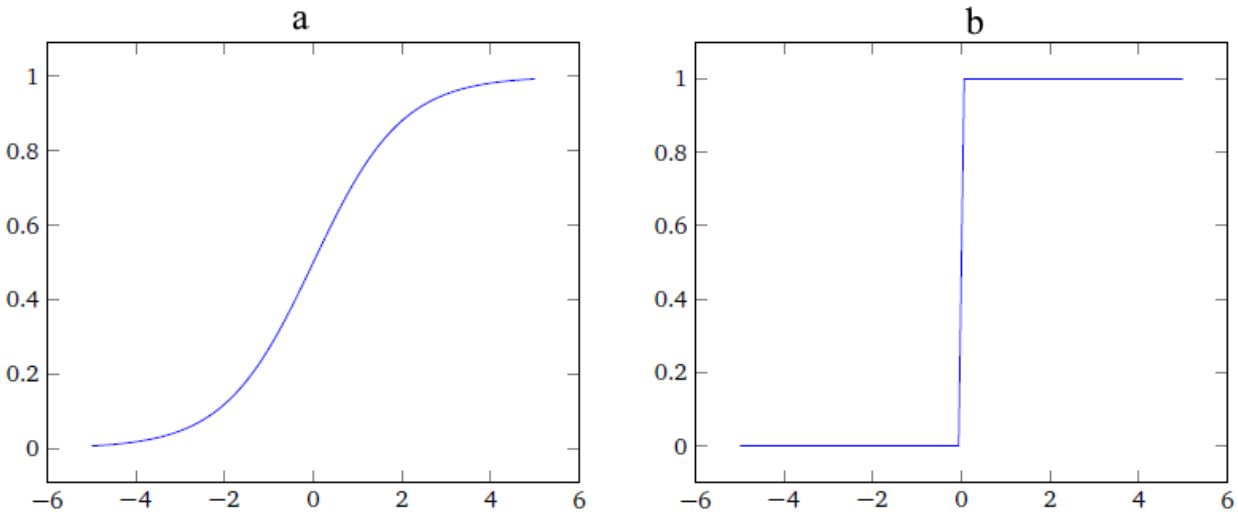
Kiek patobulintas perceptronas yra vadinamas sigmoido neuronu (angl. *sigmoid neuron*). Skirtumas šiame neurone yra aktyvacijos funkcija – vietoje Heaviside funkcijos yra naudojama sigmoido

funkcija [14, 15, 16, 18]. Tai reiškia, kad išvestyje gali būti ne tik 0 arba 1, bet visos vertės tarp 0 ir 1. Matematiškai, sigmoido neuronas skiriasi nuo perceptrono:

$$y = \frac{1}{1 + \exp(-\sum_j w_j x_j - b)} \qquad (4)$$

Šio neurono privalumas yra tas, kad mažas pokytis vienoje iš įvesčių svorinių koeficientų arba slenkstinėje vertėje sukelia mažą pokytį neurono išvestyje. Tai leidžia koreguoti mūsų funkciją taip, kad išvestis atitiktų mūsų norimus rezultatus pagal įvestį. O kaip tik šių parametrų koregavimas tam, kad būtų pagerintas modelio tikslumas (arba būtų sumažintas klaidų skaičius) ir yra mokymo procesas dirbtinių neuro tinklų srityje. Laikoma, kad neuroninio tinklo mokymo procesas yra užbaigtas, kai modeliui apdorojant naujus treniravimosi rinkinio duomenis, klaidų skaičius nemažėja (arba mažėja labai nežymiai). Matematiškai modelio motyvavimas, aiškinimas, ar buvo priimtas teisingas sprendimas ar ne, vyksta panaudojant kaštų funkciją (angl. *cost function*) arba nuostolių funkcija (angl. *loss function*) [8, 14, 15, 18]. Šios funkcijos paskirtis yra skaičiumi apibrėžti kiek modelio sprendimas buvo tikslus. Žinoma, kaštų funkcijų yra nemažai ir kiekviena iš jų gali būti naudojama tam tikruose užduotyse, tačiau viena iš dažniausiai sutinkamų yra vidutinis kvadratinis nuokrypis (angl. *mean squared error* arba *MSE*). Nors mus domina teisingų spėjimų skaičius, negalime tiesiog paimti to kaip kaštų funkcijos, kadangi tokia funkcija nebūtų glotni, ir mažais žingsniais koreguojant svorinius ir slenkstinius parametrus nepasiektume mažo pokyčio išėjime. Taigi, jeigu turime j treniravimo pavyzdžių ir žinome, kad teisingas spėjimas yra t_j, o mūsų modelio apskaičiuota vertė yra y_j, norime sumažinti šių verčių vidutinį kvadratinį nuokrypį E [14, 18]:

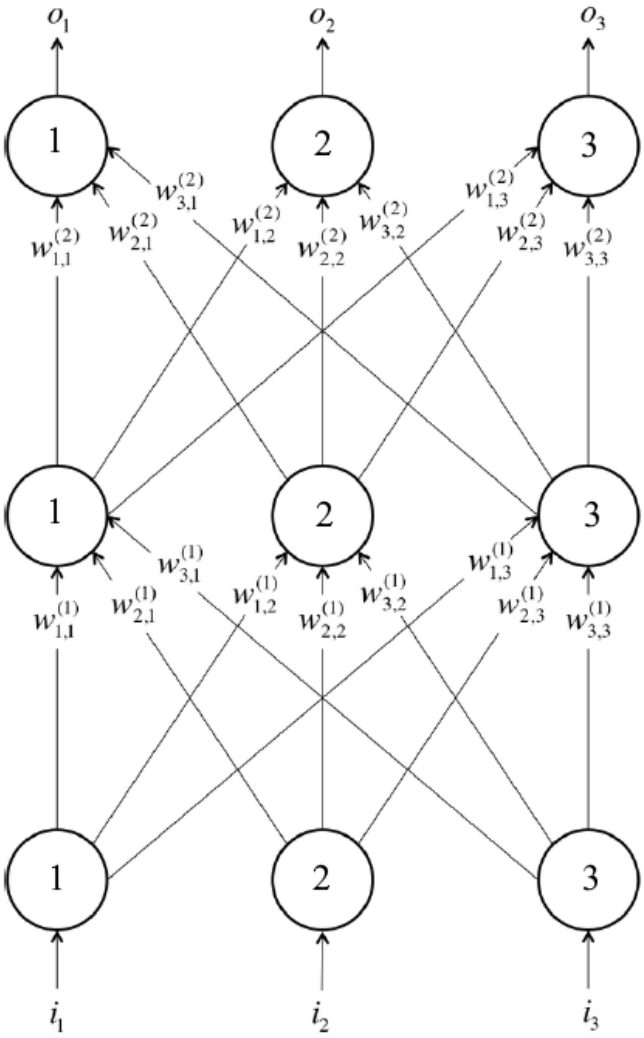
$$E = \frac{1}{2}\sum_j (t_j - y_j)^2 \qquad (5)$$



6 pav. a) Sigmoidė b) Heaviside funkcija [18].

E yra 0, kai modelis priima visiškai teisingą sprendimą. Kitaip tariant, kuo arčiau 0 yra ši vertė, tuo geriau veikia modelis. Taigi, mūsų tikslas yra sumažinti *E*, keičiant svorines bei slenkstines funkcijos vertes. Vėlgi, tam galima naudoti daugybę būdų ar algoritmų, tačiau dažnai naudojamas yra gradientinis nusileidimas (angl. *gradient descent*) [8, 14, 15, 18]. Šis algoritmas yra skirtas diferencijuojamos funkcijos lokalių minimumų suradimui [15]. Mintis yra paprasta: žengiame iš anksto apibrėžto ilgio žingsnius į priešingą gradiento pusę. Neigiamas gradientas parodo stačiausio nusileidimo kryptį.

1.4. Daugiasluoksniai neuro tinklai



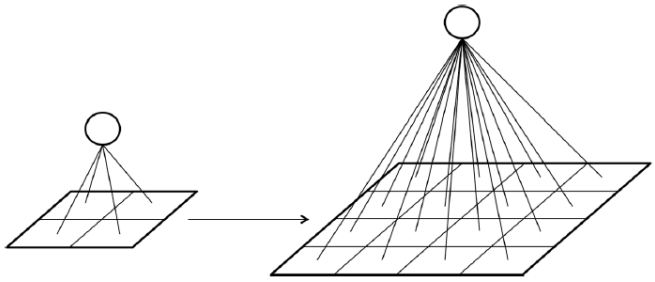
7 pav. Daugiasluoksnio tiesioginio sklidimo neuroninio tinklo pavyzdys . Šį neuro tinklą sudaro įvesties ir išvesties sluoksniai bei vienas paslėptas sluoksnis [14].

Žinant dirbtinio neuro tinklo sudedamąsias dalis ir skirtingus algoritmus reikalingus jo veikimui yra įmanoma sukonstruoti aukštesnio lygio neuro tinklą – daugiasluoksnį neuro tinklą (angl. *multilayer neural network)* [8]. Jie, kaip išduoda pavadinimas, turi ne vieną neuronų sluoksnį. Šie papildomi sluoksniai yra įvedami tarp į įvesties ir išvesties sluoksnių, ir jie yra vadinami paslėptaisiais sluoksniais (angl. *hidden layers*), kadangi skaičiavimai juose nėra matomi [15]. Paprasčiausias daugiasluoksnis neuro tinklas yra vadinamas multisluoksniu perceptronu (angl. *multilayer perceptron*). Ir perceptronai, ir multisluoksniai perceptronai priklauso tiesioginio sklidimo neuroninių tinklų (angl. *feedforward neural network*) šeimai . Tai reiškia, kad duomenys šiuose neuro tinkluose keliauja tik iš žemesnio į aukštesnį sluoksnį, nėra grįžtamojo ryšio. Daugiasluoksnio neuro tinklo pavyzdį galima pamatyti 7 pav. Žemiausias tinklo lygmuo priima įvesties duomenis, o aukščiausiame lygmenyje apskaičiuojamas galutinis tinklo atsakymas. Paslėptajame sluoksnyje kiekvienas neuronas iš praeito sluoksnio yra sujungtas su kiekvienu neuronu iš sekančio sluoksnio

(nors tai nėra būtina tokios struktūros neuro tinklams). Kiekvienas sujungimas taip pat turi savo svorinius koeficientus.

1.4.1. Konvoliuciniai neuro tinklai

Viena iš populiariausių ir tinkamiausių vaizdo atpažinimui naudojamų neuro tinklų rūšių yra konvoliuciniai neuroniniai tinklai. Pagal apibrėžimą, neuroninis tinklas yra laikomas konvoliuciniu, jeigu bent viename iš sluoksnių vietoje bendros matricų sandaugos yra naudojama konvoliucija [16].



8 pav. Sujungimų skaičiaus tarp neuronų tankis sparčiai didėja, didėjant atvaizdų dydžiui [14].

Skaitmenų atpažinimo problemoje pasinaudojant MNIST duomenų rinkiniu, kiekvieno atvaizdo dydis buvo 28×28 pikselių bei jie buvo juodai balti. Taigi įvesties sluoksnyje yra 784 neuronai, o tai reiškia dar daugiau svorinių koeficientų (kadangi kiekvienas įvesties neuronas yra sujungtas su kažkokiu skaičiuimi neuronų iš paslėptojo sluoksnio). Kompiuteriai yra pajėgūs apskaičiuoti tokį kiekį operacijų, o ir tokio dirbtinio intelekto tikslumas yra apie 98% (priklausomai nuo konkrečios neuro tinklo architektūros) [18]. Tačiau toks atvaizdų atpažinimo problemos sprendimas nėra tinkamas, kai atvaizdai didėja ir sudėtingėja. Pavyzdžiui, esant spalvotam 200×200 pikselių atvaizdui, svorinių koeficientų skaičius didėja iki 120000 ($200 \cdot 200 \cdot 3$, t. y. visų pikselių RGB, raudono, žalio, mėlyno kanalų vertės), neatsižvelgiant į įvesties neuronų sujungimų skaičių su paslėptojo sluoksnio neuronais. Konvoliuciniai neuro tinklai pasinaudoja tuo faktu, kad jie yra naudojami beveik išskirtinai darbui su atvaizdais. Neuroninio tinklo architektrūra yra supaprastinama, parametrų skaičius modelyje mažėja. Taip pat, panaudojant konvoliuciją, iš atvaizdo galima gauti įvairias objekto jame detales, jo bruožus.

Konvoliucija, kompiuterinės regos srityje, yra naudojama kaip būdas apjungti dvi matricas, dažnai skirtingų dydžių, bet tų pačių dimensijų, gaunant trečią, tos pačios dimensijos, matricą [19]. Trečiosios matricos elementai yra sudaryti iš pirmosios ir antrosios matricų tiesinės kombinacijos. Atvaizdų apdorojimo kontekste, įprastai, viena iš matricų yra tiesiog atvaizdas, o kita, daug mažesnė, vadinama branduolio filtru (angl. *kernel* arba *filter*) [8, 18, 19]. Konvoliucija yra atliekama talpinant branduolio filtrą į kiekvieną įmanomą poziciją atvaizde (arba matricoje) taip, kad matricų elementai pilnai persiklotų, tada yra atliekama skaliarinė elementų sandauga. Konvoliucijos pavyzdį galime

matyti 9 pav. Kairėje turime atvaizdą matriciniu pavidalu, kur kiekvienas matricos elementas yra atvaizdo pikselio reikšmė, per vidurį yra branduolio filtras, o dešinėje – konvoliucijos rezultatas, vadinamas bruožų žemėlapiu (angl. *feature map*) [8] arba aktyvacijų žemėlapiu (angl. *activation map*).

I_{11}	I_{12}	I_{13}	I_{14}	I_{15}	I_{16}
I_{21}	I_{22}	I_{23}	I_{24}	I_{25}	I_{26}
I_{31}	I_{32}	I_{33}	I_{34}	I_{35}	I_{36}
I_{41}	I_{42}	I_{43}	I_{44}	I_{45}	I_{46}
I_{51}	I_{52}	I_{53}	I_{54}	I_{55}	I_{56}
I_{61}	I_{62}	I_{63}	I_{64}	I_{65}	I_{66}

K_{11}	K_{12}
K_{21}	K_{22}

O_{11}	O_{12}	O_{13}	O_{14}	O_{15}
O_{21}	O_{22}	O_{23}	O_{24}	O_{25}
O_{31}	O_{32}	O_{33}	O_{34}	O_{35}
O_{41}	O_{42}	O_{43}	O_{44}	O_{45}
O_{51}	O_{52}	O_{53}	O_{54}	O_{55}

9 pav. Kairėje, mažas 6×6 atvaizdas (matriciniu pavidalu), viduryje, 2×2 branduolio filtras, dešinėje konvoliucijos rezultatas, 5×5 bruožų žemėlapis.

Jeigu turime $M \times N$ dydžio atvaizdo matricą, $m \times n$ dydžio branduolio filtro matricą, tai išvestyje turėsime $(M - m + 1) \times (N - n + 1)$ dydžio matricą. Bendra išraiška konvoliucijai atvaizdų apdorojimo srityje gali būti užrašoma [16, 19]:

$$O(i,j) = \sum_{k=1}^{m} \sum_{l=1}^{n} I(i+k-1, j+l-1) K(k,l) \qquad (6)$$

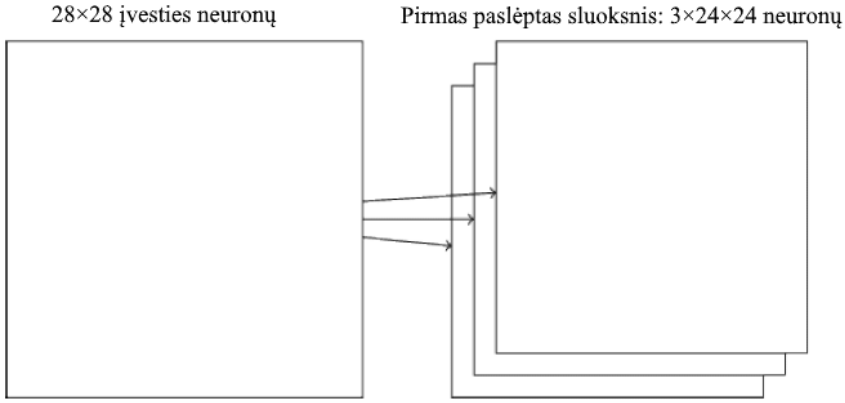
kur $1 < i < M - m + 1$, bei $1 < j < N - n + 1$. Galime užrašyti vieno bruožų žemėlapio elemento apskaičiavimą:

$$O_{34} = I_{34}K_{11} + I_{35}K_{12} + I_{44}K_{21} + I_{45}K_{22}$$

Priklausomai nuo pasirinkto filtro (konkrečių branduolio filtro matricos elementų verčių), konvoliucija gali turėti skirtingą poveikį atvaizdui. Taip pat svarbu paminėti, kad kiekvienas sujungimas tarp įvesties sluoksnio neuronų ir neuronų paslėptajame sluoksnyje turi identiškus svorinius koeficientus ir slenkstines vertes (viename bruožų žemėlapyje) [18]. Taip yra daroma dėl to, kad kiekvienas neuronas paslėptajame sluoksnyje detektuotų tuos pačius bruožus skirtinguose lokaliuose matymo srityse (angl. *local receptive fields*) [8] – regionuose, kurie konvoliucijos pagalba buvo susieti su sekančiu neuronu. Pavyzdžiui, viename bruožų žemėlapyje neuronai gali reaguoti į vertikalias linijas, kitame į horizontalias linijas.

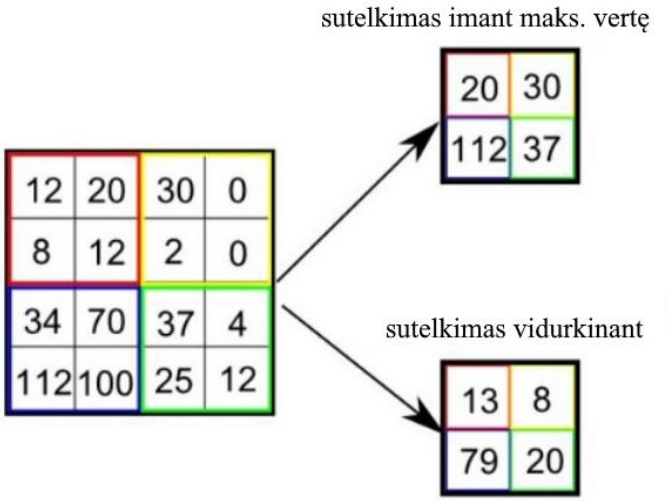
Praktikoje, neretai atvaizdai būna spalvoti, arba yra daugiakanaliai (angl. *multichannel*). Tai reiškia, kad visas atvaizdas yra sudarytas iš kelių komponentų (atvirkščiai nei juodai baltų atvaizdų atveju),

dažniausiai 3-jų. Pavyzdžiui, nuotrauka, padaryta paprasta skaitmenine kamera, turės 3 kanalus – raudoną, žalią ir mėlyną. Tokiu atveju branduolio filtro konvoliucija yra daroma su kiekvienu atvaizdo kanalu (gaunami identiško dydžio skirtingų kanalų bruožų žemėlapiai). Papildomai, konvoliuciniuose neuro tinkluose įprastai nėra naudojamas tik vienas branduolio filtras, o daugiau, dėl ko gaunamas ne vienas bruožų žemėlapis (pavyzdžiui, viena pirmųjų konvoliucinių tinklų architektūrų, LeNet-5, turėjo 6-is bruožų žemėlapius) [18]. Šiuolaikiniai dirbtiniai tinklai gali turėti dar daugiau, kartais 20 ar net 40. Literatūroje, konvoliucinių neuro tinklų architektūra, o konkrečiau bruožų žemėlapiai, paprastai atvaizduojami papildomos dimensijos pagalba, kaip pavaizduota 10 pav.



10 pav. Pasinaudojant skirtingais branduolio filtrais gaunami skirtingi bruožų žemėlapiai [18].

Dar vienas dažnai sutinkamas sluoksnis konvoliuciniuose neuro tinkluose yra sutelkimo sluoksnis (angl. *pooling layer*) [8, 14, 18, 20]. Juose tam tikra informacija iš kelių neuronų yra sutelkiama į vieną neuroną, tokiu būdu sumažinant duomenų kiekį, kuris vėliau yra perduodamas tolimesniems sluoksniams. Iš esmės, galima teigti, kad yra gaunama kondensuota bruožų žemėlapio versija. Dažniausiai naudojami du šio sluoksnio variantai – sutelkimo vidurkinant (angl. *average pooling layer*) ir sutelkimo imant maksimalią vertę (angl. *max pooling layer*) sluoksniai. Sutelkimo sluoksnis priima tam tikro dydžio (pavyzdžiui, 2×2) neuronų regioną iš prieš tai esančio sluoksnio (įprastai, iš konvoliucinio sluoksnio). Tada, priklausomai nuo sutelkimo sluoksnio tipo, yra padaromi skaičiavimai ir išvestis yra perduodama tolimesnio sluoksnio neuronui. Jeigu naudojamas sutelkimas vidurkinant, rezultatas yra regiono neuronų verčių aritmetinis vidurkis. Sutelkimo imant maksimalią vertę metu, rezultatas yra didžiausia skaitinė vertė iš regiono neuronų. Pabaigoje turime sumažintą bruožų žemėlapį, kuriame yra išsaugoti visi objekto bruožai. Tokiu būdu, tolimesniuose sluoksniuose esančių neuronų skaičius gali būti mažesnis. Tai gali teigiamai paveiktį skaičiavimo greitį, nepadarant neigiamios įtakos atvaizdo bruožų kokybei.



11 pav. Sutelkimo sluoksnio variantai [20].

1.5. Atvaizdų atpažinimo ypatumai

Atvaizdų atpažinimas yra kompiuterinės regos ir dirbtinio intelekto subkategorija. Šios srities pagrindinė užduotis yra atvaizdų analizės ir objektų aptikimo juose automatizavimas. Sistemos naudojančios šią technologiją geba identifikuoti vietas, žmones, objektus arba kitus elementus ir daryti išvadas juos analizuojant. Atvaizdų atpažinimas yra plati sąvoka, tad sistemos uždavinys yra dažnai sukonkretinamas [21, 22]:

1.5.1. Atvaizdų klasifikavimas

Atvaizdų klasifikavimas vyksta priskiriant etikėtę (arba kategoriją, klasę) visam atvaizdui. Tai reiškia, kad viename atvaizde gali būti tik vienas ieškomas objektas. Šiam atvaizdų atpažinimo uždavinio tipui galima priskirti tokius taikymus kaip Rentgeno nuotraukos klasifikavimą kaip vėžį ar ne (dviejų klasių klasifikavimas), rašytinių skaitmenų nuotraukų klasifikavimą (daugelio klasių klasifikavimas).

1.5.2. Atvaizdų klasifikavimas su lokalizacija (objektų žymėjimas)

Objektų žymėjimas yra didesnio tikslumo atvaizdų klasifikacija. Jo metu objektas atvaizde yra ne tik klasifikuojamas (t. y. jam priskirama kategorija), bet ir parodoma jo pozicija atvaizde pasinaudojant stačiakampiu (angl. *bounding box*) [8]. Šio tipo uždaviniai yra Rentgeno nuotraukų klasifikavimas kaip su vėžiu (ir parodant auglį nuotraukoje su stačiakampiu) ar be, taip pat klasifikuojant atvaizdą kaip turintį gyvūną ar ne, bei brėžiant stačiakampį aplink gyvūną, jei jis yra. Atvaizde taip pat gali būti ne vienas objektas ir jie visi turi būti pažymėti.

1.5.3. Objektų aptikimas

Objektų aptikimas yra panašus uždavinys į klasifikavimą su lokalizacija, tačiau skirtumas yra tame, kad viename atvaizde gali būti daugiau nei vienos kategorijos objektas. Tai reiškia, kad algoritmas atpažįsta, kokie objektai yra atvaizde bei juos parodo stačiakampio pagalba. Tokio uždavinio

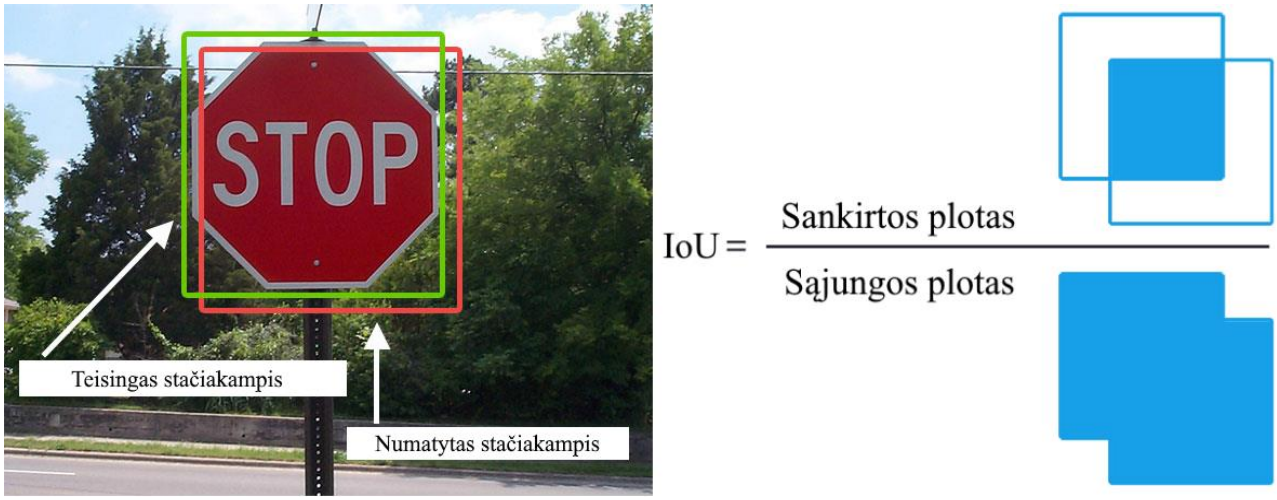
pavyzdys gali būti visų baldų aptikimas kambario nuotraukoje arba medžių, gyvūnų aptikimas kraštovaizdžio nuotraukoje.

1.5.4. Objektų segmentavimas

Objektų segmentavimas, arba semantinis segmentavimas (angl. *semantic segmentation*) [8], yra toks objektų aptikimo uždavinys, kai objektai yra ne tik surandami, bet ir yra pažymimi patys objektai, jų kraštai. Jų žymėjimas vyksta ne stačiakampiu, bet objekto pikseliams yra priskiriama kategorija. Tokiu būdu, atvaizdas gali būti padalinamas į tam tikrus segmentus, dalis. Galimi variantai, kai kategorija priskiriama kiekvienam pikseliui atvaizde arba tik ieškomiems objektams.

1.5.5. Atvaizdų atpažinimo modelių vertinimas

Kuriant dirbtinį intelektą yra labai svarbu turėti įrankius ir būdus, skirtus jo vertinimui. Sprendžiant tokius uždavinius, kur reikalinga objekto lokalizacija (t. y. ieškomas objektas yra pažymimas stačiakampiu), dažnai naudojamas tikslumo įvertinimo būdas yra sankirta padalinta iš sąjungos (angl. *intersection over union* arba *IoU*) [8, 23]. Sankirta ir sąjunga yra aibių teorijos srities veiksmai. Jeigu turime aibes A ir B, tai šių aibių sankirtos rezultatas yra nauja aibė su bendrais aibių A ir B elementais, o jų sąjungos rezultatas yra nauja aibė, kurioje yra visi skirtingi aibių A ir B elementai. Taigi, norint įvertinti modelio tikslumą, reikia turėti teisingus stačiakampius (angl. *ground-truth bounding box*), kurie dažniausiai žymimi rankiniu būdu, pasinaudojant tam skirta programa, ir numatytus stačiakampius (angl. *predicted bounding box*), kuriuos apskaičiuoja dirbtinio intelekto modelis. Turint teisingus ir numatytus stačiakampius, galime skaičiuoti sankirtos ir sąjungos santykį. Kuo jis



12 pav. Kairėje, teisingo ir numatyto stačiakampių pavyzdžiai, dešinėje IoU skaičiavimas [23].

didesnis, tuo didesnė dalis numatyto stačiakampio ploto atitinka teisingo stačiakampio plotą.

Rezultatų ir tikslumo vertinimo metu svarbu suprasti, kas yra teisingas spėjimas, o kas ne. Iš esmės, galima teigti, kad ieškomas objektas arba yra atvaizde, arba jo nėra. Jeigu modelis padaro teisingą spėjimą, kad objektas yra atvaizde, tai atsakymas yra tikrai teigiamas (angl. *true positive* arba *TP*),

tačiau jeigu modelis spėja, kad atvaizde yra objektas, nors jo nėra, tai atsakymas yra vadinamas klaidingai teigiamu (angl. *false positive* arba *FP*). Tokia pati logika taikoma, kai modelis teisingai spėja, kad objekto atvaizde nėra – atsakymas vadinamas tikrai neigiamu (angl. *true negative* arba *TN*) ir jeigu spėjama, kad objekto nėra, nors atvaizde jis yra, toks atsakymas vadinamas klaidingai neigiamu (angl. *false negative* arba *FN*). Pasinaudojus šiomis sąvokomis, galima užrašyti dar kelis modelio efektyvumą aprašančius dydžius – tikslumą (angl. *precision*) ir atkūrimą (angl. *recall*) [24]. Tikslumas atsako į klausimą "Kokia dalis spejimų, kad atvaizde yra objektas buvo teisinga?", tuo tarpu atkūrimas atsako į klausimą "Kokia dalis tikrų objektų buvo teisingai identifikuota?". Šias sąvokas galima užrašyti formulių pagalba:

$$\text{Tikslumas} = \frac{TP}{TP + FP} \tag{7}$$

$$\text{Atkūrimas} = \frac{TP}{TP + FN} \tag{8}$$

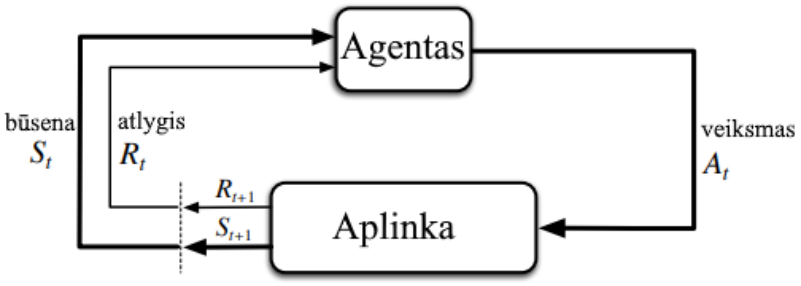
Kadangi vaizdo įrašai ir srautai yra sudaryti iš atvaizdų, tos pačios taisyklės galioja ir tokiems taikymams. Dirbant su vaizdo medžiaga, labai svarbus rodiklis yra kadrų per sekundę skaičius (angl. *frames per second*). Jis parodo kiek atvaizdų arba kadrų gali būti rodoma per sekundę. Į šį laiką įeina ir visi modelio daromi skaičiavimai. Įprastai, 30 kadrų per sekundę užtenka, kad žmogaus smegenys nepastebėtų trukčiojimo ir vaizdo srautas atrodytų naturaliai.

1.6. Skatinamasis mokymas

Palyginus su kitais mašininiame mokyme taikomais metodais, skatinamojo mokymo išskirtinė savybė yra ta, kad mokymosi procesas yra tiesmukiškai nukreiptas į tikslo pasiekimą esant sąveikai tarp agento ir aplinkos [25]. Tokio tipo uždaviniuose agentui tenka išmokti kokia žingsnių seka – kaip išmokti surišti konkrečią situaciją su konkrečiais žingsniais – padeda pasiekti maksimalų, skaičiumi aprašomą, atlygį. Skatinamąjį mokymą, iš esmės, galima apibūdinti trimis pagrindinėmis savybėmis: pirmą, problemos, sprendžiamos šių algoritmų pagalba, turi būti uždarojo ciklo problemos, kadangi dabartiniai algoritmo sprendimai gali paveikti vėlesnias įvestis, antrą, mokymosi sistemai nėra tiksliai pasakoma, kokius sprendimus priimti, atvirkščiai nei kitose mašininio mokymosi srityse, o ji turi pati išsiaiškinti, kokie veiksmai duoda didžiausią atlygį, juos išbandant, ir trečią, kad atlikti veiksmai daro įtaką ne tik šiuo metu gaunamam atlygiui, bet ir ateities situacijoms, o ko pasekoje, ir ateities atlygiams.

Taikant skatinamojo mokymo modelius, dažnai tenka spręsti problemas, kurios kitose mašininio mokymo srityse neatsiranda. Viena iš tokių problemų yra tyrinėjimo ir išnaudojimo balansas (angl. *exploration and exploitation trade-off*) [8]. Tam, kad gautų daug teigiamo atlygio, modelis privalo

teikti pirmenybę tiems veiksmams, kurie buvo išbandyti praeityje ir atnešė daugiausiai atlygio. Tačiau prieš žinant apie tokius veiksmus, modelis turi apie juos kažkokiu būdu sužinoti, išbandyti naujus, dar nebandytus veiksmus. Agentas turi išnaudoti savo žinias norėdamas gauti didesnį atlygį, bet ir privalo tyrinėti naujus scenarijus, kad turėtų geresnę veiksmų bazę ateities sprendimams. Čia atsiranda dilema – nei tyrinėjimas, nei išnaudojimas negali būti išskirtinai prioretizuojami agentui, nes uždavinys nebus įvykdytas. Geriausiu atveju, modelis išbando didelį kiekį įvairių veiksmų bei palankiai vertina tuos, kurie atnešė didžiausią atlygį. Tačiau skatinamojo mokymo požiūris turi ir privalumų. Taikant tokį modelį, uždavinys yra suformuluojamas tokiu būdu, kad jo išsprendimas apima visą problemą (o ne tik tam tikrą jos dalį, kaip prižiūrimo ar neprižiūrimo mokymo atvejais); agentą, bandantį pasiekti konkretų tikslą sąveikaujant su neapibrėžta aplinka, bei jai darant įtaką. Agentas, arba, kitaip, mokinys arba sprendimų priemėjas, nuolat sąveikauja su aplinka, į kurią, pagal apibrėžimą, įeina viskas, kas nėra agentas. Tokiu būdu, agentui renkantis ir atliekant veiksmus, o aplinkai į juos reaguojant, aplinka keičiasi, ir agentas atsidūria vis kitose situacijose. Iš aplinkos ateina ypatinga skaitinė vertė, vadinama atlygiu, kurią agentas bando maksimaliai padidinti. Tiksliau, tarp agento ir aplinkos kas tam tikrą diskrečią vertę t = 0, 1, 2, 3, ... įvyksta sąveika. Kiekvieną laiko žingsnį t agentas gauna aplinkos reprezentaciją, arba būseną, $S_t \in \boldsymbol{S}$, kur $\boldsymbol{S}$ yra visų įmanomų būsenų rinkinys, ir to pagrindu pasirenka veiksmą $A_t \in \boldsymbol{A}(S_t)$, kur $\boldsymbol{A}(S_t)$ yra visų įmanomų veiksmų rinkinys esant S_t būsenai. Po t+1, agentas gauna tam tikrą atlygį $R_{t+1} \in \boldsymbol{R} \subset \mathbb{R}$, kur $\boldsymbol{R}$ yra visų įmanomų atlygių rinkinys. Tuo pačiu metu agentas atsiranda kitoje, S_{t+1}, būsenoje ir visas procesas kartojasi (13 pav.). Kiekvienos laiko vertės metu agentas susieja būseną su tikimybe pasirinkti kiekvieną įmanoma veiksmą. Šis surišimas yra vadinamas agento strategija (angl. *policy*) [8] ir yra žymimas π_t:



13 pav. Agento ir aplinkos

$$\pi(A_t = a | S_t = s) \qquad (9)$$

Agentai, esant skirtingiems skatinamojo mokymo metodams, turės skirtingus strategijos pokyčius. Agento tikslas yra taip pakeisti savo strategiją, kad gaunamas atlygis būtų kuo didesnis. Žinoma, toks modelis, požiūris į skatinamąjį mokymą gali būti pratestas ne tik diskrečioms laiko vertėms. Bendrai, agento tikslas arba paskirtis formaliai apibūdinamas kaip tam tikras atlygio signalas iš aplinkos jam.

Kiekvieną laiko momentą atlygis yra skaičius $R_t \subset \mathbb{R}$. Agento užduotis yra pasiektį maksimaliai didelį pilną atlygį, t. y. ne atlygį konkrečiu laiko momentu, o visų atlygių sumą. Tokia mintis vadinama atlygio hipoteze (angl. *reward hypothesis*) [25, 26].

Dažnai yra svarbu tiksliai atskirti, kas yra agento, o kas aplinkos dalis. Ši riba neretai atrodo neintuityvi, kadangi, pavyzdžiui, roboto ar gyvūno kūno atvejais, ši riba nėra paprasčiausia riba tarp fizinio kūno ir aplinkos. Įprastai, šis atskyrimas brėžiamas arčiau agento – roboto varikliai, mechaninės jungtys ir įvairūs sensoriai yra priskiriami prie aplinkos. Atlygių skaičiavimas, nors ir vyksta dirbtinėje mokymosi aplinkoje, agento viduje, yra laikomi ne jo dalimi. Bendrai, viskas, kas negali būti pakeista bet kuriuo laiko momentu, savavališkai, yra priskiriama prie aplinkos. Ne viskas aplinkoje agentui yra nežinoma. Atvirkščiai, dažnai agentui yra žinoma daugelis dydžių iš aplinkos, atlygių skaičiavimo funkcija, o kartais net visi aplinkos kintamieji. Tačiau atlygio skaičiavimas yra visada laikomas išoriniu agento atžvilgiu, kadangi ši funkcija aprašo visos sistemos elgėsį ir jos veiksmus, tad ji negali būti keičiama savavališkai.

1.6.1. PPO

Vienas iš neseniai pasirodžiusių skatinamojo mokymo algoritmų, PPO, arba angl. *Proximal Policy Optimization*, greitai tapo vienu iš populiariausių metodų taikomų šioje srityje. Autorių ir kitų mokslininkų nuomone, jis yra labiau efektyvus (arba bent panašiai evektyvus į kitus) ir patikimas, bet tuo pačiu lengviau suprantamas ir pritaikomas nei kiti skatinamojo mokymo algoritmai [27, 28]. Šis OpenAI komandos sukurtas algoritmas yra didelės strategijos gradiento (angl. *policy gradient*) algoritmų šeimos dalimi. Šie metodai tiesiogiai modeliuoja ir optimizuoja strategijos funkciją. Strategija dažniausiai yra modeliuojama parametrų funkcijos θ atžvilgiu. Taigi funkcijos vertė priklauso nuo pačios strategijos. Strategijos funkciją galima apibrėžti kaip [29, 30]:

$$J(\theta) = \sum_{s \epsilon \boldsymbol{S}} d^{\pi}(s) V^{\pi}(s) = \sum_{s \epsilon \boldsymbol{S}} d^{\pi}(s) \sum_{a \epsilon \boldsymbol{A}} \pi_{\theta}(a|s) Q^{\pi}(s,a) \tag{10}$$

kur $d^{\pi}(s)$ yra pastovus Markovo grandinės skirstinys, $V^{\pi}(s)$ yra būsenos s vertė esant strategijai π, $Q^{\pi}(s, a)$ veiksmo-vertės funkcija esant strategijai π. Turint šią funkciją, užduotis tampa aiški – surasti tokią θ vertę kuri grąžintų maksimalią funkcijos vertę esant π_{θ} strategijai. Tam ir yra naudojamas gradientinis pakilimas arba nusileidimas. Įprastai, toks skaičiavimas būtų sudėtingas, tačiau strategijos gradiento metodas tą skaičiavimą supaprastina [29, 30]:

$$\begin{aligned} \nabla J(\theta) &= \nabla \sum_{s \epsilon \boldsymbol{S}} d^{\pi}(s) \sum_{a \epsilon \boldsymbol{A}} \pi_{\theta}(a|s) Q^{\pi}(s,a) \\ &\propto \sum_{s \epsilon \boldsymbol{S}} d^{\pi}(s) \sum_{a \epsilon \boldsymbol{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s,a) \end{aligned} \tag{11}$$

Tačiau be jokių papildomų operacijų, treniruojant modelius tokiu būdu daromi žingsniai yra itin dideli ir dažnai optimalios vertės yra praleidžiamos. Todėl bandyta gradiento verčių mažėjimo ar didėjimo žingsnius mažinti įvairiais būdais, dėl ko ir atsirado PPO.

Šiame algoritme yra naudojama dažniausiai taikoma gradiento įvertinimo išraiška [27, 29, 31]:

$$\theta_{t+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_{senas}}}{E} [J(\theta)] \tag{12}$$

Čia *E* žymi baigtinio skaičiaus bandinių empirinį vidurkį. Tuomet yra įvedamas pagalbinis dydis, žymintis senos ir naujos strategijų tikimybių santykį:

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{senas}}(a|s)} \tag{13}$$

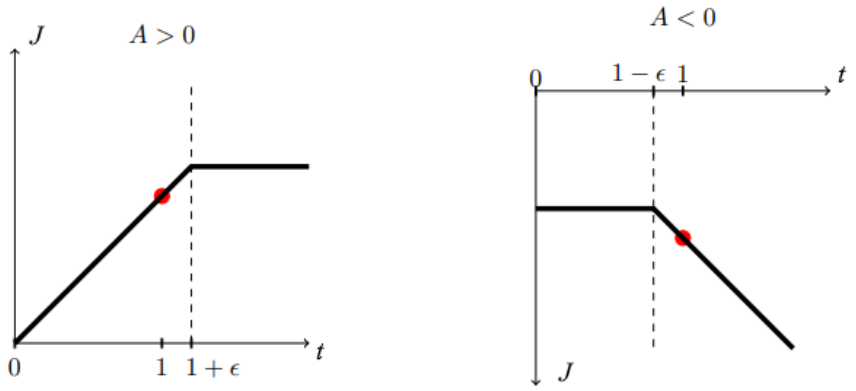
Ir toliau $J(\theta)$ apibrėžiamas kaip:

$$J(\theta) = min(r(\theta)A^{\pi_{\theta_{senas}}}(s,a), g(\epsilon, A^{\pi_{\theta_{senas}}}(s,a))) \tag{14}$$

Kur $A(a, s) = Q(s,a) - V(s)$ yra pranašumo funkcija (angl. *advantage function*) bei:

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A \leq 0 \end{cases} \tag{15}$$

ϵ - hiperparametras, kontroliuojantis, kaip stipriai nauja strategija gali nutolti nuo senos vieno žingsnio metu.



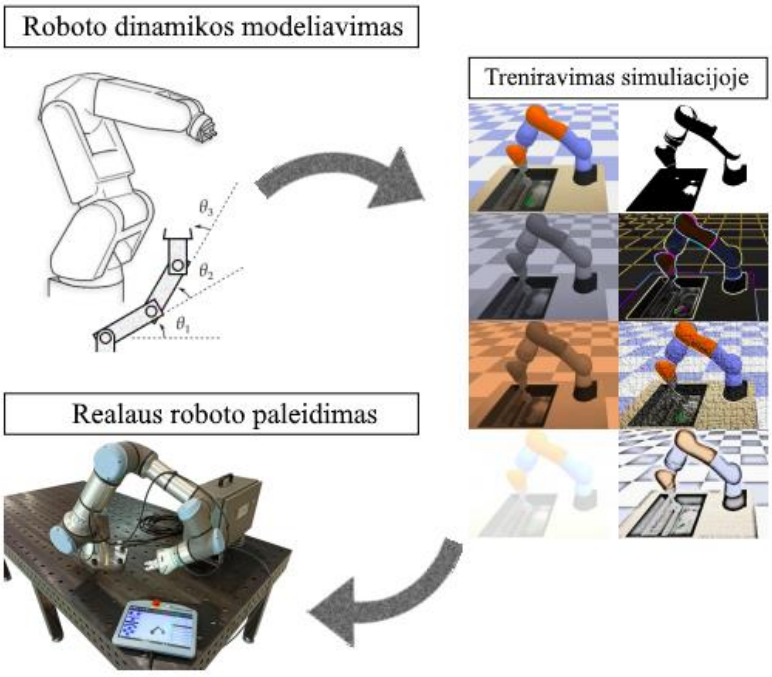
14 pav. Grafikai, rodantys *J* kitimą po vieno laiko tarpo *t*. Kairėje, pranašumas teigiamas, dešinėje pranašumas neigiamas. Raudonas apskritimas reiškia pradžios tašką optimizavimui, t. y. $t = 1$. [27]

Visa tai leidžia gradientinio nusileidimo ar pakilimo žingsnius "kirpti" (angl. *clip*) ir tokiu būdu kontroliuoti jų dydį. Jeigu žingsnis yra per didelis, t. y. pasirinktas veiksmas buvo daug geresnis nei tikėtasi arba tas veiksmas turi itin didelę tikimybę būti pasirinktam dabartinėje strategijoje, jis pasieks tiesinę grafiko dalį, kurioje šio pokyčio poveikis visai strategijai pasiekia maksimalią vertę (14 pav.)

ir toliau nedidėja. Algoritmas veikia analogiškai esant blogesniam nei tikėtasi veiksmui arba veiksmui su itin maža tikimybe būti pasirinktam. Dėl to pasiekiamas didesnis tikslumas optimizuojant funkcijos parametrus esant tam tikrai būsenai bei hiperparametrams.

1.7. Robotinių mechanizmų treniravimo ypatumai

Skatinamasis mokymas yra plačiai taikomas įvairiose srityse – kompiuterinėse sistemose, finansuose, transporte, žaidimuose, robotikoje [25, 32]. Tačiau šių metodų taikymas robotikoje siejamas su nemažais iššūkiais, kadangi taikant dirbtinį intelektą šioje srityje stipriai išryškinamos jau egzistuojančios problemos. Fiziškai treniruojant modelį, t. y. keičiant agento strategiją realybėje, o ne skaitmeninėje erdvėje, problemų yra apstu: roboto būsenos bei veiksmai yra iš prigimties tolygūs, tad tenka apsispręsti, ar taikyti priimamų veiksmų diskretizavimą, ar funkcijų aproksimaciją; neretai veiksmų ir būsenų dimensijos yra itin didelės (dėl dažnai didelių robotų laisvės laipsnių skaičių), dėl ko skaičiavimų trukmės gali būti per ilgos norint efektyviai pritaikyti modelį [32, 33]. Kadangi robotų konstrukcijos būna sudėtingos ir kompleksiškos, tokių sistemų treniravimas ir taikymas reikalauja žmogaus įsitraukimo, remonto ir priežiūros darbų. Žinoma, taip pat yra sudėtinga pašalinti realybėje egzistuojančius triukšmus ir neapibrėžtumus.



15 pav. Abstraktus žvilgsnis į simuliacijos pernešimo į realybę procesą [34].

Dalies šių problemų įtaką galima sumažinti, arba kartais pilnai panaikinti, pasinaudojant simuliacijomis [32, 33, 34]. Toks dirbtinio intelekto mokymo metodas vadinamas angl. *sim2real*. Idealiu atveju, aplinkos parametrai bei pats agentas, jo fiziniai parametrai, būtų sumodeliuoti tokiu būdu, kad pilnai atspindėtų realybę, bet simuliacijos iš prigimties yra netobulos ir sudėtingai kalibruojamos [35, 36]. Taigi pasiekti idealų tikslumo lygį nėra įmanoma, o tai atvejais, kai tai yra

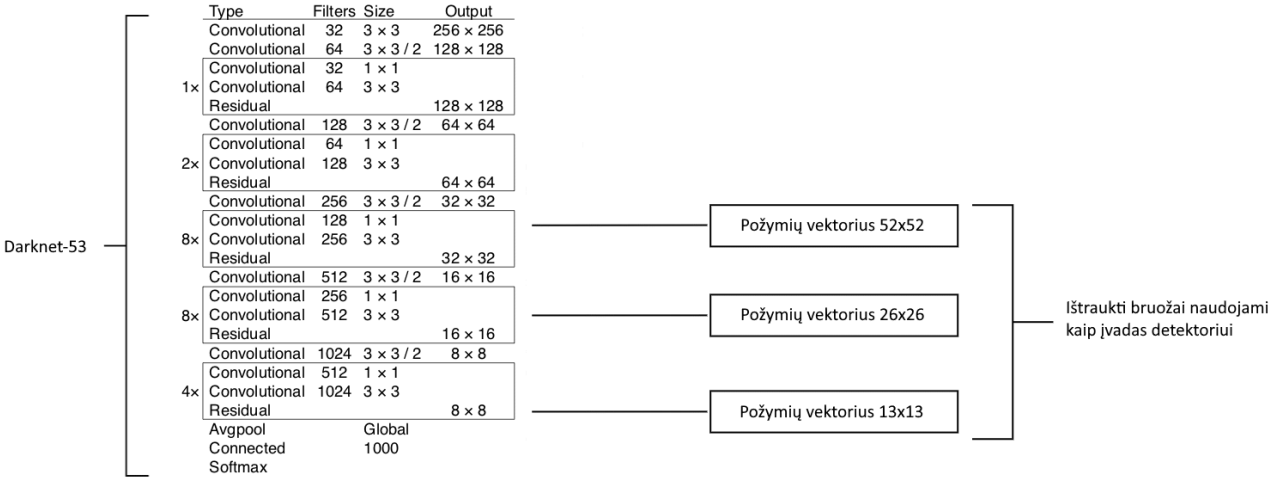
įmanoma, reikalingas milžiniškas kiekis duomenų. Maža klaida dėl nepakankamo aplinkos arba agento detalumo simuliacijoje gali greitai sukelti nuokrypį nuo realybės, nes dėl didelio žingsnių (o tuo pačiu ir strategijos keitimų) kieko, mažos klaidos susideda. Tokiais atvejais, kai modelis pilnai neatitinka realių sąlygų, tenka įvesti papildomus pataisymus taikant modelį. Nepaisant to, treniravimas simuliacijos pagalba lieka patenkinamas būdas pasiekti tokį tikslą. *Sim2real* metodas yra pakankamai pigus, saugus ir, svarbiausia, veikiantis būdas dirbtinio intelekto modelių treniravimui [35, 36]. Priklausomai nuo roboto tipo ir paskirties, formuluojama užduotis ir atlygio funkcija gali žymiai skirtis. Atlygio funkcijos suformulavimas yra labai atsakingas ir kruopštaus planavimo reikalaujantis žingsnis taikant skatinamąjį mokymą robotikoje. Nors iš pirmo žvilgsnio gali atrodyti nereikšmingai, šis sprendimas lemia visą modelio elgesį ir jo vystymąsi. Deliojant atlygio funkciją atrodo naturalu atlyginti algoritmą už laimėjimą arba pasiektą tikslą. Tačiau tokia abstrakti ir sunkiai pasiekiama sąlyga gali likti nepasiekta per visą treniravimo ir taikymo laikotarpį. Norint pasiekti gerą rezultatą, reikia skirti atlygį už tam tikrus tarpinius rezultatus, tam, kad mokymosi procesas būtų vedamas tesinga kryptimi [33, 34]. Duodant atlygius už tam tikrus veiksmus, reikia įsitikinti, kad tai neprives prie per dažno to veiksmo kartojimo, kas gali fiziškai gadinti patį mechanizmą, nekalbant apie užduoties nepadarymą. Taip pat skatinamojo mokymo algoritmai yra pagarsėję savo sugebėjimu nustebinti savo kūrėją netikėtais problemų sprendimo būdais.

1.8. Įrankiai ir bibliotekos

1.8.1. YOLOv3 (AlexeyAB versija) – Github

YOLOv3 (*You Only Look Once*, v3) yra trečia YOLO konvoliucinio neuro tinklo iteracija, naudojama vaizdų atpažinime [37]. Pirmoji YOLO versija buvo išleista dar 2015 metais ir nuo to laiko tapo vienu geriausius rezultatus rodančių atvaizdų atpažinimo neuro tinklų [37]. Pirmoji šio algoritmo versija buvo išskirtinė tuo, kad naudojo vieną konvoliucinį neuro tinklą (iki šio algoritmo, įprasta praktika buvo ieškoti objektų tik dominančioje srityje (angl. *region of interest*) [8], kas reikalavo ne vieno neuro tinklo viename modelyje) [38]. Taigi, pirmasis modelis turėjo porą privalumų, lyginant su kitais tuometiniais modeliais: pirmą, algoritmo skaičiavimo greitis buvo labai didelis, antrą, šis modelis gebėjo suprasti atvaizdo kontekstą bei gebėjo gerai apibendrinti [39]. Tačiau YOLOv1 turėjo ir neigiamų savybių: jis buvo pakankamai netikslus bei, palyginus su kitais modeliais, blogiau lokalizavo objektus atvaizde. Iš architektūros pusės, visi YOLO neuro tinklai naudoja DarkNet sistemą (angl. *framework*) kaip bruožų detektorių [25, 38, 40]. DarkNet, iš esmės, yra daug konvoliucinių sluoksnių turintis neuro tinklas. Objektų aptikimo srityje šis sluoksnis dar vadinamas stuburu (angl. *backbone*). Pats DarkNet nepriima sprendimo, kur objektas yra atvaizde, ši sistema tik išskiria atvaizdo bruožus ir praneša, koks objektas yra atvaizde. YOLOv3 naudoja DarkNet-53 versiją, kurios architektūrą galima pamatyti 16 pav. Iš šio tinklo gaunami 3 skirtingo dydžio bruožų

žemėlapiai, kurie yra naudojami kaip įvadas YOLO detektoriaus neuro tinklui – vadinamam galva (angl. *head*). Architektūroje yra 53 konvoliuciniai tinklai, ir po kai kurių tinklų turime liekanų sluoksnius (angl. *residual layer*) [8] (kurie padeda bendram neuro tinklo efektyvumui). Visas modelis buvo iš anksto treniruotas kūrėjų panaudojant Imagenet duomenų rinkinį, optimizuojant jį objektų atpažinimo uždaviniui.



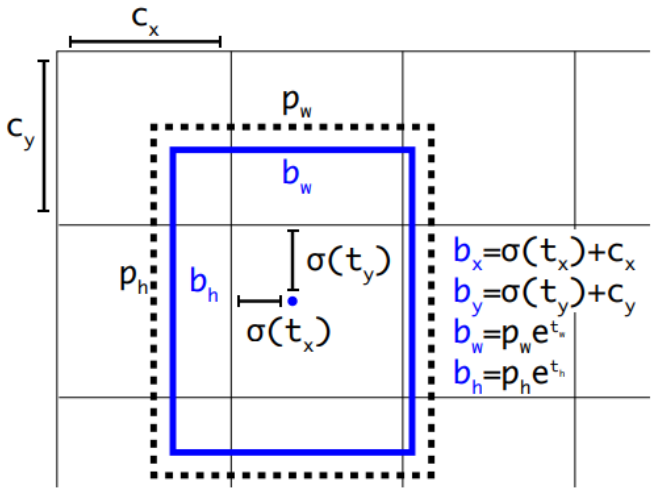
16 pav. Darknet-53 architektūra [25].

YOLOv3 detektorius, kaip minėta anksčiau, priima 3 skritingų skalių atvaizdo bruožų žemėlapius. Kiekvienas atvaizdas yra padalinamas į $S \times S$ dydžio tinklelį (angl. *grid*). Kiekvienas tinklelio langas numato B stačiakampių ir įsitikinimo vertę (angl. *confidence score*). Ši įsitikinimo vertė atspindi kiek modelis yra įsitikinęs, kad objektas yra numatytame stačiakampyje bei kiek tikslus yra stačiakampis (formaliai, darbo autoriai tai pristato kaip $P(objektas) \cdot IoU^{teisingas}_{numatytas}$) [39]. Jeigu objekto centras patenka į tinklelio langą, šis langas yra atsakingas už to objekto detektavimą ir tokiu atveju, numato: B atraminius stačiakampius (angl. *anchor boxes*) [8], kurie turi 4 koordinates, t_x, t_y, t_w, t_h, objekto buvimo tikimybę stačiakampyje (angl. *objectness score*) bei klasių buvimo stačiakampyje tikimybes (angl. *class probabilities*) [25]. Kadangi numatytos koordinatės yra skaičiuojamos tinklelio lango atžvilgiu, jas reikia konvertuoti į koordinačių vertes viso atvaizdo atžvilgiu ir tam naudojamos formulės, parodytos 17 pav. Išvestyje gauname sekančių dimensijų tenzorių (C yra klasių kiekis):

$$S \times S \times (B \cdot 5 + C)$$

Šiame darbe buvo naudojama YOLOv3 *AlexeyAB* algoritmo atšaka. Šioje saugykloje yra daug smulkių pataisymų pačiame algoritme bei patobulinta jo taikymo specifika [41]. Iš esmės, viskas, ko reikia šio modelio treniravimui yra treniravimo duomenų rinkinys bei pakankamai galingas kompiuteris. Saugykloje egzistuoja konfigūracinis failas, kuriame yra nustatomi įvairūs modelio treniravimo parametrai – hiperparametrai. Taip pat atskirame faile yra pačio YOLOv3 modelio architektūra (DarkNet-53 bei detektorius). Kadangi DarkNet-53 yra parašytas C ir CUDA

programavimo kalbomis, norint pasinaudoti modeliu reikia jį sukompiliuoti (angl. *build*) – gauti vykdomuosius failus. Turint vykdomąjį failą bei treniravimo duomenis galima pradėti modelio apmokymą.



17 pav. Koordinačių numatymas YOLOv3 modelyje. Čia: c_x ir c_y tinklelio lango dydis, p_w ir p_h atraminio stačiakampio dydis, t_x, t_y, t_w, t_h yra atraminių stačiakampių koordinatės, b_x, b_y, b_w, b_h numatyto stačiakampio koordinatės ir dydis, o σ žymi sigmoidę [25].

Nepriklausomai nuo YOLO versijos, treniravimo duomenys turi būti paruošti konkrečiu formatu. Kiekvienam atvaizdui (*png* formatu) turi būti sukurtas to pačio pavadinimo tekstinis failas (*txt* formatu) su sekančiu teisingų stačiakampių ir klasių žymėjimu:

<objekto klasė> <x> <y> <plotis> <aukštis>

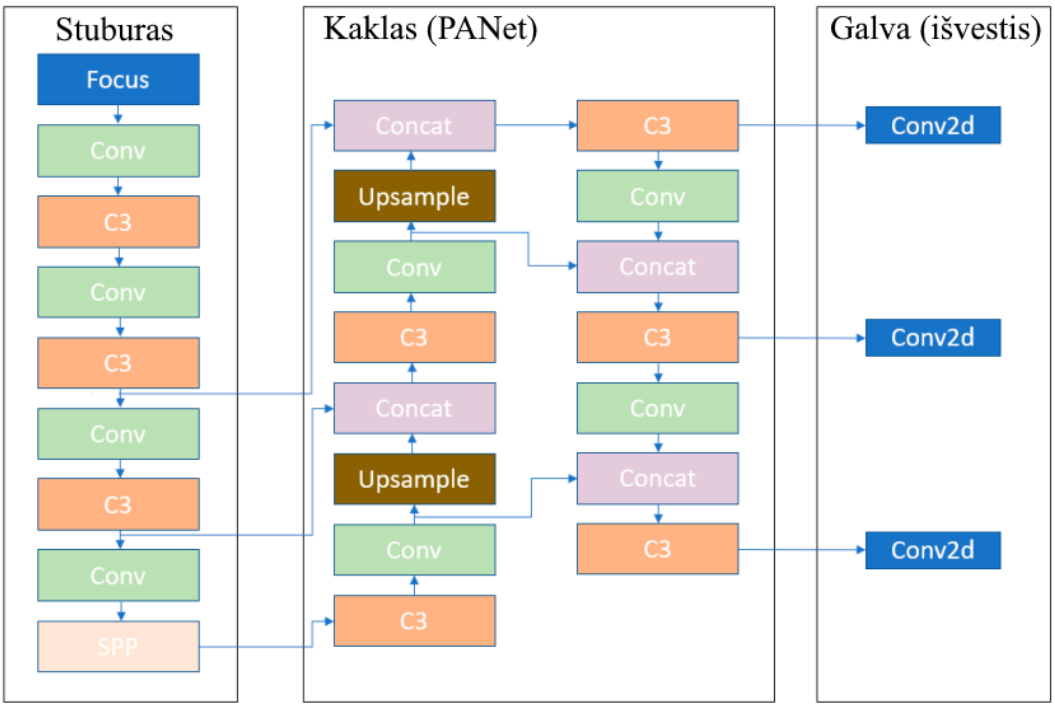
Kur <objekto klasė> yra sveikas skaičius nuo 0 iki (klasių skaičius - 1). Kiekvienas skaičius žymi konkrečią objekto klasę. <x>, <y>, <plotis>, <aukštis> yra slankiojo kablelio skaičiai, ∈(0.0 iki 1.0]. <x> ir <y> žymi santykines (su atvaizdo dydžiu) stačiakampio centro koordinates, <plotis> ir <aukštis> yra viso atvaizdo dydžiai. Tarkime turime atvaizdą pavadinimu atvaizdas1.png, tokiu atveju stačiakampių žymėjimo failas bus atvaizdas1.txt ir atrodys taip (atvaizde esant 1 objektui):

1 0.716797 0.395833 0.216406 0.147222

1.8.2. YOLOv5

YOLO algoritmai buvo tobulinami nuo pat pirmosios versijos pasirodymo. Šio darbo rašymo metu naujausia šių algoritmų šeimos versija yra v5. Palyginus su ankstesnėmis versijomis, nors pagrindinė algoritmo mintis nepasiketė, YOLOv5 implementacijoje galima rasti pakankamai pastebimų skirtumų. Didžiausias skirtumas yra Pytorch naudojimas vietoje Darknet architektūros, skaičiavimo greitis bei tikslumas [42, 43]. Šio algoritmo stuburas yra CSPDarknet53, kuris yra letėsnis bet

užimantis mažiau vietos diske bei tikslesnis nei Darknet-53. Tarp stuburo ir galvos yra pridedamas papildomas sluoksnis – kaklas (angl. *neck*), kuris pagreitina informacijos judėjimą tarp apatinių ir viršutinių sluoksnių. Toks metodas yra vadinamas kelių susibūrimo tinklas (angl. *path aggregation network* arba *PANet*). Papildomai jame yra naudojamas bruožų piramidės tinklas (angl. *feature pyramid network*) kuris pagerina žemo lygmens bruožų atradimą. Viso modelio galva, palyginus su ankstesnėmis versijomis, lieka nepakitusi – išvestyje gaunami 3 bruožų žemėlapiai. Tai leidžia nesudėtingai keisti modelius tarpusavyje. Taikymo metu atvaizdas yra paduodamas į CSPDarknet53 bruožų ištraukimui bei tuo pačiu metu į PANet bruožų suliejimui. Paskutinis YOLO sluoksnis generuoja išvestį. YOLOv5 architektūrą galima pamatyti 18 pav.



18 pav. YOLOv5 architektūros schema [42].

1.8.3. OpenCV, imutils, PIL

OpenCV (*Open Source Computer Vision Library*) yra daygybę kompiuterinės regos algoritmų implementacijų apimanti atviro kodo biblioteka [44]. Dažnai naudojama vaizdų apdorojimo arba išankstinio vaizdų apdorojimo tikslais, video analizei, dirbtinio intelekto modelių taikymui. Taip pat yra siūloma grafinė vartotojo sąsaja. Darbe naudojamos funkcijos:

- *readNetFromDarknet* – leidžia nuskaityti YOLO modelio treniravimo metu sugeneruotą dirbtinio neuro tinklo svorinių koeficientų failą (*weights* failą). Grąžina objektą, kuris naudojamas kaip kreipinys į dirbtinio intelekto modelį.

- *blobFromImage* – sukuria specialią duomenų struktūrą, vadinama blob (pati struktūra yra NCHW tipo), kurioje yra laikomi atvaizdo duomenys. Būtent ši duomenų struktūra yra naudojama operacijoms su modeliu.
- *forward* – šis metodas iškviečiamas ant kreipinio į dirbtinio intelekto modelį. Šiam metodui yra perduodami galutinio YOLO tinklo sluoksnio pavadinimai (galimos objektų kategorijos). Grąžina visus objektus, kuriuos randa neuro tinklas (nepriklausomai nuo to, kiek įsitikinęs yra modelis). Tai reiškia, kad vienam objektui bus sukurtas ne vienas stačiakampis žymėjimas.
- *NMSBoxes* – šios funkcijos pagalba taikomas nemaksimalaus apjungimo algoritmas (angl. *non-maximum suppression algorithm*). Jo pagalba yra išrenkamas numatytas stačiakampis, turintis didžiausią *IoU* [45]. Tokiu būdu lieka vienas, tiksliausias numatytas stačiakampis.

imutils yra atviro kodo biblioteka, kurioje galima rasti didelį skaičių pagrindines vaizdo apdorojimo operacijas palengvinančias funkcijas [46]. Darbe naudojama funkcija:

- *VideoStream* – funkcija, naudojama sąsajai tarp Python programos ir prie kompiuterio prijungtos kameros sukurti. Perduoda kameros gaunamus kadrus į Python programą.

PIL (*Python Imaging Library)* yra Python biblioteka, skirta darbui su atvaizdais pačioje Python aplinkoje. Darbe naudotas modulis [47]:

- *Image* – šiame modulyje galima rasti to pačio pavadinimo klasę, kuri sukuria norimo atvaizdo reprezentaciją *PIL* bibliotekos formatu. Pačiame modulyje yra nemažas skaičius funkcijų, skirtų darbui su atvaizdo reprezentacija: galima gauti atvaizdo dydį, kiekvieno pikselio spalvos vertę.

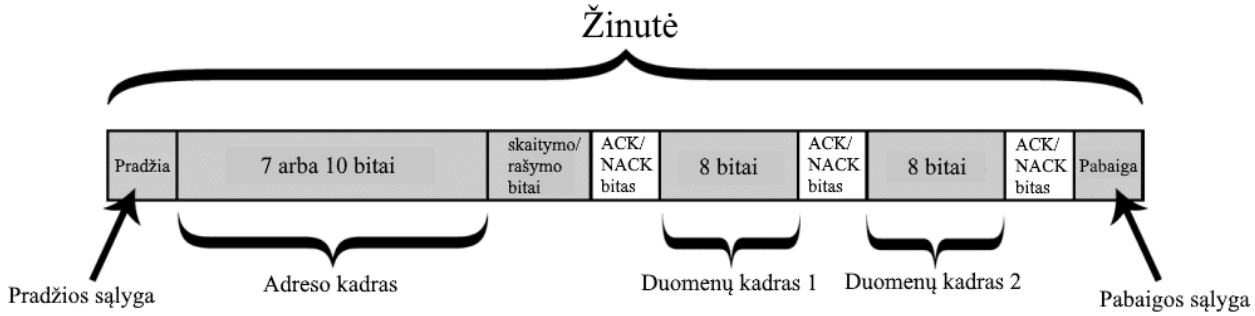
1.8.4. Arduino, Arduino bibliotekos, I2C, PWM ir UART

Arduino yra atviro kodo elektronikos platforma paremta technine bei programine įranga [48]. Šio projekto valdikliai yra pajėgūs gauti įvestį iš tam tikrų sensorių (šviesos ar dregmės sensorių, mygtukų itt.) ir paversti tai į išvestį (įjungti šviesos diodą, atvaizduoti žodį ekrane arba pasukti variklį). Visa tai daroma perduodant instrukcijas į mikrovaldiklį, naudojant Arduino programavimo kalbą bei Arduino programavimo aplinką. Vienas iš Arduino privalumų yra didelis kiekis nesudėtingai pritaikomų atviro kodo bibliotekų, kurios gali būti naudojamos atskirai arba kartu su tam tikra technine įranga.

Priklausomai nuo Arduino valdiklio modelio, įvesties ir išvesties kontaktų skaičius gali skirtis. Robotinėms paskirtims dažniausiai esamo skaičiaus neužtenka, todėl yra naudojami pagalbiniai įrankiai suvaldyti didelį komunikacijos sąsajų kiekį. Vienas iš tokių įrankių yra Adafruit PCA9685 plokštė bei kodo bazė skirta komunikacijai tarp variklių ir Arduino bei servo variklių ir Arduino. Šiai

plokštei reikia mažiau nei penkių kontaktų, dvejais iš kurių vyksta komunikacija tarp Arduino ir Adafruit plokščių per I2C sąsają [49]. Tuo tarpu servo varikliai yra kontroliuojami PWM (angl. *pulse width modulation*) pagalba. Iš kodo pusės valdymas vyksta *setPWM* komandos pagalba, kurioje nurodomas reikiamo kontakto, prie kurio yra prijungtas servo variklis, numeris bei impulso trukmė. PCA9685 poktštė taip pat gali kontroliuoti kitas plokštes, tokias kaip HG7881C, kurios valdo nuolatinės srovės variklius.

I2C (angl. *Inter-Integrated Circuit*) yra sinchroninis, serijinis komunikacijos protokolas naudojamas nedidelio kiekio, didelių greičių nereikalaujančių duomenų persiuntimo taikymuose. Šio protokolo privalumai yra maži implementacijos kaštai bei paprastumas [50]. Puikiai tinka mažų dydžių LCD ir OLED ekranų valdymui. Komunikacijai naudojami 2 kanalai: SDA (angl. *Serial Data*), linija, naudojama duomenų persiuntimui tarp pagrindinio (angl. *master*) ir pavaldaus (angl. *slave*) įrenginių, bei SCL (angl. *Serial Clock*), linija, kuria sinchronizuojamas taktinis dažnis. Taigi, duomenys yra siunčiami sinchroniniu būdu, t. y. bitai siunčiami vienas po kito taktiniu dažniu. Kiekviena žinutė, arba duomenų paketas, susideda iš duomenų kadrų (angl. *frames of data*), pavaldaus įrenginio adreso bei pagalbinių bitų: pradžios ir pabaigos bitų, skaitymo/rašymo bitų ir ACK/NACK bitų tarp duomenų kadrų.

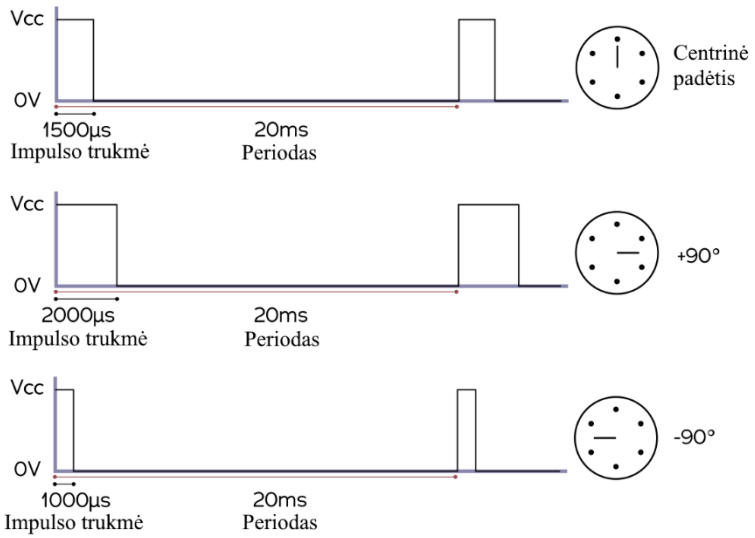


19 pav. I2C žinutės struktūra [51].

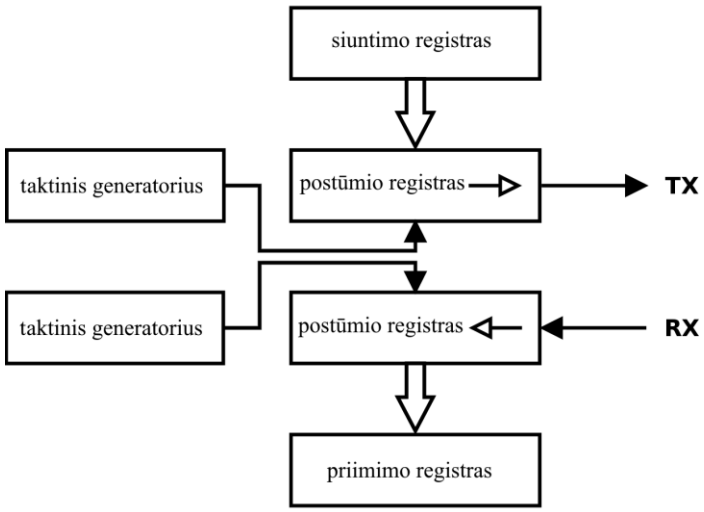
Pradžios ir pabaigos signalai yra žinutės pradžios ir pabaigos žymėjimai, naudojami pagrindiniuose ir pavaldžiuose įrenginiuose, kontroliuoja kada vyksta žinutės skaitymas. Adreso kadras yra sudarytas iš 7 arba 10 bitų, identifikuoja pavaldų įrenginį kai pagrindinis įrenginys nori su juo komunikuoti. Skaitymo/rašymo bitas kontroliuoja, ar pagrindinis įrenginys siunčia duomenis į pavaldų įregninį, ar prašo siųsti duomenis į pagrindinį iš pavaldaus įrenginio. Po kiekvieno duomenų kadro yra įrašomas ACK/NACK bitas, kuris parodo, ar duomenų paketas buvosėkmingai nuskaitytas.

PWM yra toks moduliacijos metodas, kai yra generuojami kintančio pločio impulsai kurie atvaizduoja analoginį signalą [52]. Įprastai būtent PWM yra naudojamas servo variklių kontrolei, nepriklausomai nuo paskirties. Į servo yra paduodamas PWM signalas, pasikartojančių impulsų serija, kurio metu

impulsų plotis keičiasi laike. Priklausomai nuo servo variklio tipo, impulso trukmė gali reikšti judėjimą arba poziciją.



20 pav. Tipinės PWM signalų trukmės servo variklio kontrolei [52].



21 pav. Blokinė UART diagrama.

Komunikacija tarp Python pusprogramio (angl. *script*) ir Arduino mikrovaldiklio vyksta UART (universalus asinchroninis imtuvas-siųstuvas, angl. *universal asynchronous receiver-transmitter*) sąsajos pagalba. UART nėra komunikacijos protokolas, kaip, pavyzdžiui, I2C, o verčiau fizinė grandinė mikrokontroleryje arba atskirame integriniame grandyne. Ši sąsaja leidžia dviems įrenginiams komunikuoti vienas su kitu tiesiogiai – siųstuvas konvertuoja lygiagrečius duomenų bitus į nuoseklius ir siunčia į imtuvą, kitą UART įrenginį, kuris tuo tarpu duomenis gauna ir konvertuoją nuoseklius bitus atgal į lygiagrečius. Komunikacija vyksta asinchroniškai, kas reiškia, kad nėra vieno taktinio generatoriaus, kuri būtų atsakingas už komunikacijos veikimą. Vietoje to, siųstuvas prideda pradžios ir pabaigos bitus į duomenų paketą prieš jį siunčiant. Tie bitai apsako paketo duomenų pradžią ir pabaigą ir tokiu būdu imtuvas žino, kada reikia pradėti ir užbaigti skaityti duomenis. Kai

imtuvas detektuoja pradžios bitą, pradedamas duomenų paketo skaitymas esant tam tikram, iš anksto nustatytam dažniui. Dėl to yra svarbu, kad abu įrenginiai naudotu tą pačią duomenų perdavimo spartą. Arduino programavimo kalboje UART sąsaja yra sukūriama pasitelkus *Serial* komanda.

1.8.5. Unity programinė įranga, ML-Agents įrankių rinkinys

Unity programinė įranga (angl. *Unity software*) yra žaidimų kūrimo variklis, parašytas C++ kalba, sukurtas Unity Technologies. Naudojamas 3D ir 2D projektams ne tik žaidimų kūrime, bet ir kitose srityse, pavyzdžiui: automobilių pramonėje, architektūroje, statybos sferoje, bei net JAV ginkluotuose pajėguose [53]. Šiame variklyje yra siūlomi įvairūs įrankiai, tokie kaip: C# aplikacijų programavimo sąsaja (angl. *API* arba *Application Programming Interface*), kuri skirta pačiai Unity rengyklei (angl. *Unity editor*) bei įskiepiams, taip pat vizualiniai programavimo įrankiai. Šios programinės įrangos 3D dalyje galima rasti fizikos simuliacijoms naudojamus įrankius.

Įprastai darbas prasideda pasirinkus projekto tipą (3D, 2D, itt.). Sukūrus projektą yra sudeliojami žaidimo objektai, 3D modeliai, kameros bei šviesos šaltiniai, pasirenkamos tekstūros. Unity variklis leidžia sudelioti tam tikrą objektų hierarchiją, kuri duoda galimybę tiksliau kontroliuoti žaidimo elgesį. Tuo tarpu pačių žaidimo objektų elgesys yra kontroliuojamas pusprogramiais, rašomais C# kalba. Juose, objektinio programavimo pagalba, galima keisti objektų parametrus, aprašyti judėjimą ir kt. parametrus. Tokių objektų, pusprogramių, tekstūrų, papildynių rinkinys yra vadinamas scena (angl. *scene*).

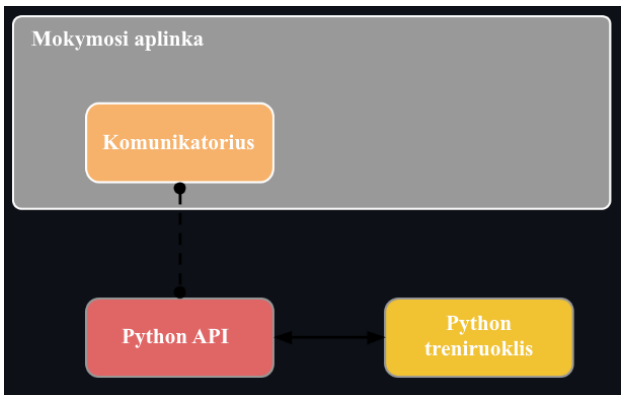
22 pav. Unity programinės įrangos pagrindinis darbo langas [54].

Unity mašinino mokymo agentų programavimo priemonių paketas (angl. *Unity Machine Learning Agents Toolkit* arba *ML-Agents*) yra atviro kodo projektas, leidžiantis žaidimams ir simuliacijoms veikti kaip agentų, arba dirbtinio intelekto modelių, treniravimo aplinka. Jame galima rasti pilną komplektą naujausių ir geriausių algoritmų, skatinamojo mokymo modelių, tarp kurių yra PPO [54, 55]. Šie sudėtingesni įrankiai ir modeliai yra prieinami Python aplikacijų programavimo sąsajos pagalba, kur šie algoritmai yra implementuoti PyTorch įskiepio pagalba. Šio projekto tikslas yra leisti žaidimų kūrėjams bei dirbtinį intelektą tiriantiems mokslininkams sukurti patogią platformą kurioje

galima nesudėtingai kurti ir tirti įvairius algoritmus. Visus įrankius galima naudoti Unity rengyklėje, kuriant įvairius scenarijus, automatiškai keičiant jų parametrus, naudojant skirtingus hiperparametrus.

ML-Agents papildinyje galima išskirti 4 pagrindinius aukšto lygmens komponentus [55, 56]:

- Mokymosi aplinka (angl. *Learning Environment*) – joje yra scena ir visi simuliacijos dalyviai. Aplinka, jos parametrai ir hierarchija priklauso nuo pasirinkto skatinamojo mokymo modelio tikslo.
- Python žemo lygmens aplikacijų programavimo sąsaja (angl. *Python Low-level API*) – sąsaja, leidžianti sąveikauti su ir keisti mokymosį aplinką. Šis komponentas nėra Unity rengyklės dalimi, jis egzistuoja už jos ribų ir bendrauja su Unity per išorinį komunikatorių.
- Išorinis komunikatorius (angl. *External Communicator*) – sujungia mokymosi aplinką su Python žemo lygmens aplikacijų programavimo sąsaja. Egzistuoja mokymosi aplinkoje.
- Python treniruoklis (angl. *Python Trainer*) – šiame komponente yra visi mašininio mokymo algoritmai. Šie algoritmai yra implementuoti atskiroje Python bibliotekoje *mlagents*.

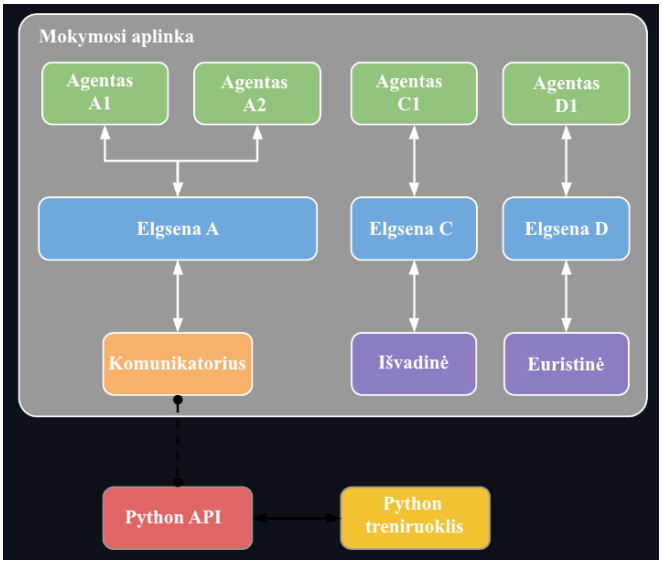


23 pav. Supaprastinta ML-Agents blokinė diagrama [55].

Mokymosi aplinkoje egzistuoja 2 Unity komponentai, kurie padeda organizuoti sceną:

- Agentai – priskiriamas žaidimo objektui ir valdo obzervacijų generavimą, veiksmų atlikimą ir atlygio priskirimą. Kiekvienas agentas yra susietas su elgsena.
- Elgsena (angl. *behavior*) – apibūdina konkrečius agento parametrus ir atributus. Kitaip tariant, tai yra funkcija, gaunanti agento obzervacijas ir atlygius bei grąžinanti veiksmus. Gali būti trijų tipų: mokomoji (angl. *learning*), išvadinė (angl. *inference*) bei euristinė (angl. *heuristic*). Mokomoji elgsena yra ta, kuri šiuo metu yra treniruojama, dar neapibrėžta. Išvadinės elgsenos metu yra naudojamas jau išmokytas dirbtinis intelektas. Euristinė elgsena yra aprašoma specializuotais taisyklių rinkinias (kitaip tariant, rankinis valdymas).

Kiekviena mokymosi aplinka visada turės vieną agentą susietą su vienu žaidimo ar simuliacijos veikėju. Kiekvienas agentas privalo turėti savo elgseną, tačiau elgsena gali būti susieta su daugiau nei vienu veikėju (24 pav.).



24 pav. Pavyzdinio žaidimo blokinė diagrama [55].

Agento logika yra kontroliuojama specialiais metodais, kuriuos galima rasti *agent* klasėje:

- OnEpisodeBegin()
- CollectObservations(VectorSensor sensor)
- OnActionReceived(ActionBuffers actionBuffers)

ML-Agents įrankių rinkinyje treniravimo procesas vyksta epizodiškai, kur kiekvieno epizodo metu agentas bando atlikti jam paskirtą užduotį. OnEpisodeBegin() metodas nustato pradines epizodo sąlygas. Jame gali būti atsitiktinai parenkami įvairūs parametrai, objektų koordinatės. Šis metodas yra kviečiamas kiekvieno epizodo pradžioje. Po to, kai epizodas yra paruoštas, yra kviečiamas CollectObservations(VectorSensor sensor) metodas, kuriame yra nurodoma, kokie dydžiai yra žinomi modeliui. Šie dydžiai yra surenkami į galimybių vektorių (angl. *feature vector*) ir perduodami skatinamojo mokymo modeliui. Algoritmas priima sprendimus remdamasis tik šiais, jam žinomais dydžiais. Paskutinė šio ciklo dalis yra gautų veiksmų interpretavimas bei atlygių dalinimas. Aiškiai nurodoma kokie agento dydžiai (greitis, pozicija, sukimo momentas itt.) gali keistis. Modelis grąžina būtent pasirinktų dydžių pokyčius. Taigi, OnActionReceived(ActionBuffers actionBuffers) metodas gauna veiksmų sąrašą bei paskirsto atlygį. Tai yra labai svarbi dalis, kadangi tai, iš esmės, yra atlygio funkcija – už kokią aplinkos būseną reikia skirti atlygį? Čia galimybės irgi yra plačios, kadangi simuliuojamoje aplinkoje yra žinomi kone visi dydžiai – objektų pozicijos, dydžiai, greičiai, pagreičiai itt.

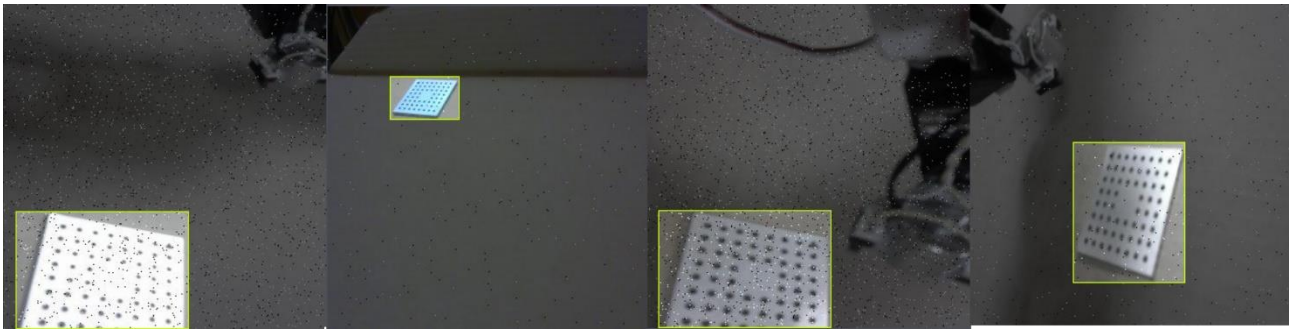
2. Dirbtinių neuro tinklų taikymas

2.1. Treniravimo ir testavimo rinkinių generavimas vaizdo atpažinimui

Treniravimo rinkiniai yra labai svarbūs mašininio mokymo srityje. Daug mokslininkų, privačių įmonių dirba ties efektyvesniu įvairių duomenų rinkimu. Nemažas jų kiekis yra prieinamas internete, nemokamai, pavyzdžiui: MNIST, COCO, įvairių konkrečių objektų (obuolių, vežio auglių itt.). Šiame darbe nuspręsta naudoti išskirtinį, plačiai neprieinamą duomenų rinkinį. Dažnai tai daroma randant daug nuotraukų internete ir su specialios įrangos pagalba žymint tesingus stačiakampius. Nors tai yra viena ilgiausių mašininio mokymo taikymo dalių, ji yra ypatingai svarbi. Šiame darbe, ieškomas objektas yra kvadrato formos metalo gabaliukas su skylutėms. Vaizdo kameros pagalba buvo padaryti 102 atvaizdai. Viename atvaizde maksimaliai gali būti tik 1 ieškomas objektas, t. y. jis gali arba būti atvaizde, arba ne. Toliau, pasinaudojus Roboflow įrankiu, rankiniu būdu buvo pažymėti teisingi stačiakampiai (jeigu objektas atvaizde yra). Prieš pradedant treniravimą, buvo daromi 2 išankstinio apdorojimo veiksmai: iš atvaizdų ištrinami metaduomenys bei atvaizdai buvo sumažinami iki 320×320 pikselių dydžio. Pirmasis veiksmas buvo daromas tam, kad atvaizde nebūtų išsaugota atvaizdo orientacija, kadangi taikymo metu kameros pozicija gali keistis ir atvaizdai gali būti tiek horizontalūs, tiek vertikalūs, o tai gali patrukdyti teisingų stačiakampių koordinatėms. Antras veiksmas yra daromas norint pagreitinti algoritmo veikimą, objekto radimą atvaizde. Toliau buvo daromas treniravimo ir testavimo rinkinių padidinimas (angl. *augmentation*). Tam tikras jau esamų atvaizdų pertvarkymas (ir jų pridėjimas prie jau esančio rinkinio):

- 90° pasukimas, su laikrodžio rodykle ir prieš;
- Soties (angl. *saturation*) keitimas tarp -70% ir + 70%;
- Triukšmo pridėjimas iki nuo 0% iki 5%;
- Apvertimas;

Po visų žingsnių bendras atvaizdų kiekis buvo padidintas iki 244. Treniravimo/validavimo/testavimo rinkiniams buvo skirta, atitinkamai, 87% (213), 8% (19), 5% (12) visų atvaizdų.



25 pav. Treniravimo rinkinio pavyzdiniai atvaizdai su pyžymėtais teisingais stačiakampiais.

2.2. YOLOv5 modelio treniravimas

Kitaip nei YOLOv3, kurio stuburą sudaro DarkNet-53 konvoliucinis tinklas (parašytas C ir CUDA kalbomis), YOLOv5 naudoja CSPDarknet53, kuris yra perrašytas su Python programavimo kalba, pasinaudojant PyTorch biblioteka. Tai palengvina taikyma tuo, kad nėra privaloma naudoti vaizdo plokštės, palaikančios CUDA. Greitis neuro tinklų traniravime yra pageidautinas dalykas, nes skirtumas tarp treniravimo trukmės naudojant vaizdo plokštę ir naudojant centrinį procesorių gali būti keliasdešimt valandų. Tuo tikslu buvo naudojamas Google Research siūlomas įrankis Google Colab. Colab yra yra speciali programavimo aplinka skirta Python kalbai. Šis įrankis veikia ant virtualios mašinos su Linux paremta operacine sistema. Jis yra pritaikytas darbui su mašininio mokymo uždaviniais, o ypač darbui su dirbtiniais neuro tinklais. Įdomu tai, kad kiekvienam vartotojui skiriami resursai bei skirta vaizdo plokštė kinta laikui bėgant. Dažnai skiriamos vaizdo plokštės yra Nvidia K80, T4, P4 ir P100 [57]. Tai yra ypatingai galingos vaizdo plokštės, puikiai tinkančios dirbtinių neuro tinklų treniravimui.

Visa YOLOv5 saugykla yra parsisiunčiama. Tuomet modelis yra konfigūruojamas: nustatoma OpenCV biblitekos naudojama versija, pažymima, kad treniravimas vykst naudojant vaizdo plokštę, nustatomi partijos dydis (angl. *batch size*) į 64 ir subdivizijų skaičius (angl. *subdivisions*) į 16. Tai reiškia, kad vienu metu bus naudojami 4 atvaizdai (64/16). Pagal autorių instrukcijas yra koreguojama pati YOLO architektūra, nustatomi hiperparametrai (nurodomas klasių skaičius bei pakeičiami kai kurių sluoksnių parametrai, susiję su klasių skaičiumi). Taip pat buvo redaguotas pirminis YOLO algoritmo kodas tam, kad svorinių koeficientų vertės būtų saugomos dažniau. Specialiuose failuose nurodomi klasių pavadinimai, treniravimo rinkinio failų keliai. Parsisiunčiamas YOLO svorinių koeficientų failas, kuris buvo iš anksto treniruotas darbo autorių bendram objektų detektavimui. Visas projektas yra suspaudžiamas *zip* formatu ir įkeliamas į Google Drive, kadangi abu šie Google produktai, Colab ir Drive, gali nesudėtingai bendrauti ir dalintis failais tarpusavyje.

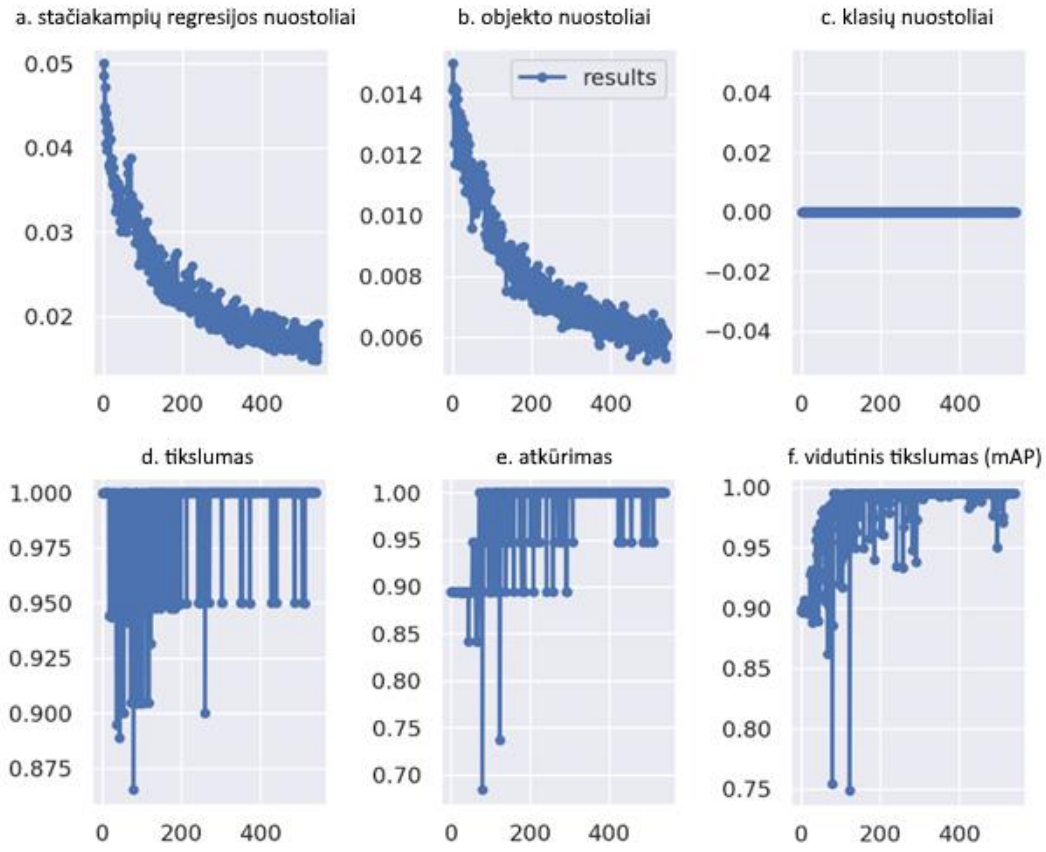
Perkėlus visą YOLOv5 modelį į Colab buvo pradėtas mokymo procesas, naudojant sugeneruotų atvaizdų paketą – treniravimo rinkinį. Modeliui buvo perduodama visa reikalinga informacija – konfigūraciniai failai bei svorinių koeficientų failas. Treniravimas vyko apie 1 val.

2.3. YOLOv5 modelio vertinimas

Modelis buvo treniruojamas iki momento, kai mAP pasiekė 99%, po kurio treniravimas buvo automatiškai stabdomas. Naujos YOLO versijos privalumas yra tas, kad visi grafikai yra automatiškai generuojami treniravimo metu.

Svarbu paminėti, kad tikslumo-atkūrimo kreives bei *mAP* vertinimo kriterijų šiame darbe galima naudoti dėl to, kad tokiame taikyme (objekto atpažinime) nėra labai svarbu atpažinti figūros

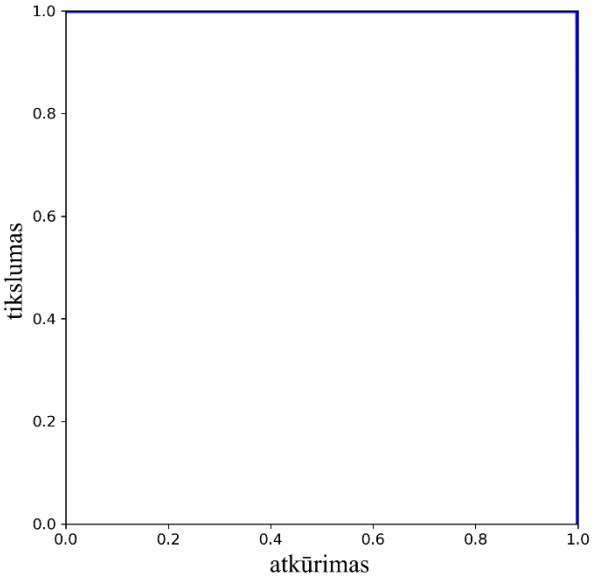
nebuvimą atvaizde. Tikslas yra atpažinti figūrą ir ją lokalizuoti atvaizde. Tai, kad figūra yra ar nėra atvaizde yra tik pagrindinio uždavinio pasekmė. Šis neuro tinklo efektyvumo vertinimo kriterijus nebūtų tinkamas, jei toks elgesys modeliui būtų svarbus, kadangi žiūrint į (7) ir (8) formules galima pastebėti, kad juose nėra atsižvelgiama į tikrai neigiamus rezultatus (*TN*). Tai reiškia, kad paprasčiausiai neatsižvelgiame, ar algoritmas teisingai numatė, kad atvaizde nėra figūros. Praktiškai, tai reiškia, kad gali būti pasirinktas nebūtinai geriausias svorinių koeficientų failas. Tokiu atveju kartais gali atsirasti figūrų detektavimas ten, kur jų nėra. Tai nėra tokia didelė problema video taikymuose, nes toks atvejis gali pasitaikyti viename kadre iš keliasdešimties.



26 pav. Skirtingų dydžių pokyčiai treniravimo metu.

Stačiakampių regresijos nuostoliai (angl. *bounding box regression loss*) parodo kiek tikslūs buvo modelio numatytieji stačiakampiai palyginus su teisingais stačiakampiais. Mažėjant šiai vertei, modelio stačiakampių spėjimų kokybė didėja, t. y. modelis vis teisingiau nustato objekto dydį atvaizde. Objekto nuostoliai (angl. *object detection loss*) yra susijęs su modelio gebėjimu būti įsitikinusiam dėl savo spėjimo. Šiai vertei mažėjant, didėja modelio įsitikinimas, kad spėjimai, kuriuos jis daro, yra teisingi. Klasių nuostoliai (angl. *class loss*) yra dydis, aprašantis modelio įsitikinimą, kad numatytame stačiakampyje esantis objektas priklauso būtent spėtai klasei. Kadangi šiame darbe yra ieškomas tik vieno tipo objektas (yra tik 1 klasė) šis dydis nesikeičia treniravimo metu. Tikslumo ir atkūrimo vertės didėja kartu su didėjančių treniravimo iteracijų skaičiumi ir

pasiekia maksimalią vertę, kas reiškia, kad modelis suranda objektą kiekviename atvaizde (jeigu jis yra) bei surasi objektai iš tiesų yra atvaizde. Didėjant iteracijų skaičiui, vidutinis modelio tikslumas didėjo. Treniravimo pabaigoje vidutinis modelio tikslumas siekė 0.995. Iš tikslumo-atkūrimo bei vidutinio tikslumo kreivių, modelio tikslumą galima vertinti labai teigiamai. Tačiau reikia nepamiršti, kad tokie geri rezultatai gali taip pat reikšti modelio pertreniravimą, kai modelis tobulai veikia su tūrimais treniravimo ir testavimo rinkiniais, tačiau realiai taikant modelį gali daryti neteisingus sprendimus.



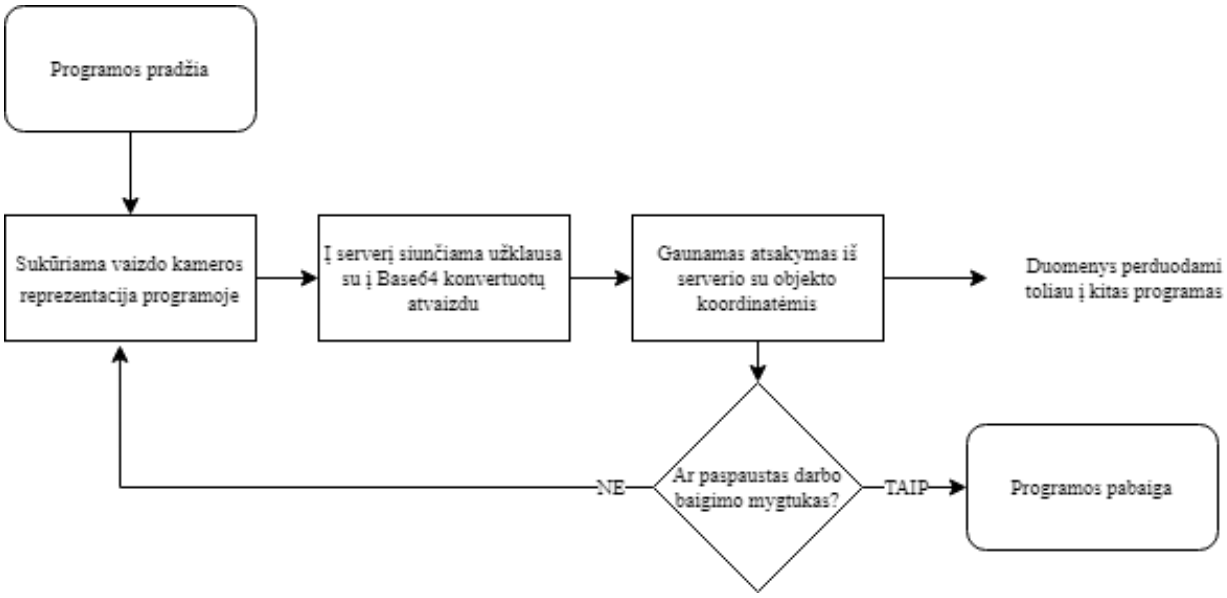
27 pav. Tikslumo-atkūrimo kreivė.

2.4. Praktinis dirbtinio neuro tinklo išbandymas

Norint pritaikyti bet kokį neuro tiklą, tame tarpe YOLOv3 ar YOLOv5, reikia pasirinkti, kurį svorinių koeficientų failą naudoti. Atsižvelgus į rezultatus iš praeito skyriaus, pasirinkta naudoti svorinių koeficientų vertes treniravimo pabaigoje. Toks sprendimas grindžiamas tuo, kad teorinio testavimo metu neuro tinklais su tokiais parametrais parodė geriausius rezultatus – aukštą tikslumą, aukštą atkūrimą – ypač aukštą bendrą vidutinį tikslumą. Papildomai reikalingi modelio architektūros, klasių pavadinimų bei konfigūracinis failai. Jie yra sudedami į vieną aplanką, šalia su pagrindine programa, kurios schema parodyta 28 pav.

Kadangi iš ankstesnių bandymų rezultatų nustatyta, kad YOLOv3 ant esamo kompiuterio veikia ganėtinai lėtai, nuspręsta YOLOv5 laikyti virtualioje aplinkoje ant galingesnio kompiuterio ir modelį pasiektį per internetą. Žinoma, užklausų siuntimas ir atsakymo gavimas gali užtrukti netrumpą laiko tarpą, tačiau manytina, kad didesnis resursų kiekis kompensuos laiko skirtumą. Dar vienas tokio būdo privalumas yra supaprastinta programos implementacija bei veikimas. Patalpinus YOLOv5 modelį aplinkoje, į jį galima kreiptis per programavimo aplikacijos sąsają, siunčiant užklausas su atvaizdo

duomenimis BASE64 formatu į serverį. Priklausomai nuo užklausos tūrinio, galima gauti atvaizdą su teisingais stačiakampiais arba objektų atvaizde koordinates JSON formatu. Kadangi šie duomenys bus vėliau naudojami kai įvadas į kitą algoritmą, nuspręsta duomenis gauti būtent objekto koordinates, o ne visą atvaizdą.



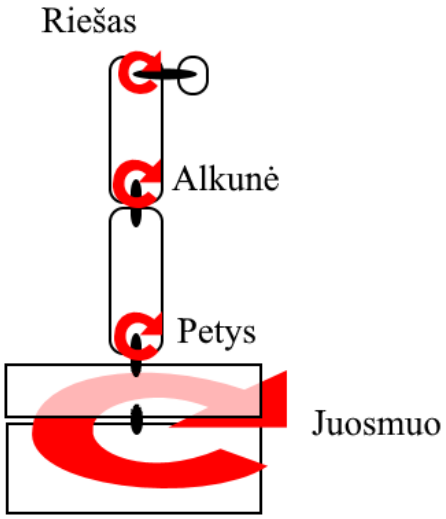
28 pav. YOLOv5 figūrų detektoriaus programos shema.

Kadangi yra naudojamas skyriuje ankščiau aprašytas dirbtinio intelekto modelis, tikslumas gaunamas pakankamai didelis. Praktiškai, modelis gerai reaguoja į objektą gaunamame atvaizde, teisingai numato stačiakampį bei yra įsitikinęs savo spėjimu, tad pertreniravimo problemą galima atmesti. Nors kartais atsakymas iš serverio grįžta tuščias (lyg objekto atvaizde nebūtų) tai netrukdo bendram sistemos veikimui. Tačiau tokio algoritmo veikimas irgi yra pakankamai lėtas, panašiai į YOLOv3, kadrų per sekundę skaičius yra apie 5.

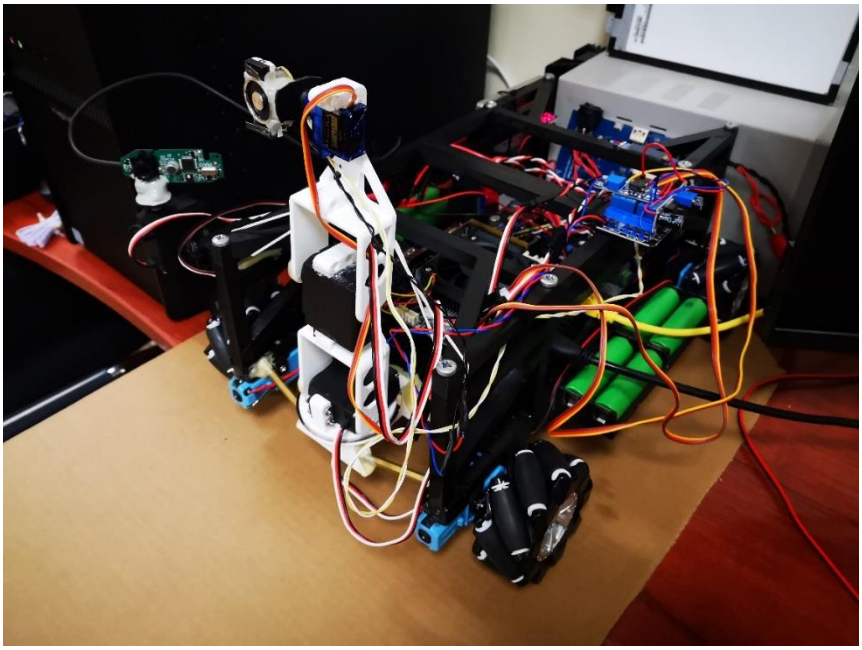
2.5. Roboto specifikacijos, mechanizmo konstrukcija

Roboto pagrindą sudaro specialus, Udoo kompanijos kompiuteris. Buvo naudojamas Udoo BOLT v3 modelis: AMD Ryzen V1202b dviejų branuolių/keturių gijų (angl. *thread*) 2.3 GHz procesorius, integruotas AMD Radeon Vega 3 grafinis procesorius, 2x DDR4 4GB RAM atmintis. Papildomai luste yra integruotas Arduino Leonardo mikrovaldiklis, kuris bus naudojamas mechaniniam rankos ir roboto valdymui, instrukcijų iš dirbtinio intelekto modelio interpretavimui. Thingiverse puslapyje buvo surastas tinkamas roboto rankos modelis. Dėmesys buvo kreipiamas į rankos dydį, jos sudėtingumą, konstrukcijos patvarumą. Suradus tinkamą modelį, jis buvo atspausdintas 3D spausdintuvu ir pritvirtintas prie roboto/kompiuterio karkaso. Žnyplių valdymui naudojamas SG90 servo variklis. Kadangi rankos ilgis yra pakankamai didelis, ant jos galo dedamas svoris privalo būti mažas, tad šis paprastas servo variklis puikiai tinka tokiam taikymui. Kitos roboto dalys yra valdomos

MG996R servo varikliu, kuris, palyginus su SG90, yra patvaresnis bei išgauna didesnį sukimo momentą. Taip pat roboto šone servo variklių pagalba pritvirtinama kamera, kurios gaunamas vaizdas bus perduodamas vaizdo atpažinimo algoritmui. Papildomai ant kameros galo pritvirtinamas ultragarsinis atstumo matavimo modulis HC-SR04P. Jis, kartu su vaizdo atpažinimo algoritmu, padės nustatyti ieškomo objekto poziciją. Viso įrenginio maitinimui yra naudojamos 8 ličio jonų baterijos. Kadangi įvairioms plokštėms ir įrenginiams yra reikalinga skirtinga įtampa bei srovė, yra naudojami antriniai maitinimo šaltiniai, su skirtingomis įtampų ir srovių vertėmis. Ant roboto rankos galiuko yra pritvirtinamas elektromangetas su dviem mygtukais. Elektromagnetas yra valdomas GPIO pagalba su Arduino rėlės pagalba. Mygtukai, taip pat GPIO pagalba, perduoda įvestį į Arduino. LCD ekranas yra naudojamas tam tikrų dydžių išvedimui (ar objektas yra matomas ar ne, jo koordinatės), tuo tarpu OLED ekranas naudojamas kaip grafinė sąsaja tarp vartotojo ir kompiuterio.

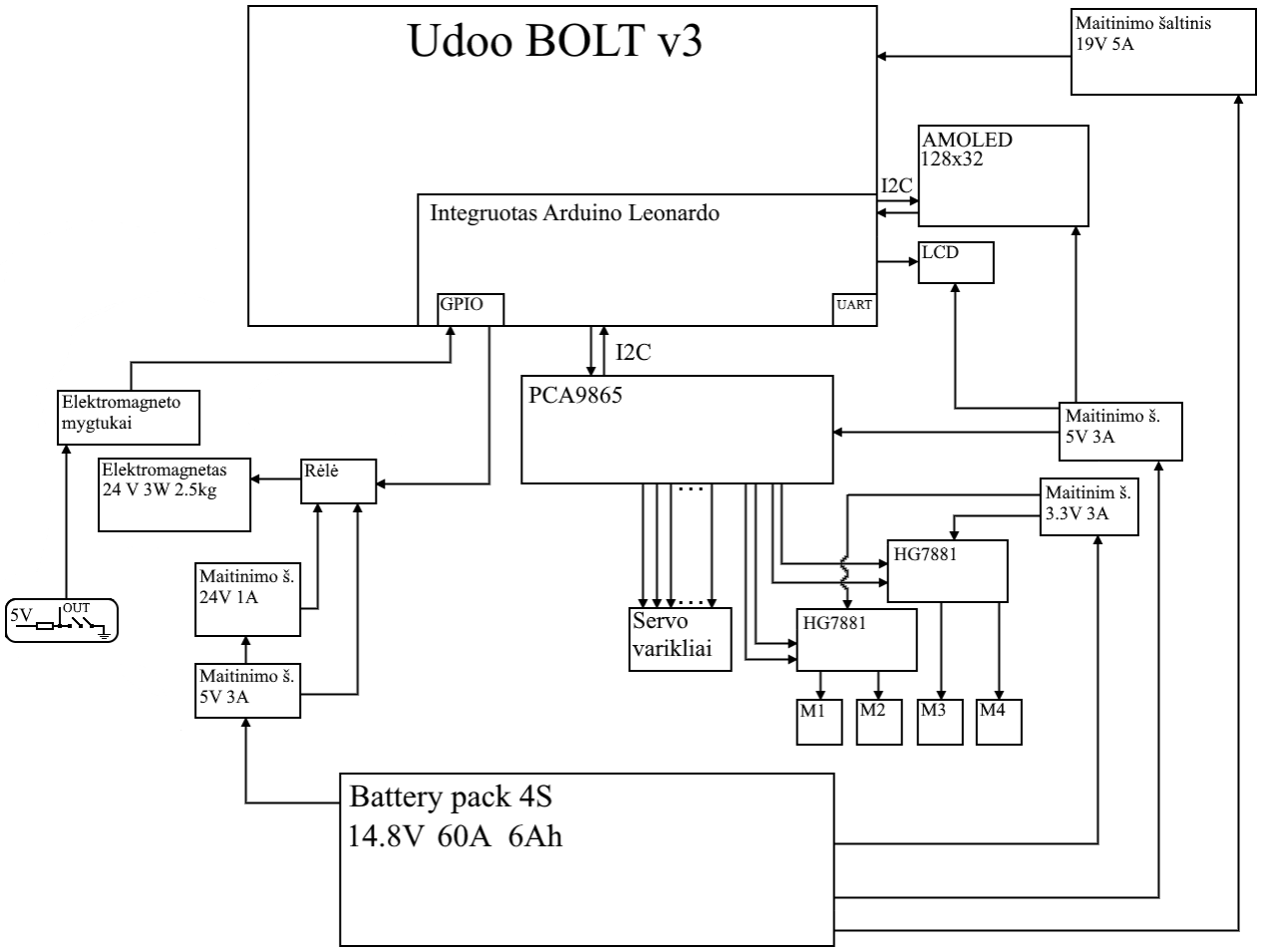


29 pav. Roboto rankos laisvės laipsniai.



30 pav. Robotas – matoma ranka, kamera, kompiuteris.

Viso kompiuterio, Arduino, mechaninių ir valdymo dalių architektūrą galima pamatyti 31 pav.



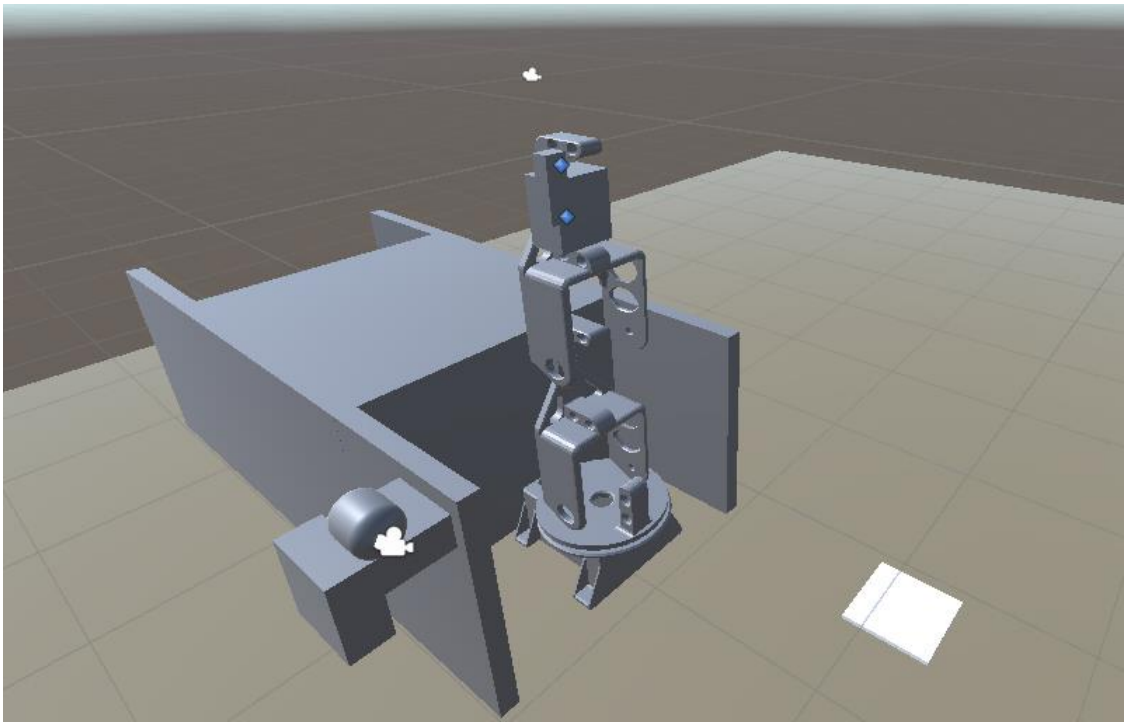
31 pav. Sistemos architektūrinė schema.

2.6. Skatinamojo mokymo modelio treniravimas

Modelio treniravimas atliekamas simuliuojamoje aplinkoje. Tam naudojama Unity programinė įranga, aprašyta 1.8.5 dalyje. Pirmas žingsnis yra tos aplinkos paruošimas. Sukūriamas naujas 3D projektas su standartiniu įrankių ir papildinių paketu. Toliau, norint naudoti naujausią ML-Agents versiją, yra kopijuojamas saugyklos projektas, jis įkeliamas į Unity projektą per papildynių tvarkykle. Lygiagrečiai, iš tos pačios saugyklos yra instaliuojama Python biblioteka *mlagents*. Šiuo atveju nėra naudojama jokia virtuali aplinka, kadangi kompiuterio operacinė sistema yra naujai suinstaliuota ir tuščia, tad konfliktų rizika tarp bibliotekų yra labai maža. Taip pat instaliuojama PyTorch biblioteka. Kadangi ML-Agents yra sąlyginai naujas projektas, daugelyje vietų gali atsitikti versijų neatitikimai su kitais naudojamais įrankiais (PyTorch, Python biblioteka mlagents, Unity programinės įrangos versija), tad reikia įdėmiai sekti jų versijas.

Žinoma, pagrindinė šio projekto dalis yra agentas bei jo aplinka. Išskyrus patį roboto modelį, Unity projekto aplinkoje yra plokštuma, ant kurios treniruojasi agentas, bei kvadratinė detalė, kurią agentas privalo paliesti. Pradžiai buvo pasirinktas kiek lengvesnis tikslas, kurį pasiekus būtų prasminga

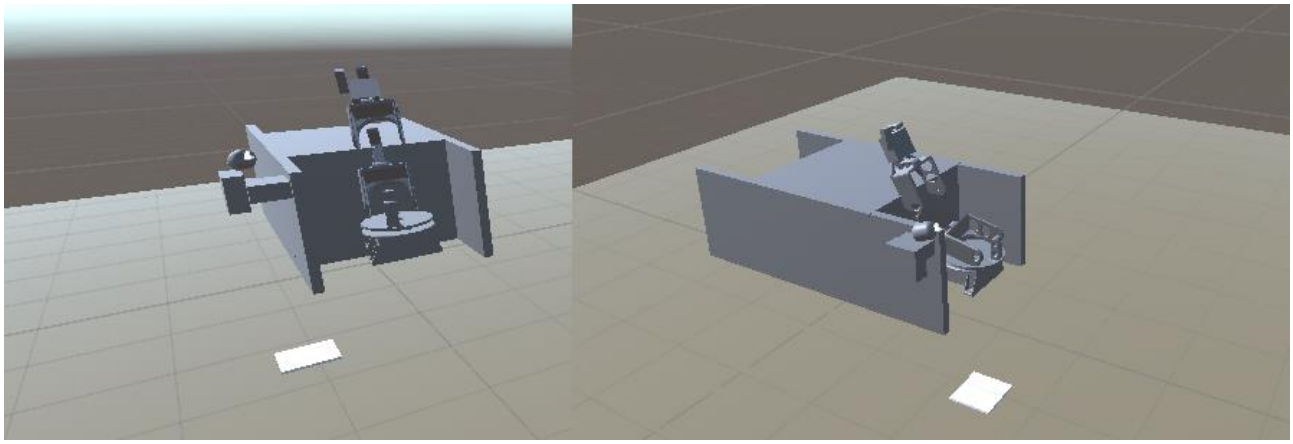
uždavinį apsunkinti. Toks būdas yra tinkamas kai pagrindinė užduotis yra sudėtinga. Agentui įveikiant lengvesnę uždavinio versiją, tolimesnės panašaus tipo užduotys jam palengvėja (su prielaida, kad tęsiamas to pačio modelio treniravimas). Toliau fizinis roboto kūnas buvo pernešamas į virtualią erdvę. Nekreipiant didelio dėmesio į detales, o labiau į fizinius dydžius, mastelio ir laisvės laipsnių skaičiaus išlaikymą, buvo modeliuojamos roboto dalys Unity rengyklės pagalba. Kai visos dalys buvo sumodeliuotos, jos buvo sujungtos specialiomis Unity įrankyje prieinamomis konfigūruojamomis jungtimis tose vietose, kur realybėje yra servo variklis. Jungtims buvo nustatyti judėjimo suvaržymai bei judėjimo plokštumos. Taip pat svarbu paminėti, kad norint, jog Unity fizikos simuliacija bei modelio treniravimas vyktų efektyviai, privaloma taisyklingai nustatyti objektų hierarchiją. Galutinis rezultatas gali būti matomas 32 pav.



32 pav. Sumodeliuoto roboto konstrukcija Unity aplinkoje.

Turint fizinio roboto modelį, sekantis žingsnis yra paruošti aplinką skatinamojo mokymo modelio treniravimui. Epizodo pradžioje yra nustatoma visų roboto dalių pradinė pozicija, rotacija, sustabdomas jų judėjimas, t. y. greičių ir kampinių greičių vertės prilyginamos 0. Kartu yra atsitiktinai nustatoma tikslo, t. y. kvadratinės detalės pozicija. Į agento obzervacijų vektorių įeina 29 pav. raudonai pažymetų judančių dalių rotacijos (kampų dydžiai prijungtos pirminės detalės atžvilgiu). Taip pat yra skaičiuojamas atstumas nuo kameros iki ieškomo objekto, bei viršutinio servo variklio, ant kurio yra kamera, kampas. Iš viso modelis gauna 6 dydžius, kuriuos turi įvertinti. Algoritmo pabaigoje gaunamas veiksmų sąrašas visoms judančioms dalims. Norint supaprastinti uždavinį modeliui, visas robotas negali judėti X, Y, Z kryptimis. Modelio išvestyje gaunamos sekančios, modelio numatytos, judančių dalių kampų vertės, iš viso 4. Iš ankstesnių bandymų buvo pastebėta,

kad algoritmas veikia geriau, kai išvestyje yra gaunamos jėgos, kuriomis reikia paveikti judančias dalis. Tačiau toks veikimas visiškai netinka realiam taikymui, kadangi servo varikliai fiziniame robote kaip įvestį gauna tik jų poziciją. Unity aplinkoje nėra paprasto, iš anksto aprašyto būdo kaip nustatyti sąnario (angl. *joint*) rotaciją, dėl ko implementacija buvo sudėtinga. Taip pat buvo žymiai sumažintas gaunamų obzervacijų skaičius, kadangi, visų pirmą, didelės dalies tokių verčių nebūtų įmanoma gauti fiziškai taikant modelį, o antrą, algoritmui buvo paprasčiausiai per daug dydžių, kuriuos reikėjo įvertinti, dėl ko veikimas nebuvo optimalus.



33 pav. Roboto treniravimas – ieškomas kvadratinis objektas.

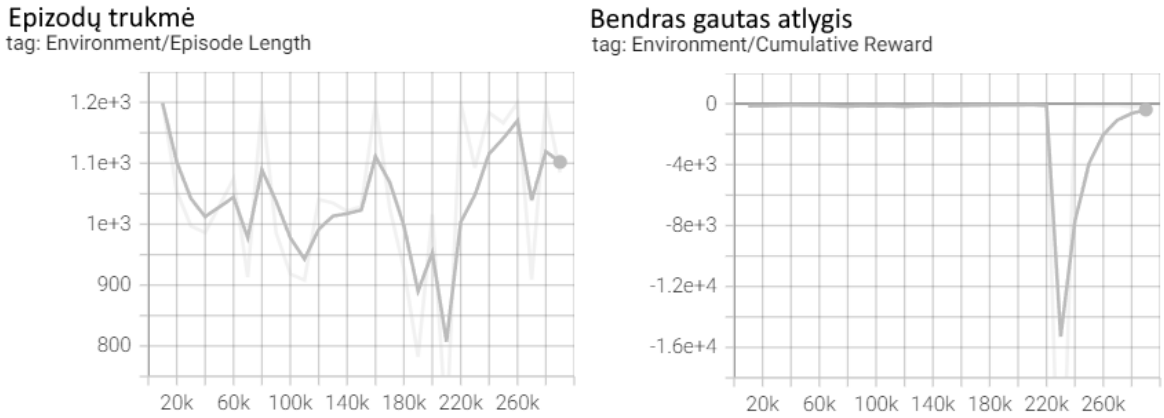
Teigiamas atlygis buvo gaunamas, jeigu roboto rankos galiuko atstumas iki kamuoliuko buvo mažiau nei pakankamai mažas dydis. Pasiektas tikslas reiškė, kad roboto rankos magneto plokštumos 2 jutikliai palietė reikiamą objektą. Epizodo pabaiga taip pat buvo skelbiama, jeigu robotas nukrenta nuo plokštumos ir jeigu buvo padarytas tam tikras žingsnių skaičius nepasiekus tikslo. Pabaigoje buvo konfigūruojamas pats skatinamojo modelio algoritmas, PPO. Pasirinkti hiperparametrai:

```
behaviors:
  RobotReacher:
    hyperparameters:
      batch_size: 512
      beta: 0.001
      buffer_size: 20480
      epsilon: 0.2
      lambd: 0.95
      learning_rate: 0.0003
      learning_rate_schedule: linear
      num_epoch: 3
    keep_checkpoints: 50
```

```
max_steps: 100000
network_settings:
  hidden_units: 128
  normalize: true
  num_layers: 2
  vis_encode_type: simple
reward_signals:
  extrinsic:
    gamma: 0.995
    strength: 1.0
summary_freq: 10000
threaded: true
time_horizon: 1000
trainer_type: ppo
```

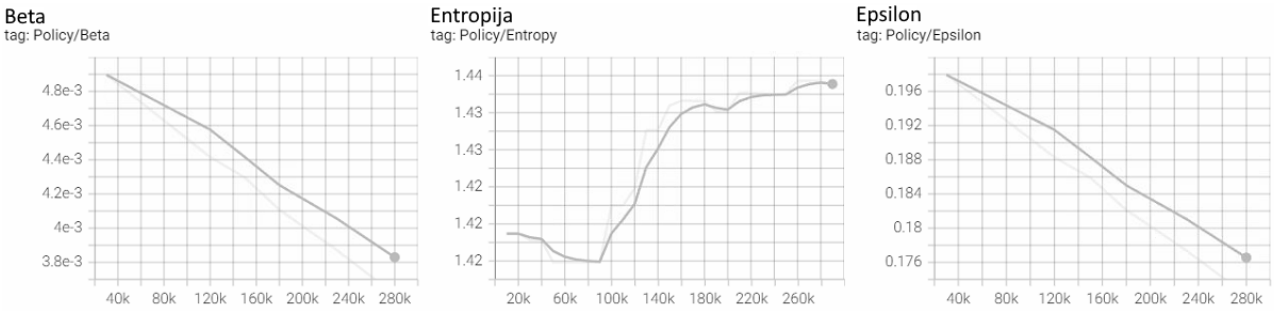
Atlygio funkcija buvo keičiama ne vieną kartą, buvo bandomos įvairios variacijos. Duodant atlygį vien už tai, kad rankos galiukas yra arti kamuoliuko buvo nepakankamai detalus. Kadangi atsitiktinai darant veiksmus pasiekti kamuoliuką yra labai mažą galimybė, jokie veiksmai, nors ir teisingi, nebuvo sustiprinami ir naudojami pakartotinai. Pakeitus atlygio funkciją taip, kad būtų duodamas papildomas mažas atlygis už tai, kad atstumas iki kamuoliuko sumažėjo, rezultatai pagėrėjo, t. y. roboto ranka priartėdavo prie kamuoliuko. Po to, kai buvo pakeisti obzervacijų vektoriai (sumažintas jų skaičius) kartu buvo keičiamos teigiamų ir neigiamų atlygių vertės į labiau proporcingas viena kitai (pavyzdžiui, vietoje -1000 už nukritimą, buvo duodama -1). Po tokių pakeitimų modelis pradėjo 30% visų bandymų pasiekti objektą, tačiau trūko tokio elgėsio, kur modelis pakeistų magneto kampą taip, kad jo jutikliai liestų objektą tinkamu būdu. Norint, kad jutikliai teisingai paliestų objektą, reikia teigiamai atlyginti tokius veiksmus, kurie privestų prie tokio elgėsio. Todėl buvo duodamas nedidelis atlygis tada, kai magneto kampas žemės atžvilgiu buvo tarp 85° ir 95° (pradinė pozicija 0°). Išbandžius tokį modelį didelių pokyčių modelio veikime nebuvo. Toliau buvo keisti hiperparametrai: beta padidintas iki 0.005, *time_horizon* sumažintas iki 500. *time_horizon* vertė nustato vėlesnių žingsnių įtaką bendrai agento strategijai. Sumažinus šią vertę, sprendimai, kurie buvo priimti ankščiau treniravimo metu, turės daugiau įtakos strategijai. beta parametras yra atsakingas už agento priimtų žingsnių atsitiktinumą. Šiai vertei esant per mažai, agentas gali nepakankamai tirti aplinką. Iš entropijos grafiko pastebėta, kad entropijos vertė krenta per greitai, dėl to, pagal autorių rekomendacijas, beta parametras buvo padidintas. Apačioje yra matomi vienos paskutiniųjų treniravimo sesijos rezultatai. Čia modelis yra pritaikytas veikimui realioje aplinkoje, t. y.

obzervuojami ir gaunami tokie dydžiai, kokie gali būti gaunami ir realybėje. Pasiekti gero modelio veikimo nepavyko. Tas matoma iš grafikų ir iš bendro modelio elgėsio treniravimo metu.



34 pav. Epizodų trukmės ir bendro gauto atlygio priklausomybės nuo žingsnių skaičiaus

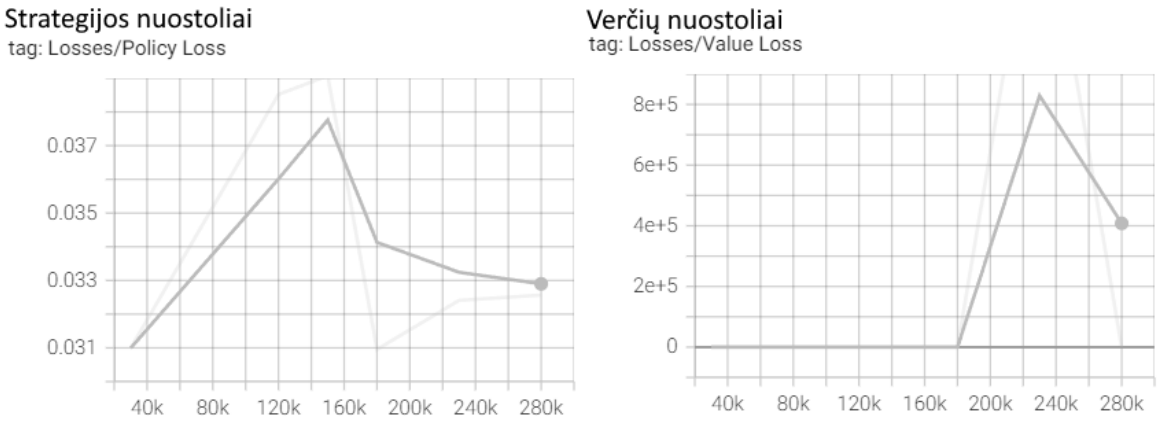
Epizodų trukmė dažnai daug nepasako apie treniravimo kokybę, tačiau įprastai yra geras ženklas, jeigu ši vertė nusistovi. Tai reiškia, kad agentas arba visada pasiekia tikslą darant tuos pačius, veiksmingus žingsnius, arba nepasiekia jo išvis. Šio proceso metu, jokia vertė nenusistovėjo, trukmė tai didėjo, tai mažėjo, kas reiškia, kad optimalaus sprendimo agentas nerado. Bendras gautas atlygis yra bendra vidutinė atlygio vertė tam tikro žingsnio metu. Jeigu agentas sėkmingai mokosi, ši vertė turėtų lėtai didėti proceso metu. Toks rezultatas nebuvo pasiektas, vietoje to, atlygis niekada neišaugo, o tik sumažėjo. Tai gali reikšti neoptimizuotą atlygio funkciją.



35 pav. Beta, epsilon parametrų bei entropijos priklausomybė nuo žingsnių skaičiaus.

Entropija atvaizduoja kaip agento sprendimų atsitiktinumą. Sėkmingo mokymo proceso metu turėtų lėtai mažėti. Iš grafiko 35 pav. galima pamatyti, kad entropija didėjo – agento priimami sprendimai buvo atsitiktiniai. Tai galima paaiškinti tuo, kad agentui nepavyko surasti optimalios strategijos ir norėdamas tai kompensuoti, pradėjo daryti vis daugiau atsitiktinių veiksmų. Tačiau, kadangi kuo ilgiau agentas treniruojasi, tuo veiksmai daro mažiau įtakos strategijai, vidiniai koeficientai nenusitovėjo. Strategijos nuostoliai (angl. *policy loss*) koreliuoja su tuo, kaip stipriai ar greitai kinta agento strategija, t. y. sprendimų priėmimo procesas. Šio dydžio vertė turėtų mažėti, nors smulkus pakilimai ir nusileidimai yra priimtini. Treniravimo proceso metu tokia kreivės eiga nebuvo pasiekta. Tai sutampa su ankstesniais rezultatais. Verčių nuostoliai (angl. *value loss*) atvaizduoja agento

vidutinį verčių funkcijos pokytį. Verčių funkcija yra atsakinga už strategijos keitimą. Kitaip tariant, ši vertė parodo kiek gerai agentas geba numatyti ateities aplinkas ir reakcijas į ją. Sėkmingai treniruojant modelį, turėtų didėti kol agentas mokosi, o kai gaunamas atlygis stabilizuojasi, pradėti mažėti. Sprendžiant iš anksčiau minėtų dydžių, treniravimo laikotarpis buvo per trumpas, kad būtų tiksliai įvertinti verčių nuostoliai.



36 pav. Strategijos bei verčių nuostolių priklausomybės nuo žingsnių skaičiaus.

Bendrai, negalima laikyti šio treniravimo proceso sėkmingu. Taip yra dėl to, kad Unity aplinkoje agento priimti sprendimai yra neteisingai reprezentuojami, būtent dėl šioje sistemoje naudojamo sąnarių judėjimo sistemos. Tačiau esant tam tikroms sąlygoms (Unity aplinkoje valdant roboto rankos galūnes paveikiant jas tam tikra jėga, o ne nustatant galūnių rotaciją) modelis mokosi pakankamai gerai – sėkmingai pasiekiamas tikslas 3 iš 10 bandymų.

2.7. DI modelio pernešimas į realybę (sim2real)

Dirbtinio intelekto modeliai Unity programinėje įrangoje yra išsaugomi ONNX (angl. *Open Neural Network Exchange*) formatu. ONNX yra atviro kodo projektas, prie kurio jau yra prisijungę tokie dirbtinio intelekto taikymo srities gigantai kaip PyTorch, TensorFlow, Nvidia itt. Tai yra vienas pirmųjų bandymų turėti vieną bendrą mašininio mokymo modelio struktūrą. Nors pati Unity Technologies oficialiai nepalaiko jų programinėje įrangoje treniruotų dirbtinio intelekto modelių taikymą už jos ribų, tokį apribojimą būtent ir padeda apeiti ONNX. Dėl implemenavimo paprastumo ir didelio kiekio pagalbinės informacijos buvo pasirinkta naudoti TensorFlow. Python kalba parašytame pusprogramyje yra užkraunamas ONNX modelis. Jam rankiniu būdu paduodant tinkama obzervacijų vektorių, (1, 6) dydžio, kuriame yra tokie dydžiai, būtinai tokia tvarka: atstumas nuo roboto kameros iki ieškomo objekto, juosmens kampas, peties kampas, alkūnės kampas, riešo kampas. Modelio išvestyje gaunamas (1, 4) dydžio vektorius, į kurį įeina naujos vertės juosmens, peties, alkūnės ir riešo kampams.

Tokį modelio pritaikymą išorėje (už Unity programinės įrangos ribų) pavyko pasiekti tik su pirminėmis roboto galūnių valdymo iteracijomis, kurių metu roboto rankos dalių pozicija buvo valdoma pridedant tam tikrą, iš modelio gautą, jėgą. Rankiniu būdu pabandžius į modelio įvestį paduoti tam tikras išgalvotas vertės (obzervacijų vektorių), modelio išvestyje buvo gaunami tam tikri sekantys servo variklių kampai. Tačiau perėjus ant kitokio valdymo principo, t. y. galūnių valdymo tiesiogiai kontroliuojant jų kampus modeliui nepavyko išmokti paliesti objektą, tad jis nebuvo išbandomas praktiškai.

Toliau, pasinaudojus 28 pav. pavaizduota programa, kuri veikė lokaliai kompiuteryje kaip serveris, Arduino pusprogramyje buvo sukurta kameros valdymo logika, kadangi pati kamera veikia atskirai nuo roboto valdymui skirto DI algoritmo. Kadangi skatinamojo mokymo modelis buvo treniruojamas su prielaida, kad kamera visada žiūri į objektą, toks pat veikimas privalo būti ir realybėje. Todėl mintis yra paprasta – ieškomas objektas turi būti kameros matomo atvaizdo centre. Tokiu būdu yra tiksliai gaunamas viršutinio servo variklio, ant kurio yra kamera, kampas, bei teisinga kryptimi nukreiptas atstumo matavimo prietaisas, šiuo atveju ultragarso pagrindu veikiantis sensorius. Objekto centravimo algoritmas yra pakankamai paprastas – jeigu surastas objektas yra už tam tikrų, iš anksto apibrėžtų verčių (nustatytų pagal viso atvaizdo dydį), kamera turi judėti atitinkama kryptimi. Tokiu būdu buvo pasiektas pakankamai geras objekto centravimas atvaizde. Šio algoritmo veikimo spartą riboja objektų detektavimo algoritmas, veikiantis lokaliai kompiuteryje.

IŠVADOS

1. Išmokyto YOLOv5 neuroninio tinklo vidutinis tikslumas (mAP) siekia 99% ir jis puikiai sugeba surasti ieškomą objektą, tačiau nors modelio sparta siekia 4 kadrus per sekundę dėl nepakankamų kompiuterinių resursų kieko ir išsiųstų užklausų atsakymo delsimo, tai yra pakankama sparta šiame taikyme.
2. Priklausomai nuo roboto valdymo būdo virtualioje aplinkoje, modeliui pavyksta surasti ir paliesti ieškomą objektą 3 iš 10 kartų. Vis dėlto, ši vertė nėra pakankama tokiam taikymui ir reikia dar optimizuoti aprašytas atlygių funkcijas.
3. Ištyrus skatinamojo mokymo modelį ir bandant jį perkelti į realybę, nustatyta, kad pasirinktas galūnių valdymo būdas (veikiant jas tam tikra jėga) yra tinkamas taikymui tik virtualioje aplinkoje. Valdant galūnių kampus tiesiogiai modelis nepasiekė pakankamai gero tikslumo ir nebuvo pernešamas į realybę.

LITERATŪRA

1. Khemani, D. (2013). A First Course in Artificial Intelligence. McGraw Hill Education (India).
2. Russell, S. and Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd Edition), Prentice Hall.
3. Murphy, J. (2018). Artificial Intelligence, Rationality, and the World Wide Web. IEEE Intelligent Systems, 33(1), 98–103. doi:10.1109/mis.2018.012001557
4. Kaplan, A., & Haenlein, M. (2018). Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. Business Horizons. doi:10.1016/j.bushor.2018.08.004
5. Poole, David & Mackworth, Alan & Goebel, Randy. (1998). Computational Intelligence: A Logical Approach.
6. Domingos, P. (2015). The master algorithm: How the quest for the ultimate learning machine will remake our world. Basic Books.
7. Hetland, M. (2010). Python Algorithms: Mastering Basic Algorithms in the Python Language. Apress.
8. https://klevas.mif.vu.lt/~linp/page/savokos.html (paskutinį kartą tikrinta: 2022-05-30)
9. Mitchell, T. (1997). Machine Learning. McGraw-Hill, Inc..
10. Shalev-Shwartz, S., Ben-David, S. (2014). Understanding Machine Learning - From Theory to Algorithms. Cambridge University Press.
11. Christopher M. Bishop (2006). Pattern Recognition and Machine Learning. Springer.
12. LeCun, Y. & Cortes, C. (2010). MNIST handwritten digit database.
13. http://neupy.com/2016/11/12/mnist_classification.html (paskutinį kartą tikrinta: 2021-06-01)
14. Buduma, N., & Locascio, N. (2017). Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms. O'Reilly Media, Inc..
15. Aggarwal, C. C. (2018). Neural Networks and Deep Learning. Cham: Springer.
16. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. The MIT Press.
17. Gershenson, Carlos. (2003). Artificial Neural Networks for Beginners.
18. Nielsen, M. A. (2018). Neural Networks and Deep Learning [misc]. Determination Press
19. https://homepages.inf.ed.ac.uk/rbf/HIPR2/convolve.htm (paskutinį kartą tikrinta: 2021-06-13)
20. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (paskutinį kartą tikrinta: 2021-06-14)
21. https://deepomatic.com/en/what-is-image-recognition (paskutinį kartą tikrinta: 2021-06-14)

22. https://machinelearningmastery.com/applications-of-deep-learning-for-computer-vision/ (paskutinį kartą tikrinta: 2021-06-14)
23. https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/ (paskutinį kartą tikrinta: 2021-06-15)
24. https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall (paskutinį kartą tikrinta: 2021-06-16)
25. Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.
26. Morales, M. (2020). Grokking Deep Reinforcement Learning. Manning Publications.
27. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, & Oleg Klimov. (2017). Proximal Policy Optimization Algorithms.
28. Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, & David Meger. (2019). Deep Reinforcement Learning that Matters.
29. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html#what-is-policy-gradient (paskutinį kartą tikrinta: 2022-05-30)
30. https://towardsdatascience.com/policy-gradients-in-a-nutshell-8b72f9743c5d (paskutinį kartą tikrinta: 2022-05-30)
31. https://spinningup.openai.com/en/latest/algorithms/ppo.html (paskutinį kartą tikrinta: 2022-05-30)
32. Yuxi Li. (2019). Reinforcement Learning Applications.
33. Kober, J., Bagnell, J., & Peters, J. (2013). Reinforcement Learning in Robotics: A Survey. The International Journal of Robotics Research, 32, 1238-1274.
34. Wenshuai Zhao and Jorge Peña Queralta and Tomi Westerlund (2020). Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. CoRR, abs/2009.13303.
35. Sandeep Singh Sandha, Luis Garcia, Bharathan Balaji, Fatima M Anwar, & Mani Srivastava (2020). Sim2Real transfer for deep reinforcement learning with stochastic state transition delays. In CoRL 2020.
36. Sebastian Höfer, Kostas Bekris, Ankur Handa, Juan Camilo Gamboa, Florian Golemo, Melissa Mozifian, Chris Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Karen Liu, Jan Peters, Shuran Song, Peter Welinder, & Martha White. (2020). Perspectives on Sim2Real Transfer for Robotics: A Summary of the R:SS 2020 Workshop.
37. Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv e-prints, arXiv:1804.02767.
38. https://medium.com/@anand_sonawane/yolo3-a-huge-improvement-2bc4e6fc44c5 (paskutinį kartą tikrinta: 2022-05-22)

39. Joseph Redmon and Santosh Kumar Divvala and Ross B. Girshick and Ali Farhadi. (2015). You Only Look Once: Unified, Real-Time Object Detection. CoRR, abs/1506.02640.
40. Joseph Redmon. (2013–2016). Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.
41. AlexeyAB. (2016). Yolo v4, v3 and v2 for Windows and Linux [GitHub saugykla]. https://github.com/AlexeyAB/darknet
42. Nepal, U., Eslamiat, H., (2022). Comparing YOLOv3, YOLOv4 and YOLOv5 for Autonomous Landing Spot Detection in Faulty UAVs. Sensors. https://doi.org/10.3390/s22020464
43. Glenn Jocher. (2020). YOLOv5 https://github.com/ultralytics/yolov5 [GitHub saugykla].
44. https://docs.opencv.org/4.5.2/d1/dfb/intro.html (paskutinį kartą tikrinta: 2021-06-18)
45. https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/ (paskutinį kartą tikrinta: 2021-06-18)
46. Adrian Rosebrock. (2015). Imutils [GitHub saugykla]. https://github.com/jrosebr1/imutils
47. https://pillow.readthedocs.io/en/stable/reference/Image.html (paskutinį kartą tikrinta: 2022-05-20)
48. https://www.arduino.cc/en/Guide/Introduction (paskutinį kartą tikrinta: 2022-05-22)
49. Christ, R. D., & Wernli, R. L.. (2014). The ROV Manual (Second Edition). doi:10.1016/B978-0-08-098288-5.00034-8
50. NPX Technologies, I^2C-bus specification and user manual, Rev. 7.0, 2021-ųjų metų rugsėjo 1-oji.
51. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/#:~:text=I2C%20is%20a%20serial%20communication,always%20controlled%20by%20the%20master (paskutinį kartą tikrinta: 2022-05-28)
52. https://learn.adafruit.com/16-channel-pwm-servo-driver?view=all (paskutinį kartą tikrinta: 2022-05-22)
53. https://unity.com/solutions/automotive-transportation-manufacturing (paskutinį kartą tikrinta: 2022-05-20)
54. https://blog.unity.com/technology/evolving-the-unity-editor-ux (paskutinį kartą tikrinta: 2022-05-20)
55. Unity-Technologies. (2017). ml-agents [GitHub saugykla]. https://github.com/Unity-Technologies/ml-agents
56. Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.

57. https://research.google.com/colaboratory/faq.html#gpu-availability (paskutinį kartą tikrinta: 2021-06-18)

Daniel Zakševski

APPLICATION OF ARTIFICIAL NEURAL NETWORKS FOR IMAGE RECOGNITION AND FOR AUTOMATION OF MECHANICAL DEVICES

SUMMARY

In this work, two different artificial neural networks are trained and applied. First, the theory behind artificial intelligence, machine learning and different supervised and reinforcement algorithms is reviewed. Additionally, the tools used in this work are presented.

The first neural network is used for object detection. An algorithm called YOLOv5 (You Only Look Once) is used for this purpose. The searched object is a square piece of metal. Once fully trained, the model reached a mAP (mean average precision) value of 99%, a recall and precision values of 1 are also reached. Practically, the model works very well, correctly identifying and finding the object but only at a 5 frame per second rate using a video feed from a camera due to insufficient computing resources.

The second neural network used in this work is responsible for the control of a mechanical device - a robot arm. The algorithm used for this task is called PPO, or Proximal Policy Optimization. A virtual environment is created in Unity Software, where the training of the model happens using a plugin called ML-Agents. Training is not as successful due to the constraints of object controls presenting in Unity Software. The best result reached is a success rate of 3 out of 10 times of finding the object. While practically possible, transferring an artificial intelligence model from a virtual environment to a real robotic system proves difficult.

The aim of this work is to train and apply an artificial neural network for image recognition and teaching a robot to find and touch an object in a virtual environment.