VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
STUDIJŲ PROGRAMA: INFORMATIKA

Konvoliucinių neuroninių tinklų paieška nuotraukoms klasifikuoti

Convolutional neural network search for image classification

Baigiamasis bakalauro darbas

Atliko: Paulius Karlonas
VU el. p.: paulius.karlonas@mif.stud.vu.lt

Vadovas: Irmantas Radavičius

Vilnius
2022

Turinys

Padėka

Darbo autorius dėkoja Vilniaus universiteto Matematikos ir informatikos fakulteto Informacinių technologijų atviros prieigos centrui už suteiktus HPC išteklius šio darbo skaičiavimams atlikti.

Įvadas

Didelio informacijos kiekio apdorojimas per trumpą laiką gali būti žmogui sunkiai įvykdoma užduotis. Vis dėlto informatikos mokslas šią užduotį galėtų padėti supaprastinti. Šio mokslo šakos viena iš pagrindinių užduočių – ieškoti būdų kaip apdoroti informaciją. Iš tikrųjų minėtieji būdai – tai algoritmai. Algoritmas – tai uždavinio sprendinio veiksmų seka, tačiau tam, kad ji būtų įgyvendinama ir ne per daug kompleksiška, turi patenkinti keletą savybių: diskretumą, programiškumą, determinuotumą, žingsnių elementarumą bei masiškumą [Dič05]. Taigi pasinaudojus algoritmais didelį informacijos kiekį galima supaprastinti taip, kad tai būtų žmogui įveikiama užduotis.

Algoritmai sprendžia nemažai skirtingo tipo uždavinių. Vieni iš jų yra optimizacijos uždaviniai. Optimizacijos uždaviniai išsiskiria tuo, kad juose bandoma surasti geriausią elementą su apibrėžtais apribojimo kriterijais iš visų galimų alternatyvų [SD08]. Optimizacija taikoma ir neuroniniams tinklams. Čia ji pritaikoma ieškant šių tinklų neuroninių jungčių svorių, dauginimo koeficientų, taip „apmokant“ tinklą. Tinklas apmokomas su apmokymo duomenimis, kad šis vėliau grąžintų norimą rezultatą davus jam dar nematytus (validavimo) duomenis. Iš tikrųjų šių tinklų paprasčiausias bei pats pirminis panaudojimas yra kaip klasifikatoriaus. Vis dėlto, dirbtiniai neuroniniai tinklai dažnai nėra tinkamiausias pasirinkimas nuotraukų klasifikavimui, nes dažnai nuotraukų rezoliucija būna nemaža, o tiek neuronų pridėti kainuoja daug resursų, taip pat tai labai greitai priveda prie modelio persimokymo [LVG+20]. Tuomet naudojami konvoliuciniai neuroniniai tinklai. Šis neuroninių tinklų poklasis turi kelis papildomus elementus, kurie sugeba sumažinti ir naudojamų resursų kiekį, ir geriau įsisavinti nuotraukos savybes [AMA17a]. Apmokius konvoliucinius neuroninius tinklus galima gauti klasifikatorių, kuris nuotraukas galėtų supaprastinti, taip palengvindamas užduotį žmogui.

Vis dėlto neuroninių tinklų klasifikavimo rezultatas nemažai priklauso nuo šių jungčių tarpusavio išdėstymo bei jų kiekio – architektūros [SZ15]. Žmogui sudėlioti šią neuroninių tinklų architektūrą užtrunka nemažai laiko ir reikia turėti specifinių žinių šioje sferoje, todėl tai gali atlikti tik nedidelė dalis žmonių [RXC+20] [WZL+19]. Dėl minėtų priežasčių architektūros paieškai atsirado tam tikra algoritmų klasė – NAS, neuroninių architektūrų paieška. NAS klasė aprašo tokius algoritmus, kurie ieško neuroninių tinklų architektūros specifiniams uždaviniams spręsti. Tad pasinaudojus šios klasės algoritmais būtų galima rasti tokią architektūrą, kuri galėtų pakankamai gerai klasifikuoti.

Vis dėlto, dar tik prieš keletą dešimtmečių NAS klasėje daugiausiai buvo kuriami bei naudojami diskretūs algoritmai – šie išbando visas įmanomas architektūras ir renkasi tokią, kuri geriausiai pasirodo atlikdama užduotį [LSY19]. Vis dėlto didžiausia tokių algoritmų problema, kad tokių architektūrų ir hiperparametrų yra labai daug, kas architektūrų paiešką padaro labai ilgą – nuo kelių mėnesių iki keletos metų ar netgi ilgiau [RXC+20] [LSY19]. Kartais ši problema gali būti išsprendžiama algoritmus paralelizuojant, tačiau tai kainuoja nemažai resursų. Todėl šiuo metu NAS algoritmų klasėje atsiranda vis daugiau ir daugiau naujų būdų, kaip galima ieškoti architektūros. Tai ir „svorių pasidalinimo“, ir klasifikavimo teisingumo spėjimo technikos, ir dar kiti

metodai [LSY19]. Taip pat sugalvota architektūrą skaidyti į tam tikras celes – mažesnę architektūrą. Šias celes sujungus gaunama pilnoji (besikartojanti) architektūra. Padalijus architektūrą į celes sumažinamas paieškos laukas, taip pat šią architektūrą apmokius su vienu duomenų rinkiniu, ją lengvai galima panaudoti kitam, dažnu atveju pakeičiant tik šių celių skaičių [EMH19]. Taip pat pradėta orientuotis į algoritmus, kurie sumažina naudojamų resursų kiekį. Vieni iš jų yra paieškos tolydžioje erdvėje algoritmai [LSY19] [LTQ+19]. Šie algoritmai išsiskiria tuo, kad vietoje to, kad išbandytų visas įmanomas architektūras, jie atlieka relaksaciją – šį uždavinį paverčią iš diskrečios į tolydžią erdvę. Vėliau pritaikius tolydaus optimizavimo metodus ieško architektūros tolydžioje erdvėje [LSY19]. Toks sprendimo būdas sumažina naudojamų resursų kiekį, taip pat užtrunkamą laiką [LSY19]. Taigi, pasinaudojus algoritmais, kurie ieško architektūros tolydžioje erdvėje, galima drastiškai sumažinti paieškos laiką, sumažinti naudojamų resursų kiekį bei taip rasti architektūrą, kuri gali klasifikuoti pagal pasirinktus kriterijus.

Tad šio **darbo** tikslas ir yra pasinaudojus CNAS ir NAO algoritmais nustatyti, kuris iš šių algoritmų suranda geresnę architektūrą pagal pasirinktus lyginimo kriterijus bei parametrus.

Šio darbo uždaviniai yra tokie:

- Konvoliucinių neuroninių tinklų klasės analizė
- CNAS ir NAO algoritmų analizė
- Aplinkos paruošimas
- Duomenų parinkimas
- Lyginimo kriterijų nusistatymas
- Tyrinėjamų parametrų pasirinkimas
- Architektūrų paieška su abiem algoritmais pasirinktuose parametruose
- Gautųjų architektūrų palyginimas.

Darbas susideda iš keturių dalių: pirmoje aprašoma konvoliucinių neuroninių tinklų klasė; antroje – CNAS ir NAO architektūros paieškos algoritmai; trečioje aprašomi algoritmų lyginimo pasiruošimas – duomenų parinkimas, lyginimo kriterijų, parametrų nustatymas; paskutiniame, ketvirtame, lyginami gautieji rezultatai.

1. Konvoliuciniai neuroniniai tinklai

Konvoliuciniai neuroniniai tinklai (toliau – KNT) yra mašininio mokymosi, tiksliau, giliojo mokymosi, metodų klasė. Ši klasė išsiskiria tuo, kad jai priklausantys metodai bando minimizuoti (pilnojo sujungimo sluoksnio) parametrų skaičių, taip pat išryškinti svarbiausias nuotraukose esančias savybes prieš duodant klasifikuoti pilnai sujungtam sluoksniui. Pastarasis sluoksnis uždaviniuose, kur duomenų požymių yra pakankamai daug, persimoko [LMK15]. Tai ypač gerai matoma nuotraukose, kur įėjimo požymių skaičius visuomet yra nemažas. Kita problema susijusi su tuo, kad vienus dirbtinius neuroninius tinklus gali būti sunku išmokyti klasifikuoti objektus, nes šie gali būti skirtingose vietose, skirtingų dydžių bei įvairiai orientuoti nuotraukose. Reikėtų milžiniško kiekio duomenų tam, kad būtų apmokytas tik vienas objektas, taip kad visas minėtąsias situacijas būtų galima atpažinti. Galiausiai be persimokymo problemos egzistuoja ir resursų nepakankamumo problema: turint daug parametrų ir norint juos visus greitai optimizuoti (šiuo atveju – ne daugelį metų), reikia galingų skaičiavimo resursų [ON15].

KNT bando spręsti šias problemas įvesdamas kelis papildomus elementus. Per šiuos elementus bandoma surasti svarbiausias nuotraukose esančias savybes. Naudojantis papildomais elementais sumažinamas reikalingas pilnojo sujungimo sluoksnių, taip pat jų parametrų, kuriuos reikia optimizuoti, skaičius. Suradus tokią architektūrą, kuri sugeba surasti nuotraukose esančius objekto požymius bei vėliau juos identifikuoti, gaunamas klasifikatorius.

Baziniai elementai naudojami konstruojant šių tinklų architektūrą, yra tokie:

- Konvoliucinis sluoksnis
- Sujungimo sluoksnis
- Pilnai sujungti sluoksniai

1.1. Konvoliucinis sluoksnis

Šis sluoksnis tinkluose bando išskirti objektų, esančių nuotraukose, išskirtinius požymius, tam, kad kiti sluoksniai galėtų lengviau identifikuoti, dar labiau išryškinti šias savybes. Konvoliucijos operacija – tai matematinė funkcija, kuri kaip parametrus priima dvi funkcijas f ir g, o jos rezultatas parodo šių funkcijų „persidengimą“. Vis dėlto, žiūrint griežtai iš matematinės pusės, šiuose tinkluose ši operacija iš tikrųjų yra kryžminė koreliacija. Nepaisant to, dėl operacijos savybės nuotraukose galima identifikuoti jose esančių objektų išskirtinius požymius, t.y. objektų kraštus, formas, tekstūras ir kitus dalykus. Tai darant prieš pilnai sujungtą sluoksnį išryškinami svarbūs požymiai. Tuomet pilnai sujungtas sluoksnis gali žymiai lengviau identifikuoti išskirtines objekto savybes.

Be duomenų įeities šis sluoksnis turi požymio žemėlapio (angl. feature map), branduolio (filtro) dydžio, judėjimo žingsnio (angl. stride) bei duomenų padidinimo (angl. padding) parametrus.

Dažnu atveju pasirinktas filtro dydis yra santykinai mažas, palyginus su nuotraukos rezoliucija. Mažesnis filtras naudojamas todėl, kad juo galima identifikuoti daugiau objekto požymių. Kita vertus pasirinkus per mažą filtrą savybių/detalių gali būti per daug. Kitaip tariant, nuotrauka mažai

kuo pasikeistų nuo pradinės, kuri turi per daug informacijos.

Priklausomai nuo požymio žemėlapio skaičiaus, sukuriamas atitinkamas branduolio dydžio matricų kiekis. Šių matricų reikšmės dažnu atveju inicializuojamos su atsitiktinėmis reikšmėmis. Kita vertus reikšmes gali parinkti ir tinklo architektas, ypač jeigu žinoma, ko tiksliai ieškoma. Taip darant kiekviena matrica sugeba identifikuoti vis skirtingus objektų požymius. Vis dėlto, vykstant apmokymui, šių filtrų reikšmės yra keičiamos ir taip filtrai keičiasi, o tai keičia ir identifikuojamas objekto savybes.

Šio sluoksnio pagrindiniai veiksmai yra ėjimas per apibrėžto filtro dydžio langelius nuotraukoje ir jų dauginimas iš branduolio matricos. Kiekvienas sekantis ėjimas yra pajudinamas per judėjimo žingsnio kiekį ir patys ėjimai kartojami tol, kol būna „apeita“ visa nuotrauka. Taip pat aprašytieji veiksmai atliekami, kartojami tiek kartų, koks yra požymio žemėlapių skaičius. Šis dydis taip pat nurodo ir išėjimų kiekį. Pasak konvoliucinių tinklų autorių [LBB+98], atliekant šį veiksmą su daugiau negu vienu požymių žemėlapiu tikimasi, kad gautuose požymių žemėlapiuose išryškės skirtingos nuotraukose esančios struktūros ir vėliau tai perduodant į pilnai sujungtą sluoksnį bus lengviau identifikuojamos nuotraukoje esančio objekto savybės.

Šis sluoksnis be papildomų parametrų ir dėl savo specifikos gali prarasti duomenis – sumažinti rezoliuciją [AMA17b]. Nuotraukos rezoliucija gali sumažėti esant didesniam filtro dydžiui negu 1×1. Vis dėlto, tai gali būti išsprendžiama įvedant dirbtinį duomenų padidinimą. Šio proceso metu sukuriama tiek nulių aplink nuotrauką, kad branduoliui judant per nuotrauką nebūtų mažiau duomenų negu jo apibrėžtas dydis.

Iškart po šio sluoksnio, norint normalizuoti duomenis, papildomai pritaikomas ir netiesiškumas. Šiame sluoksnyje dažniausiai naudojama *ReLU* funkciją. Ji apibrėžiama kaip: $\mathrm{ReLU}(x) = max(0, x)$. Ši funkcija bendruomenėje pasirinkta dėl kelių priežasčių [AMA17b]. Viena iš jų tokia, kad šią funkciją galima greitai suskaičiuoti. Antroji priežastis susijusi su „prarasto gradiento“ problema. Ši problema dažnai sutinkama naudojant sigmoidine arba hiperbolinio tangento funkcijas dėl jų mažo atstumo nuo 0. Vis dėlto, ši problema neiškyla naudojant ReLU funkciją. Galiausiai ši funkcija sukuria ir įvairesnę išėjimų reprezentaciją negu jau minėtos funkcijos.

1.2. Sujungimo sluoksnis

Šis sluoksnis mažina išeinamų duomenų, nuotraukos atveju – rezoliucijos – kiekį, kuris proporcingas turimam kaip parametrui lango dydžiui.

Visų pirma nuotrauka yra suskaidoma į duoto lango dydžio ilgio bei pločio langelius, atsižvelgiant į judėjimo žingsnio bei duomenų papildymo parametrus. Kaip ir konvoliuciniame sluoksnyje, šiame sluoksnyje gali atsirasti problema su langelio dydžiais, jeigu filtro dydis yra didesnis negu 1×1. Tai yra, tam tikri langeliai gali būti mažesni negu apibrėžtas filtro dydis. Tuomet gali būti naudojamas dirbtinis duomenų papildymas nuliais aplink nuotrauką.

Vėliau keliaujama per kiekvieną langelį ir atitinkamai nuo šio sluoksnio tipo atliekama operacija bei grąžinama viena reikšmė. Taip darant juose paryškinami svarbūs požymiai, o nereikšmingos

detalės atmetamos. Šio sluoksnių tipų yra du: vidutinis bei maksimalus.

Vidutinis skaičiuoja einamojo langelio vidurkį bei išvestyje grąžina vienintelę reikšmę – to langelio vidurkį.

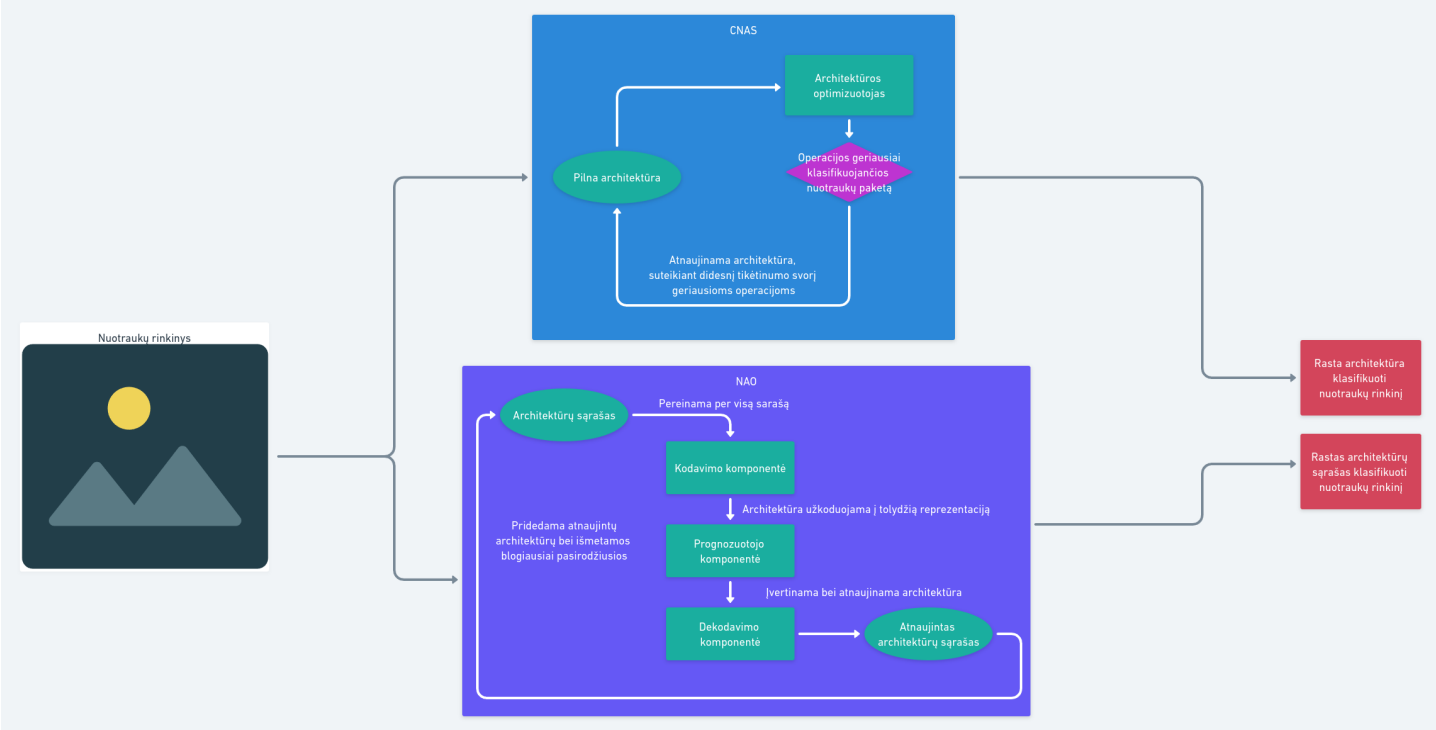
Maksimalusis ieško langelyje didžiausios galimos reikšmės ir grąžina ją vietoje viso langelio. Be to, šis metodas yra greitesnis negu vidutinis, todėl, norint tinklus apmokyti greičiau, renkamasis šis tipas.

1.3. Pilnai sujungti sluoksniai

Šiame sluoksnyje kiekviena gaunama įeitis pasiunčiama į kiekvieną neuroną bei padauginama iš to neurono atitinkamos įeities svorio. Visi šie rezultatai kiekviename neurone yra susumuojami bei pridedamas to neurono poslinkis (angl. bias). Galutinai šis rezultatas gali būti paduodamas į aktyvacijos funkciją. Dažniausiu atveju, jeigu sluoksnis yra paskutinis, naudojama funkcija, kuri grąžina kiekvienos išeities tikimybę. Viena iš tokių funkcijų, kuri naudojama dažniausiai, yra *Softmax*. Ši funkcija dažniausiai renkamasi dėl jos išeičių tikimybių susisumavimo į 1. Kitos, kaip sigmoidinė arba hiperbolinis tangentas, papildomai reikalauja visų reikšmių normalizacijos. Kitu atveju, jeigu šis sluoksnis nėra paskutinis, dažniausiai renkamasi naudoti *ReLU* funkciją.

2. Architektūros paieškos algoritmai

Šiame skyriuje aprašomi du tiriami algoritmai ieškantys architektūros: jų konceptai, pasitelktos idėjos bei kaip vykdoma pati architektūrų paieška.



1 pav. NAO ir CNAS algoritmų schema

2.1. CNAS

CNAS – tai automatinis konvoliucinių neuroninių tinklų architektūros paieškos algoritmas, kuris paverčia diskrečią architektūros paiešką į tolydžią, suranda operacijas, kurios geriausiai klasifikuoja, bei tuomet išveda architektūrą iš šių operacijų. Šis algoritmas [WZL+19] naudoja **DARTS** [LSY19] algoritmą kaip bazę. **DARTS** algoritmo išskirtinumas yra architektūros, kuri yra diskreti, paieškos transformavimas į tolydžiąją paieška. Tiksliau, diskretus optimizavimo (architektūros paieškos) uždavinys paverčiamas į tolydųjį uždavinį. Atlikus tokią transformaciją, šiam uždaviniui galima panaudoti metodus, kurie yra skirti tolydiems uždaviniams optimizuoti. Konkrečiau tariant, paieškai yra pritaikoma gradientinio nusileidimo metodas, taip darant yra pagreitinama pati paieška, kadangi architektūros paieška diskrečioje erdvėje (išbandymas visų įmanomų architektūrų) yra labai ilgas bei daug resursų pareikalaujantis procesas [LSY19]. **CNAS** algoritmo specifiškumas nuo DARTS yra susikoncentravimas į kitokią primityvių operacijų aibę [WZL+19].

2.1.1. Sudedamosios dalys

Algoritmas susidaro iš celės bei optimizatoriaus komponentų. Šios celės yra jungiamos su kitomis taip sudarant neuroninio tinklo sluoksnius – pirminę architektūrą. Vėliau šią architektūrą norima optimizuoti – gauti operacijų tikėtinumo svorius, tad tam jau yra naudojamas optimizatorius. Galiausiai, turint tikėtinumo svorius, belieka sudaryti galutinę architektūrą – išrinkti tokias operacijas, kurios labiausiai prisideda prie klasifikavimo rezultato.

2.1.1.1. Celė bei jos reprezentacija

Šiame algoritme celė – tai sluoksnis, savyje laikanti informaciją apie primityvias operacijas (žr. 2.1.3 skirsnį), jų svorius bei jų tikėtinumą. Ji reprezentuojama kaip multigrafas, neturintis ciklų. Šis grafas sudarytas iš N viršūnių, o šias viršūnes jungia briaunos, kur iš kiekvienos viršūnės išeina po K briaunų, kur K – tai visos primityviosios operacijos. Taip pat šis grafas visuomet turi bent dvi įėjimo ir vieną išėjimo viršūnę. Šį grafą galima įsivaizduoti kaip einantį iš viršaus į apačią, kuris kiekvienoje eilėje iš praėjusios pasipildo viena viršūne. Kitaip tariant, pradedant nuo viršaus, kur yra 2 viršūnės, ir paėjus viena eile į apačią, joje jau egzistuotų trys viršūnės. Taip pat visos viršūnės jungiasi su visomis buvusiomis (aukščiau esančiomis) viršūnėmis.

Į pirmąsias dvi viršūnes ateina iš praėjusių dviejų celių gauti rezultatai, o apatinėje viršūnėje pritaikoma konkatenacijos operacija – visi į ją įeinančių viršūnių rezultatai sudedami kartu.

Celės viršūnės savyje laiko informaciją apie atitinkamus svorius operacijoms, o iš šios viršūnės išeinančios briaunos yra konvoliucinių tinklų elementai – primityvios operacijos.

2.1.1.2. Optimizatorius

Optimizatorius – tai komponentė, kuri bando optimizuoti operacijų tikėtinumo bei pačių operacijų svorius celėje. Tiksliau tariant, operacijos tikėtinumo svoriai nurodo, kurios operacijos daugiausiai prisideda (turi didžiausią svorį) prie galutinio klasifikavimo rezultato, o pačių operacijų svoriai – tai kiekvienos operacijos (konvoliucijos, sujungimo ir kt.) filtro svoriai. Pats optimizacijos uždavinys formuluojamas taip: turint du (praradimo) įverčius $\mathcal{L}_{train}$ bei $\mathcal{L}_{val}$, reikia rasti tokį α(architektūrą), kad būtų minimizuota $\mathcal{L}_{val}$ reikšmė, kai turima tokius tinklo svorius w, kurie yra gauti minimizuojant $\mathcal{L}_{train}$ (žr. (1)-ąją lygtį). Kitaip tariant, turint omeny, kad operacijų svoriai yra optimalūs, pagal validavimo duomenis bandoma ieškoti tokia architektūra, kurios validavimo duomenų praradimo reiškmė būtų mažiausia. Čia $\mathcal{L}_{train}$ bei $\mathcal{L}_{val}$ yra praradimo funkcijos reiškmės.

$$\begin{aligned} &\min_{\alpha} \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ \textbf{jei, } w^*(\alpha) = &\operatorname*{argmin}_{w} \mathcal{L}_{train}(w,\alpha) \end{aligned} \tag{1}$$

Tad lygtis, kuri yra sprendžiama:

$$\nabla_{\alpha}\mathcal{L}_{\text{val}}\ (w^*(\alpha), \alpha) \approx \nabla_{\alpha}\mathcal{L}_{\text{val}}\ (w - \xi\nabla_{w}\mathcal{L}_{\text{train}}\ (w, \alpha), \alpha) \tag{2}$$

Pastebėtina, kad w^* čia bandoma aproksimuoti darant tik vieną mokymo žingsnį, o ne bandant dar papildomai optimizuoti 1 lygties vidinę dalį [LSY19].

Tad gavus dvi praradimo reikšmes bandoma optimizuoti operacijų tikėtinumo svorius bei bandoma pagerinti tų operacijų svorius.

2.1.2. Algoritmo eiga

Pati architektūros paieška yra operacijos tikėtinumo bei pačių operacijų svorių paieška celėje. Taip pat, kadangi ieškant teisingiausio tinklo reikia jį (pastoviai) įvertinti, kad (validavimo duomenų praradimo) reikšmę būtų galima perduoti optimizatoriui, bandomi optimizuoti ir pačių operacijų svoriai. Todėl galutiniame rezultate be architektūros gaunami ir šios architektūros operacijų svoriai.

Diskretus šių operacijų pasirinkimas paverčiamas į tolydų, pasinaudojus *softmax* funkcija kiekvienai operacijai bei padalinant ją iš visų operacijų sumos pritaikius *softmax* funkciją visoms toms kitoms operacijoms (žr. (3)-ąją lygtį). Čia tariame, kad $\mathcal{O}$ – tai aibė visų galimų primityvių operacijų, o $\alpha_o^{(i,j)}$ – tai celės viršūnėje (i, j) operacijos(o) tikėtinumo svoris. Kitaip tariant, kalbant minėtojo grafo apibrėžimais, tai yra briaunų, kurios jungia dvi viršūnes, prisidėjimas prie tinklo klasifikavimo rezultato.

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x) \tag{3}$$

Gavus šių operacijų prisidėjimo koeficientus bei įvertinus šių tinklų teisingumą, šie perduodami į optimizatorių, kuris vertina, kurios operacijos labiausiai prisideda prie galutinio rezultato. Šitaip prisidėjusioms operacijoms yra suteikiama dar daugiau svorio.

Galiausiai, po tam tikro epochų skaičiaus ir turint kiekvienos operacijos bei jų tikėtinumo svorius kiekvienai celės viršūnei, išrenkama tokia operacija, kuri labiausiai prisideda prie galutinio klasifikavimo rezultato, o tai nusprendžiama pagal didžiausią operacijos tikėtinumo svorį.

2.1.3. Primityvios operacijos

Primityvios operacijas – tai iš anksto apibrėžti KNT elementai su atitinkamais parametrais. Vienas iš šio algoritmo išskirtinumų nuo **DARTS** – tai skirtingai pasirinktos primityviosios operacijos, kurios yra naudojamos ieškant architektūros. **DARTS** eksperimente naudotų operacijų tipų yra mažiau – tai atskirtos konvoliucijos, išplėstosios atskiros konvoliucijos, tapatybės (angl. identity), nulinė bei vidutinės ir maksimalaus sujungimo operacijos. Taip pat konvoliucijos bei sujungimo sluoksnio operacijoms, pridėtos analogiškos, tik kito dydžio, operacijos – tai 3x3, 5x5 bei 7x7 branduolio dydžių operacijos.

Vis dėlto, [WZL+19] straipsnio autoriai išryškino savybes, kurias turi patenkinti primityviosios operacijos. Dėl šios priežasties **CNAS** eksperimentuose naudotos operacijos šiek tiek skyrėsi nuo minėtojo algoritmo. Pasak autorių, primityvi operacija turėtų patenkinti kelias sąlygas: unikalumo, turėti kiek galima mažiau parametrų bei būti greitai suskaičiuojama. Unikalumas – tai,

kai operacija turi tokias savybes, kurių negali replikuoti jokia kita operacija. Turėti kiek galima mažiau parametrų – turimas omenyje reliatyvus skaičius nuo kitų operacijų, kurias reikėtų optimizuoti. Galiausiai būti greitai suskaičiuojama – operacija turi būti kuo greitesnė (vėlgi lyginant su kitomis operacijomis). Tad turint visas šias savybes omenyje buvo pašalintos skirtingo branduolio dydžio operacijos, kadangi šias galima išgauti turint minimalią (3x3) operaciją bei dedant jas iš eilės. Papildomai prie esamų operacijų buvo pridėtos dvi prieš tai nebuvusios operacijos – suspaudimo bei sužadinimo ir kanalų sumaišymo konvoliucijos operacijos. Iš tikrųjų abi operacijos yra tam tikras KNT elementų rinkinys. Štai suspaudimo bei sužadinimo operacija, kuri pirma buvo aprašyta straipsnyje [HSA+19], ja naudojantis bandoma išgauti globalią informaciją per visus kanalus (savybių žemėlapius), taip bandant sumažinti tinklo koncentraciją į mažas, bet nereikšmingas detales. Kitoje operacijoje, kanalų sumaišymo, bandoma sukeisti kiekvieno kanalo poziciją su kita, o tai bedarant dar panaudoti ir konvoliuciją. Tai atlikus tikimasi, kad tinklas taip greitai nepersimokys ir galės labiau įsisavinti kiekvienos nuotraukos savybes [ZZL+17].

2.2. NAO

NAO – tai neuroninių tinklų optimizacija, kuri bando ieškoti (pagerintos) konvoliucinių tinklų architektūros pasinaudojus kitais neuroniniais tinklais [LTQ+19]. Šio algoritmo idėja panaši kaip **CNAS** (žr. 2.1 sekciją) – diskrečią architektūrą perkelti į tolydžią erdvę, ją optimizuoti, o tada grąžinti į diskrečiąją erdvę. Vis dėlto, šio algoritmo išskirtinės savybės lyginant su **CNAS** yra ir skirtingas perkėlimo bei grąžinimo iš diskrečiosios į tolydžiąją erdvę būdas, ir pats optimalios architektūros ieškojimo būdas.

2.2.1. Sudedamosios dalys

Šiame algoritme yra naudojamos kodavimo, prognozuotojo bei dekodavimo komponentės, kurios bando pagerinti naudojamų operacijų celėje architektūrą, tam, kad jos klasifikavimas taptų teisingesniu. Visų pirma kodavimo mechanizmas paverčia simbolių eilutę, sudarytą iš skaičių, tolydžiu vektoriumi. Tai padarius šiam vektoriui, o tiksliau – juo užkoduotai architektūrai, galima panaudoti prognozuotoją, kuris bando įvertinti šio tinklo klasifikavimo tikslumą. Įvertinus tinklą, galima panaudoti gradiento nusileidimą, kad būtų galima optimizuoti tinklą, jo reprezentaciją, ir gauti naują, tikėtinai geresnį tinklą jau dekoduojant. Atlikus šiuos veiksmus, turima nauja architektūra, kuri, tikėtina, bus geresnė negu senoji. Šiuos veiksmus atlikus daug kartų bus gauta optimali architektūrą, kurios ir yra ieškoma.

2.2.1.1. Celė ir jos reprezentacija

Celė – tai sluoksnis savyje laikantis tam tikras primityvias operacijas bei jų svorius. Atitinkamai, jeigu yra naudojamas svorių pasidalijimas, joje yra laikomos visos primityvios operacijos, kitu atveju – tik architektūros nurodytos operacijos. Pati celė reprezentuojama kaip orientuotas, neturintis ciklų multigrafas, kurį sudaro nustatytas skaičius viršūnių, taškų. Šiuose taškuose saugomi

primityvių operacijų svoriai, o iš šių viršūnių išeinančios briaunos yra tam tikros primityvios operacijos. Tai panašus celės konceptas kaip ir **CNAS** algoritme (žr. 2.1.1.1 straipsnį). Vis dėlto, šis konceptas išsiskiria tuo, kad kiekvienas taškas turi tik po dvi operacijas, išskyrus paskutinį tašką.

Atlikus abi operacijas taške, jų rezultatai yra sujungiami kartu bei iš celės taško yra išvedama šių rezultatų sąjunga. Taip pat, visos šios operacijos gali naudoti rezultatus iš jau buvusių taškų. Kitaip tariant, kiekviena operacija taške gali panaudoti jau buvusio taško, dviejų operacijų sujungta, rezultatą.

Verta paminėti, kad kiekviena celė architektūroje turi du įėjimus ir vieną išėjimą, tad pirmasis taškas visada turės bent dvi įeitis. Galiausiai šia architektūros idėja ir buvo pasinaudota todėl, kad yra naudojamas svorių pasidalinimas.

2.2.1.2. Svorių pasidalijimo komponentė

Svorių pasidalijimo idėja yra tokia, kad visos pirminės operacijos yra sudedamos į celę, o tada, pasirinkus specifinius elementus (sudarius architektūrą), iš celės yra naudojamos tik pasirinktos operacijos, taip šią bendrąją architektūrą, o tiksliau atskirai jos svorius, apmokant (su vis kitokia) pasirinkta architektūra. Šios technikos idėja pasitelkta iš [PGZ+18] straipsnio. Ji panaši į **DARTS** [LSY19] naudojamą idėją, kad visas operacijas į tinklą reikia sudėti kartu. Vis dėlto, šios idėjos išskirtinumas yra tai, kad ji aktyvuoja, naudoja tik nurodytas operacijas ir optimizuojant tinklą atnaujinami tik naudotų operacijų svoriai, neturint pačių operacijos tikėtinumo svorių. Tai yra operacijų svoriai naudojami bendrai visoms (pasirinktoms pagal elementus) architektūroms. Pasak pačių straipsnio autorių [PGZ+18], šitoks sprendimas ne tik nesumažino klasifikavimo teisingumo, bet ir jį šiek tiek padidino bei tūkstančiais kartų pagreitino visą paieškos procesą. Tad šiame algoritme pasinaudojus bendru tinklu, kuris yra nuolat apmokomas, galima gauti neblogą aproksimaciją nurodytai architektūrai, o taip pat pagreitinamas šių architektūrų radimas.

2.2.1.3. Paieškos komponentai

Šiame skirsnyje yra aprašomos trys komponentės, kurios ir yra naudojamos šiame algoritme pagerinti ieškomą architektūrą.

Kodavimo komponentė priima vieną argumentą – architektūrą, kuri reprezentuojama diskrečiai naudojant simbolių eilutę, kurią sudaro tik skaičiai. Pirmame veiksme gautajam argumentui pritaikomas „įterpimo sluoksnis“ (angl. *embedding* layer), kuris transformuoja šią simbolių eilutę į skaitinį vektorių ir vėliau šis naudojamas paverčiant architektūrą į tolydžią jos reprezentaciją. Atlikus šį žingsnį pritaikomas ilgosios-trumposios atminties neuroninis tinklas (toliau LTSM), kur jo rezultatas vektorius – tolydi architektūros reprezentacija. Toliau gautasis rezultatas perduodamas prognozuotojui.

Prognozuotojas yra vienas iš kodavimo komponentės sudedamųjų dalių. Šis yra panaudojamas iškart po architektūros užkodavimo. Ši dalis susideda iš kelių veiksmų. Iš pradžių gautajam vektoriui pritaikoma vidurkio sujungimo operacija – kiekvienas vektoriaus elementas yra padalintas iš

šio vektoriaus ilgio. Atlikus tai pritaikomas dirbtinis neuroninis tinklas, kuris bando prognozuoti šio tinklo įvertį, klasifikavimo neteisingumas. Gavus įvertį šis vėliau pritaikomas arba apmokant prognozuotoją norint sumažinti jo paklaidą, arba, kai jau ieškoma tinklo, bandant optimizuoti tinklo operacijas, pasinaudojus šiuo įverčiu kaip praradimo funkcijos reikšme.

Optimizavus tolydžią tinklo reprezentaciją ją reikia dekoduoti atgal į diskrečiąją reprezentaciją, o tada ir panaudojama dekodavimo komponentė. Ši dalis priima du argumentus: pirmasis yra atnaujinto tinklo tolydi reprezentacija, kurią šis mechanizmas bando paversti jau į diskrečią architektūrą, o antrasis argumentas – kodavimo mechanizmo užslėptoji būsena, kuri padeda dekoduoti šį tinklą panašų į prieš tai duotą kodavimo mechanizmui. Pats dekodavimo mechanizmas susideda iš dviejų dalių: viena dalis – tai LTSM modulis, kuris naudojamas ir kodavimo fazėje, tik šis veikia priešingai – konvertuoja iš tolydaus į diskrečią architektūrą, o antrasis tai dėmesio mechanizmas, kuris, turint kodavimo fazės užslėptas būsenas, gali tiksliau grąžinti panašią į pradinę architektūrą. Tad galiausiai tikėtina, kad gaunamas pagerintas tinklas.

Galiausiai patys paieškos komponentai yra tam tikri neuroniniai tinklai, tad ir šiuos reikia apmokyti, norint gauti gerus jų rezultatus. Iš tikrųjų egzistuoja du praradimai, kuriuos reikia minimizuoti – L_{pp} bei L_{rec}. Kur L_{pp} – tai tolydžiai užkoduotos architektūros įvertinimo praradimas, o tiksliau, tai yra skirtumas tarp įvertinto klasifikavimo teisingumo ir tikrojo teisingumo. Kitas įvertis L_{rec} – tai grąžinimo atgal į diskrečiąją architektūrą praradimas, kuris, tiksliau tariant, yra neteisingumas dekoduojant architektūrą, kai yra žinoma jos tikroji tolydi reprezentacija. Tad bendra praradimo formulė (žr. 4-ą formulę) susiveda į bendrą šių dviejų praradimų sumą, papildomai pridedant ir tam tikrą kompromiso santykį λ. Čia $E(x)$ yra kodavimo funkcija, f – tai prognozuotojo įverčio funkcija ir P_D – tai dekodavimo funkcijos tikimybė, jog tai x architektūrą, turint užkoduota architektūrą x.

$$L = \lambda L_{pp} + (1-\lambda)L_{rec} = \lambda \sum_{x \in X} (s_x - f(E(x))^2 + (1-\lambda) \sum_{x \in X} \log P_D(x|E(x)) \tag{4}$$

Norint apmokyti paieškos komponentus sudaromas atsitiktinių architektūrų sąrašas. Toliau įvertinamas kiekvienos architektūros iš sąrašo klasifikavimo teisingumas, tada pasinaudojus šiais dviem sąrašais bandoma apmokyti kodavimo, prognozuotojo bei dekodavimo mechanizmus, kad šie galėtų pakankamai gerai įvertinti naujų architektūrų klasifikavimo efektyvumą.

2.2.2. Algoritmo eiga

Šias komponentes, neuroninius tinklus, apmokius iki nustatyto baigimo kriterijaus, galima pradėti ieškoti optimalios, geriausią klasifikavimo teisingumą turinčios, architektūros. Pats baigimo kriterijus šiame algoritme pasirinktas tik vienas – tai pasiektas tam tikras epochų skaičius. Ieškant optimalios architektūros, yra atsitiktinai sugeneruojamos architektūros, o vėliau ieškoma tikėtinai pagerintos jų versijos. Kiekviena architektūra iš sąrašo užkoduojama bei įvertinamas jos klasifi-

kavimo neteisingumas (angl. *error rate*). Šią metriką galima naudoti kaip praradimo funkcijos reikšmę, todėl galima optimizuoti turimą tinklo architektūrą. Pati optimizacija vykdoma naudojant gradientinio nusileidimo metodą ir optimizuojama tinklo architektūros tolydžioji reprezentacija pagal jos praradimo (funkcijos) reikšmę. Papildomai turi būti nurodoma šio (gradiento) žingsnio dydis ir (gradiento) nusileidimo kryptis. Atnaujinta reprezentacija dekoduojama ir gauta naujoji architektūra išsaugoma į tą patį architektūrų sąrašą.

2.2.3. Primityvios operacijos

Naudojant bendrąjį tinklą, taip pat beieškant architektūros, naudojamos jau iš anksto su parametrais apibrėžtos konvoliucinių neuroninių tinklų operacijos. Viena vertus tai padeda išvengti dar daug parametrų, kurios reikia optimizuoti. Kita vertus nėra žinoma, ar naudojamos operacijos tikrai gali būti optimalios, kitaip tariant, galbūt apibrėžtos operacijos, gali būti prastos nuotraukų klasifikavimui [WAR+20]. Vis dėlto, šiuo atveju pasirinktos tokios operacijos, kurios dažniausiai naudojamos konvoliuciniuose neuroniniuose tinkluose, o ir šios operacijos tik šiek tiek skiriasi nuo kito algoritmo. Iš tikrųjų šio algoritmo operacijos yra labai panašios į naudojamas **DARTS** algoritme [LSY19] - tai konvoliucijos bei sujungimo sluoksnio operacijos. Vis tik konvoliucijos operacija turi tik du dydžius 3x3 bei 5x5, o ir papildomai pridėta identiteto operacija, kurios **DARTS** algoritme nebuvo. Tad naudojantis šiomis operacijomis yra ieškoma tokia operacijų kombinacija, kad klasifikavimo teisingumas būtų geriausias.

3. Algoritmų tyrimas

Šiame skyriuje aprašyta aplinkos paruošimo procesas prieš atliekant eksperimentus. Aprašyme nurodytos detalės, kad šiuos tyrimus būtų galima atkartoti.

3.1. Architektūrų paieška bei tikrinimas

Šis skyrius susidaro iš dviejų poskyrių: pirmame aprašyta pačių architektūrų paieška, kadangi kiekvieno algoritmo paieškos metodas ir eiga skirtingi, paieška taip pat turi būti implementuota skirtingai. Kitame poskyryje aprašytas šių architektūrų tikrinimas bei apmokymas, jame yra tik vienas skirsnis, kadangi gautų architektūrų panaudojimas jau yra suvienodintas, todėl tikrinimui naudojama ta pati implementacija, jai perduodant rastą architektūrą.

Kiekvienas iš algoritmų buvo suprogramuotas su „Python“ programavimo kalba pasinaudojus dalinai jau egzistuojančiomis implementacijomis – algoritmo komponenčių klasėmis. Iš tikrųjų kiekviena iš algoritmų paieškos eigų jau buvo implementuota autoriaus, atsižvelgiant į jau egzistuojančią implementaciją – komponenčių apmokymą, pačių architektūrų paieška bei bendrą veikimą. Vis dėlto programiniame kode buvo iš naujo tarpusavyje suderinti parametrai bei kiti įrankiai.

3.1.1. Paieška

Autorius implementavo algoritmus taip, kad kiekvienas iš jų gauna įrankius su vienodais parametrais: optimizuotoją, (mokymosi greičio) planuotoją bei praradimo funkciją. Optimizavimui buvo naudojamas gradientinio nusileidimo su momentu metodas. Papildomai optimizuotojui buvo naudojamas mokymosi greičio planuotojas, o šiam buvo pasirinktas kosinuso atkaitinimo (angl. Cosine Annealing) metodas. Greičio planuotojas buvo pasirinktas todėl, kad beieškant koeficientų tokiame uždavinyje funkcija, kuri yra labai netolygi, būtų greičiau sukonverguota į ieškoma optimumą [LH16]. Praradimo funkcijai buvo pasirinkta naudoti informacijos skirtumo (angl. cross entropy) metriką.

3.1.1.1. NAO

NAO paieškos įgyvendimui buvo pasinaudota dalimi šio algoritmo implementacijos [1]. Iš tikrųjų šio darbo implementacijai panaudoti tokie failai: kodavimo (angl. encoder), dekodavimo (angl. decoder), valdiklio (angl. controller), operacijų (angl. operations), modelio (angl. model) ir modelio paieškos (angl. model search). Šie failai panaudoti todėl, kad pats algoritmas nebuvo detaliai aprašytas [LTQ+19] darbe, taip pat dėl to, kad algoritmas ištestuotas, žinoma, kad jo veikimas turėtų būti panašus į aprašytą straipsnyje. Dėl minėtųjų priežasčių buvo pasirinkta naudoti jau implementuotą algoritmą, o tiksliau jo dalį – klases.

Kiekviena NAO paieškos epocha prasideda apmokant bendrąjį neuroninį tinklą su visomis galimomis operacijomis. Vis dėlto, jis apmokomas atsitiktinai traukiant architektūras iš architektūrų

[1] https://github.com/renqianluo/NAO_pytorch/tree/master/NAO_V1

sąrašo ir panaudojant tik toje architektūroje aprašytas operacijas, o ne visas vienu metu (esančias bendrajame tinkle). Vėliau, po šio apmokymo, tikrinama, ar dabartinė epocha yra įvertinimo epocha. Jeigu ne, tuomet pradedama nauja epocha. Kitu atveju toliau bandomas įvertinti, o tiksliau aproksimuojamas, visų architektūrų teisingumas, kurios yra architektūrų sąraše. Po šio įvertinimo seka jau pačio kodavimo-prognozuotojo-dekodavimo komponenčių mokymas su turimomis aproksimacijomis bei architektūromis. Pasibaigus šių komponenčių apmokymui sudaromas naujasis sąrašas su 100 geriausiai įvertintų architektūrų iš bendrojo architektūrų sąrašo. Pasinaudojus pastaruoju sąrašu generuojama tiek naujų architektūrų, kiek yra nurodyta pagal „naujų architektūrų“ parametrą. Prieš sudedant naujai gautas architektūras į naujų architektūrų sąrašą tikrinama, ar nauja sugeneruota architektūra dar neegzistuoja bendrųjų architektūrų sąraše. Atlikus visus šiuo veiksmus naujų architektūrų sąrašas kartu sumaišomas su bendruoju architektūrų sąrašu. Tam, kad bendrasis architektūrų sąrašas nesikeistų, senosios, prasčiausiai pasirodžiusios architektūros, ištrinamos, taip pritaikant elitizmą. Galiausiai iš naujo pradedama naujoji epocha.

Vėliau, po visų epochų įvykdymo, išrenkamos 5 didžiausią teisingumą turėjusios architektūros ir sukuriami modeliai, kurie turi tik architektūroje nurodytas operacijas. Tad galiausiai algoritmo paieška grąžina šias 5-ias architektūras, modelius.

3.1.1.2. CNAS

CNAS paieškai įgyvendinti buvo pasinaudota dalimi šio algoritmo implementacijos [2]. Šio darbo implementacijai buvo panaudoti tokie failai: architektūros (angl. architecture), operacijų (angl. operations), modelio (angl. model) bei modelio paieškos (angl. model search). Šie failai panaudoti todėl, kad pačio algoritmo aprašymo straipsnyje [WZL+19] nebuvo. Taip pat šį algoritmą, straipsnio autoriai jau yra ir ištestavę, ir, žinoma, jo veikimas turėtų būti labai panašus į aprašytą straipsnyje. Todėl šią implementaciją, o tiksliau jos dalį – klases, ir pasirinkta naudoti šiame straipsnyje.

Kiekviena CNAS paieškos epocha prasideda ieškant architektūros, o vėliau ją apmokant. Iš tikrųjų ieškant iškviečiamas architektūros optimizatorius, kuris, atsižvelgiant į antrosios eilės parametrą, arba pritaiko praradimo funkcija pagal gautų nuotraukų kiekvieno pikselio reikšmes ir tikimasi reikšmę, arba, jeigu įjungta antroji eilė, papildomai prie pritaikymo dar suskaičiuoja kiekvienos operacijos svorio atnaujinimą (nedaroma prielaida, kad dabartiniai architektūros operacijų svoriai yra optimalūs). Vėliau po šio architektūros optimizatoriaus žingsnio atnaujinami ir pačių operacijų svoriai. Galiausiai užbaigiant epochą įvykdomas dabartinės bendros architektūros įvertinimas ir iš naujo pradedama nauja epocha.

Po visų epochų įvykdymo iš bendrosios architektūros išvedamos tokios operacijos, kurios turi didžiausią operacijos tikėtinumo svorį. Tuomet ir sukuriama nauja architektūra-modelis, tik su atitinkamomis operacijomis, kurią vėliau, prieš tikrinant, reikia apmokyti.

[2] https://github.com/tianbaochou/CNAS

3.1.2. Apmokymas ir tikrinimas

Apmokymas ir tikrinimas suprogramuoti pačio autoriaus, kadangi algoritmų grąžinami modeliai, gali būti pritaikomi visur. Įvykdžius paiešką grąžinamas architektūrų-modelių sąrašas. Šie modeliai nėra apmokyti – svoriai nėra optimalūs, todėl prieš pradedant tikrinti pačias architektūras, jos apmokomos. Apmokymas vykdomas su 20 epochų. Toks skaičius pasirinktas todėl, kad atlikus trumpus tyrimus pastebėta, kad vieną architektūrą užtrukdavo mokyti bent apie valandą, tad nuspręsta, kad apmokius su 20 epochų, turint jau „optimalią“ architektūrą, turėtų užtekti parodyti pakankamus rezultatus lyginimui. Kiti apmokymo parametrai – optimizuotojas, (mokymosi greičio) planuotojas bei praradimo funkcija pasirinkti tokie patys kaip ir apmokymui. Optimizuotojas – gradientinio nusileidimo metodas, planuotojas – kosinuso atkaitinimo metodas, o galiausiai praradimo funkcija – informacijos skirtumo. Pasinaudojus papildomu klasifikatoriumi, gautoms modelio „praradimo“ funkcijos reikšmėms pridedamos praradimo reikšmės iš papildomo klasifikatoriaus. Galiausiai šiuos modelius apmokius juos galima patikrinti su validacijos duomenimis ir išvesti gautus įverčius.

3.2. Lyginimo kriterijai

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{8}$$

Lyginimui pasirinkti tokie matai: teisingumo (angl. accuracy) (žr. 5 formulę), tikslingumo (angl. presicion) (žr. 6 formulę), atkūrimo (angl. recall) (žr. 7 formulę), F1 (žr. 8 formulę) ir papildomai žiūrėti į laiką. Vis dėlto, į laiką žiūrima tik tada, kai eksperimentai atliekami tuo pačiu kompiuteriu. Taip pat, norint tikslingai išmatuoti užimtą laiką, reikėtų atlikti daug tyrimų su tais pačiais parametrais ir imti jų vidurkį. Vis dėlto, dėl laiko bei resursų stokos šį kartą laikas naudojamas labiau bendram vaizdui susidaryti negu kaip tikslus matas.

Teisingumas yra vienas dažniausiai naudojamų matų vertinant modelių klasifikavimo efektyvumą [Tha20]. Vis dėlto, šis matas gali paslėpti tam tikras klasifikavimo detales, kurių galima nepamatyti nepasigilinus į kitus klasifikavimo rodiklius. Dirbant su daugiau negu 2-omis ar 3-omis klasėmis gali iškilti problema, kad klasifikuojant daugumą klasių gerai, o vieną ar dvi blogai – teisingumo matas gali rodyti, kad klasifikuojama gerai [Tha20]. Tad naudojantis vien tik šiuo matu galima nepamatyti tikrojo klasifikavimo detalių. Be šios problemos egzistuoja dar ir kita: naudo-

jantis išbalansuotais duomenų rinkiniais ir blogai klasifikuojant kažkurią klasę su mažai duomenų, šis matas gali rodyti, kad pats klasifikavimas nėra blogas. Vis dėlto, pastaroji problema šiame tyrime neturėtų egzistuoti, kadangi naudojami duomenų rinkiniai yra subalansuoti. Vis tik tam, kad būtų išspręsta bei eitų pamatyti pirmąją problemą, taip pat matyti ir tikslesnį klasifikavimo vaizdą, pasinaudota tikslingumo, atkūrimo matais ir F1 rodikliu. Kadangi F1 rodiklis yra tikslingumo bei išsamumo matų (harmoninis) vidurkis, lyginimuose naudojamas šis rodiklis. Kiekvienai iš klasių šie matai suskaičiuojami atskirai.

3.3. Naudojami duomenų rinkiniai

Šiame tyrime naudojami trys duomenų rinkiniai algoritmų lyginimui: CIFAR10[3], CIFAR100[3] ir Tiny ImageNet[4]. Verta atsižvelgti į tai, kad atitinkamai kiekviename rinkinyje: CIFAR10, CIFAR100 bei ImageNet, yra vis didėjantis klasių skaičius bei vis mažėjantis nuotraukų skaičius per klasę. Viena vertus tai leidžia pamatyti, kaip kiekvienas iš algoritmų sugeba prisitaikyti su vis didėjančiu klasių skaičiumi bei mažėjančiu nuotraukų kiekiu klasėse. Kita vertus, kai abu parametrai keičiasi, sunku įvardinti, kuris iš šių parametrų turi įtakos galutiniams rezultatams. Vis dėlto, šis tyrimas labiau lygina abiejų algoritmų prisitaikymą prie skirtingų parametrų, tad šių dviejų parametrų pasikeitimas didelės įtakos neturi.

3.3.1. CIFAR10 ir CIFAR100

Šie du rinkiniai yra bene dažniausiai naudojami neuroninių tinklų tyrimuose. Juos abu sudaro 60 tūkst. nuotraukų, kurios buvo surinktos iš „TinyImages“ rinkinio. Šių dviejų duomenų rinkinių skirtumas yra klasių bei nuotraukų skaičius jose. CIFAR10 turi 10 klasių bei po 6 tūkst. nuotraukų kiekvienoje iš klasių, kur CIFAR100 turi 100 klasių ir tik 600 nuotraukų kiekvienoje klasėje. Abiejuose rinkiniuose naudojamos tokios klasės, kurios dažniausiai sutinkamos kasdieniniame gyvenime, todėl tai palankūs duomenys, kai norima išmokyti neuroninį tinklą klasifikuoti „buitiškas“ nuotraukas. Taip pat šis rinkinys visiems viešai prieinamas, todėl jį gali naudoti bet kas.

3.3.2. Tiny ImageNet

„Tiny ImageNet“ rinkinys – tai „ImageNet“ rinkinio poaibis, kuris vietoje 14 milijonų nuotraukų turi tik 100 tūkstančių bei vietoje 1 tūkst. klasių turi tik 200. ImageNet yra vienas garsiausių rinkinių dėl savo klasių įvairovės ir nuotraukų kiekio. Naudojantis šiuo duomenų rinkiniu, kiekvienais metais rengiamas neuroninių tinklų modelių konkursas, kurio metu lyginami kiekvieno iš modelių gebėjimai klasifikuoti šį duomenų rinkinį. Kadangi šis rinkinys naudojamas konkurse lyginti modelius, nuspręsta naudoti jį ir šiame tyrime lyginant du algoritmus. Vis dėlto šis rinkinys nėra visiškai laisvai prieinamas, reikia prašyti prieigos.

[3]https://www.cs.toronto.edu/ kriz/cifar.html
[4]https://www.kaggle.com/datasets/akash2sharma/tiny-imagenet

3.4. Techninės tyrimo detalės

Tyrimų vykdymui naudoti du kompiuteriai, kad būtų galima atlikti daugiau tyrimų, taip pat atlikti juos greičiau. Vis dėlto, šie kompiuteriai nėra vienodi, tad jų architektūrų radimo greitis gali skirtis.

Vienas iš kompiuterių – tai asmeninis kompiuteris, kuris savyje turi Intel I7-7700HQ procesorių, 16 GB operatyviosios atminties bei NVIDIA GTX 1060 vaizdo plokštę. Kompiuteryje naudojama Linux operacinė sistema su Kubuntu 22.04 distribucija. Vykdant eksperimentą buvo išjungtos programos, kurios nėra būtinos pačiai operacinei sistemai, kad šį būtų galima atkartoti panašiu laiku.

Kitas kompiuteris – tai VU informacinių technologijų atviros prieigos centro superkompiuteris. Jame užsakyti keturi branduoliai Intel Xeon E5-2698 procesoriaus, naudota 32 GB operatyviosios atminties ir viena NVIDIA DGX-1 Tesla V100 32GB vaizdo plokštė. Šiame kompiuteryje įdiegta Qlustar 11 operacinė sistema, Ubuntu 18.04 pagrindu.

Abu algoritmai įgyvendinti Python programavimo kalba, pasinaudojus PyTorch biblioteka, karkasu. Karkasas buvo naudojamas tam, kad kurti konvoliucinius neuroninius tinklus būtų paprasta ir šiuos būtų galima iškart persiųsti į vaizdo plokštę, kad ten vyktų jų apmokymas.

3.5. Algoritmų aplinkos sulyginimas

Dauguma kiekvieno iš algoritmo naudojamų parametrų ir įrankių buvo suvienodinti šio darbo testavimo aplinkoje, kad tiriamų algoritmų veikimas būtų kuo panašesnis. Neišėjo sulyginti tik tokių parametrų, kurie yra išskirtinai kiekvieno algoritmo specifika: NAO – NAO komponenčių parametrai, o CNAS – optimizuotojo parametrai.

Paleidus kiekvieną algoritmą, naudojamose bibliotekų atsitiktinių skaičių generatoriuose nustatyta ta pati „sėkla“ (angl. seed). Taip pat naudojamoje CUDAnn bibliotekoje nustatyta, kad perdavus veiksmus atlikti į vaizdo plokštę, šie visuomet būtų atliekami deterministiškai. Toje pačioje bibliotekoje įjungtas palyginimo rėžimas, kuris turėtų pagreitinti pačių algoritmų veikimą, kadangi nuotraukų įėjimo dydis yra nekintantis.

Kiekviename iš algoritmų suvienodinti pradiniai operacijų svoriai naudojantis HE svorių inicijavimo metodu [HZR+15]. Abiems algoritmams gali būti įjungiamas papildomas klasifikatorius (angl. Auxiliary classifier), kuris yra pridedamas tinklų gale. Tai padaryta tam, kad galimai būtų pagerintas klasifikavimo teisingumas.

Abu algoritmai naudoja sumažintų celių konceptą. Šis konceptas sumažina nuotraukų rezoliucija perpus, taip išryškinant bendresnes objektų savybes nuotraukose. Kiekviename iš algoritmų paprastos celės yra verčiamos į sumažintas, kai jų pozicija yra $\frac{1}{3}$ arba $\frac{2}{3}$ iš visų celių.

3.6. Pasirinkti parametrai

Dauguma parametrų pasirinkti tokie, kokie naudoti šių algoritmų straipsnių bandymuose. Vis dėlto ne visi parametrai sutapo, tad tokius parametrus bandyta sulyginti per vidurį, jeigu pati parametro specifika tą leido, arba pasirinkti vieno iš algoritmo straipsnių parametrai, jeigu buvo manoma arba buvo atlikti tyrimai, kad tai parodys geresnius rezultatus.

Pasirinktus aplinkos parametrus, kurie nustatyti visuose tyrimuose, galima matyti 1, 2 ir 3 lentelėse.

1 lentelė. Bendri parametrai

Parametras	**Šio tyrimo**	**CNAS tyrimo**	**NAO Tyrimo**
Paketo dydis[5]	32	32	64
Pradinis kanalų kiekis	20	24	20
Sluoksnių (celių) skaičius[6]	3	6	3
Operacijų skaičius[6]	5	4	5
Gradiento viršutinis apribojimas	5	5	5
Epochos[7]	50	60	150

2 lentelė. Architektūros optimizavimo parametrai

Parametras	**Šio tyrimo**	**CNAS tyrimo**	**NAO Tyrimo**
Svorių nykimas	0.0003	0.0003	0.0003
Mokymosi greitis	0.025	0.025	0.025
Minimalus mokymosi greitis	0.003	0.003	0.001
Momentas	0.9	0.9	0.9
CNAS antrosios eilės aproksimacija[8]	Išjungta	Išjungta	—

[5]Šitoks paketo dydis pasirinktas todėl, kad tai didžiausias dydis, kuris neviršijo asmeninio kompiuterio vaizdo plokštės atminties, taip pat su tokiu dydžiu tinklai apmokomi pakankamai greitai.

[6]Kaip ir paketo dydžio atveju, pasirinkta naudoti mažesnį šių parametrų kiekį, kad asmeninis kompiuteris galėtų ieškoti architektūros su vaizdo plokštės pagalba.

[7]Atlikus trumpus tyrimus pastebėta, kad 50 epochų yra pakankamas kiekis įvertinti abiejų architektūrų klasifikavimą. Įvykdžius daugiau epochų, tinklas jau daugiau nebesimokydavo. Taip buvo tikriausiai dėl to, kad pats nuotraukų kiekis kiekvienam rinkiniui sumažintas tik iki visų galimų 80% dėl laiko stokos.

[8]Pasak [WZL+19] straipsnio autorių, šis parametras architektūros paiešką pailgina dvigubai, tačiau reikšmingų rezultatų nesuteikia.

3 lentelė. NAO algoritmo parametrų parinkimas.

Parametras	Šio tyrimo	CNAS tyrimo	NAO Tyrimo
NAO bendri parametrai			
NAO paketo dydis architektūrai	100	—	100
NAO kompromiso santykis	0.8	—	0.8
NAO epochų kiekis	100	—	100
NAO pradinių architektūrų kiekis	600	—	600
NAO naujų architektūrų kiekis	300	—	300
NAO įvertinimo epocha[9]	kas 5	—	kas 30
NAO dekodavimo neuroninių tinklų parametrai			
Kelio numetimo tikimybė	0	—	0
Paslėptas neuronų kiekis	96	—	96
Sluoksnių	1	—	1
Dekodavimo ilgis	40	—	40
Dekodavimo abėcėlės dydis	12	—	12
NAO kodavimo neuroninių tinklų parametrai			
Kelio numetimo tikimybė	0	—	0
Paslėptas neuronų kiekis	96	—	96
Sluoksnių	1	—	1
Kodavimo ilgis	20	—	20
Dekodavimo abėcėlės dydis	12	—	12
Įterpimo dydis	48	—	48
NAO prognozuotojo neuroninių tinklų parametrai			
Svorių numetimas	0.1	—	0.1
Paslėptas neuronų kiekis	200	—	200
Sluoksnių	3	—	3
Šaltinio (kodavimo) ilgis[10]	40	—	40
NAO komponenčių optimizatoriaus parametrai			
Svorių nykimas	0.0001	—	0.0001
Mokymosi greitis	0.0003	—	0.001

Norint palyginti du algoritmus nuspręsta keisti šių parametrų reikšmes:

- Paketo dydžio parametro tiriamos tokios reiškmės: 16, 32 ir 64. Pasak straipsnio [ML18] autorių, mažesnis paketo dydis turi nemažą įtaką geriau generalizuojant klasės savybes giliųjų tinklų modeliams. Papildomai [KMN+16] atlikti eksperimentai parodė, kad dideli paketo dydžiai veda link blogesnės generalizacijos.
- Įjungti papildomus klasifikatorius kiekvienam iš algoritmų. Tikėtasi, kad įjungus papildomą klasifikatorių šis padės algoritmams rasti geresnę architektūrą.
- Epochų dydį pakeisti nuo 50 iki 100. Norėta pakeisti iki 200, tačiau po 100 epochų rezultatai reikšmingai nesikeitė, tad ties 100 ir buvo sustota. Iš tikrųjų tikėtasi, kad didinant epochų dydį bus geriau klasifikuojama, kadangi turima daugiau laiko įsisavinti klasių savybes.

[9] NAO tyrime pačios architektūros buvo įvertintos tik kas 30 epochų, tačiau iš viso epochų buvo 150. Atitinkamu santykiu sumažinti epochų ir jų įvertinimo epochų skaičiai.

[10] Iš kodavimo gauta 20 ilgio užkoduota architektūra „padvigubinama“. T.y. gautas vektorius yra dubliuojamas.

- Galiausiai naudotos nuotraukų (atsitiktinės) iškarpos su dydžiu 16. Tikėtasi, kad panaudojus nuotraukų iškarpas modeliai sugebės geriau išmokti klasių savybes, kadangi perėjus visas nuotraukas galės geriau įsisavinti atributus, kai nėra papildomos informacijos.

4. Eksperimentų rezultatai

Šiame skyriuje aprašyti gauti ir išanalizuoti eksperimentų rezultatai. Bendrai atlikti visus tyrimus užtruko apie mėnesį.

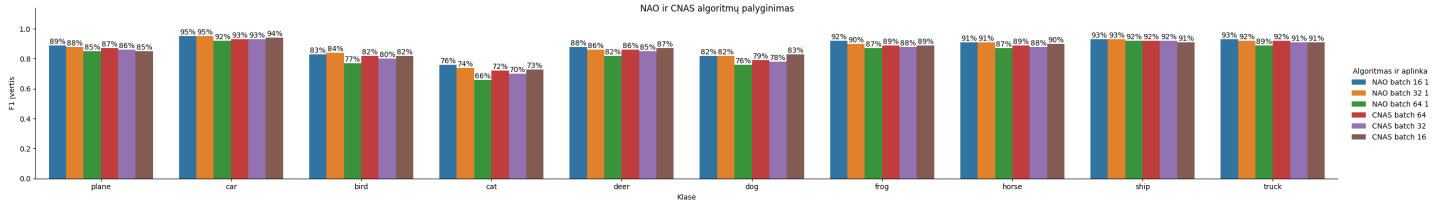
NAO algoritmas grąžindavo daugiau negu 1 architektūrą, vis dėlto pasirinkta naudoti tik vieną (geriausią) architektūrą, kai lyginama su CNAS algoritmo grąžinta architektūra. Geriausia architektūra iš NAO pasirinkta pagal jos klasių teisingumo bei F1 įvertį.

4.1. Paketo dydis

Įvykdžius eksperimentus su skirtingais paketo dydžiais NAO algoritme gauti rezultatai, kurie atitiko straipsnio [ML18] gautas išvadas – naudojantis mažesniais paketo dydžiais geriau generalizuojamos klasių savybės. Taip pat NAO algoritmas sugebėjo surasti geresnę architektūrą negu CNAS, kai paketo dydis buvo 32. Kitu atveju, kai paketo dydis buvo 64, CNAS sugebėjo pernokti NAO algoritmą. Galiausiai paketo dydį nustačius į 16 abu algoritmai pasirodė panašiai.

Sumažinus paketo dydį iki 16, CNAS ir NAO algoritmo rastų architektūrų klasifikavimo teisingumas padidėjo: architektūros atitinkamai buvo įvertintos 87% ir 88% teisingumu. Paketo dydžiui esant 32, teisingumo įverčiai tapo 85% ir 88%, o nustačius paketo dydį į 64 teisingumas įvertintas 86% ir 84%.

CNAS algoritmo atveju paketo dydį nustačius į 32 jo teisingumas sumažėjo palyginus su 64 paketo dydžiu, tačiau tokio rezultato tikimasi nebuvo, tad šiam reiškiniui išsiaiškinti reikėtų daugiau tyrimų. Kita vertus NAO algoritmo architektūros klasifikavimo teisingumas atitinkamai didėjo mažinant paketo dydį ir atvirkščiai, kas ir atitiko [ML18] straipsnio gautas išvadas.



2 pav. Surastų architektūrų klasių klasifikavimas pagal F1 rodiklį

Taip pat galima pastebėti iš 2 pav. esančio grafiko, kad abiejų algoritmų rastos architektūros su paketo dydžiu 16 (žr. mėlyną bei rudos spalvos stulpelius) daugumoje klasių sugebėjo gauti geresnius rezultatus negu tų pačių algoritmų architektūros su didesniu paketo dydžiu. Verta pastebėti ir tai, kad daugumoje klasių NAO paketo dydis (žr. mėlyną, geltoną bei žalias stulpelius) turi žymiai didesnę įtaką negu CNAS (žr. raudoną, violetinę bei rudą stulpelius) – įverčiai tarp skirtingų paketo dydžių yra didesni.

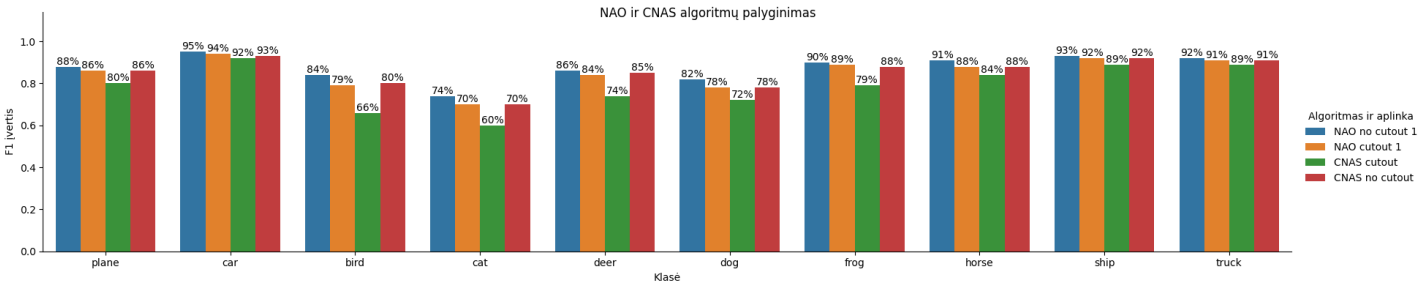
Atsižvelgus į 2 pav. esantį grafiką, o tiksliau, žiūrint atitinkamai į NAO stulpelius – mėlyną, geltoną bei žalią, ir CNAS – raudoną, violetinę bei rudą, galima teigti, kad netgi ir didinant bei mažinant paketo dydį, NAO algoritmas nėra dėsningas nugalėtojas prieš CNAS algoritmą. Galima pastebėti, kad CNAS algoritmo rasta architektūra sugebėjo pasiekti panašius įverčius kaip NAO,

kai paketo dydis 16 (žr. mėlyną bei rudą stulpelius), tad šiuo atveju architektūros pasirodo panašiai, o NAO šioks toks pagerėjimas, gali būti paprasčiausiai triukšmas. Kita vertus, kai paketo dydis yra 64 (žr. žalią bei raudoną stulpelius), atrodo, kad CNAS algoritmas netgi sugeba šiek tiek aplenkti NAO algoritmą ir pagal teisingumo įvertį, ir pagal F1 įvertį kiekvienoje iš klasių. Vis dėlto, NAO algoritmas pasirodo geriau prieš CNAS, kai paketo dydis nustatomas į 32 (žr. geltoną bei violetinį stulpelius).

Nors sumažintas paketo dydis ir šiek tiek pagerino klasifikavimo įverčius, laikas, trunkantis rasti tokias architektūras, padidėjo beveik dvigubai. CNAS algoritmas užtruko: su paketo dydžiu 16 – 1 dieną bei 15 valandų; su paketo dydžiu 32 – 21 valandą; su paketo dydžiu 64: 10 valandų. Tuo tarpu NAO užtruko: su paketo dydžiu 16 – 1 dieną bei 8 valandas; su paketo dydžiu 32 – 15 valandų; su paketo dydžiu 64 – 8 valandas.

4.2. Nuotraukų iškarpos

Pasinaudojus (atsitiktinėmis) nuotraukų iškarpomis vietoje pilnų nuotraukų, pastebėtas klasifikavimo teisingumo suprastėjimas: CNAS ir NAO atitinkamai įvertinti 79% bei 85%. Tikėtasi, kad panaudojus nuotraukų iškarpas bus geriau įsisavinamos kiekvienos klasės savybės, kadangi galima išvengti papildomos, perteklinės informacijos.



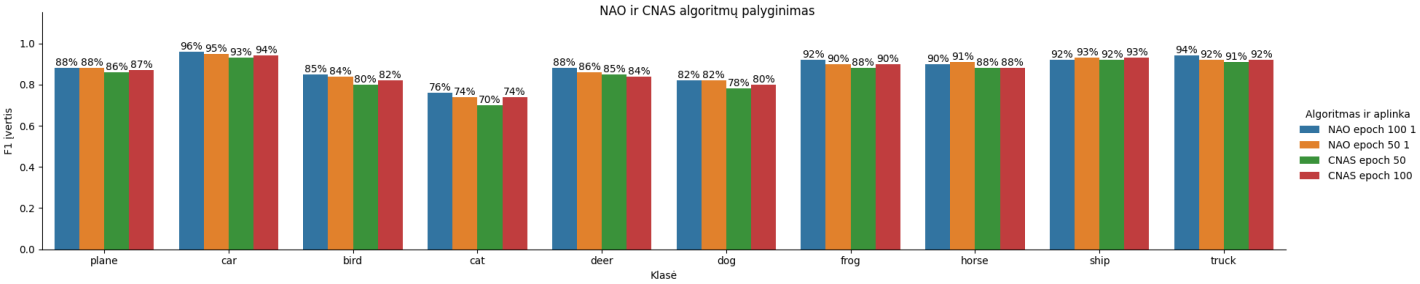
3 pav. Surastų architektūrų klasių klasifikavimas pagal F1 rodiklį

Visgi net ir suprastėjus abiejų algoritmų architektūrų klasifikavimui, NAO algoritmas sugebėjo ir pasirodyti geriau (žr. geltonos spalvos stulpelį), ir neturėti tokio didelio suprastėjimo kaip CNAS algoritmas (žr. žalią spalvos stulpelį su raudonu). Tai yra, atsižvelgus į 3 pav. esantį grafiką, galima matyti, kad CNAS algoritme panaudojus iškarpas (žr. žalia spalvos stulpelį) šis duoda žymiai prastesnius rezultatus negu jų išvis nenaudojant (žr. raudoną spalvos stulpelį).

Galima teigti, kad panaudojus nuotraukų iškarpas šių algoritmų klasifikavimas – gebėjimas generalizuoti klasės savybes – suprastėja. Taip pat galima teigti, kad net ir panaudojus jas NAO algoritmas pasirodo geriau už CNAS algoritmą, taip pat NAO algoritmui iškarpų naudojimas neturi tokios didelės (prastinančios) įtakos.

4.3. Epochos

Įvykdžius eksperimentus tarp 50 ir 100 epochų pastebėta, kad NAO architektūros teisingumas nepasikeitė ir liko 88%, o CNAS algoritmo architektūros teisingumas pakilo nuo 85% iki 86%. Vis dėlto, nors NAO teisingumas ir nepasikeitė, daugumoje klasių šio algoritmo rastos architektūros F1 rodiklis truputį pagerėjo (žr. 4 pav. esančio grafiko mėlyną ir žalią stulpelius) palyginus 100 epochų su 50. Taip pat NAO algoritmo rasta architektūra įvykdžius 100 epochų šiek tiek lenkia CNAS algoritmo rastą architektūrą.



4 pav. Rastų architektūrų klasių klasifikavimas pagal F1 rodiklį

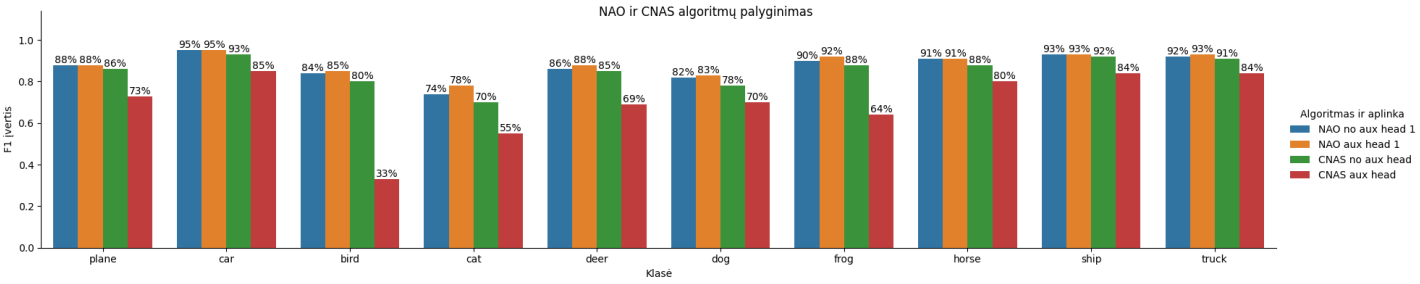
Atsižvelgus į 4 pav. esantį grafiką, tikėtina, kad net ir padidinus epochų skaičių į 100 (nuo 50), NAO algoritmas išlaiko pranašumą prieš CNAS algoritmą. Vis tik tai galėjo būti ir triukšmas, tad reikėtų atlikti daugiau tyrimų, kad tai būtų galima patvirtinti. Kita vertus, įvykdžius daugiau epochų - 100, CNAS algoritmas taip pat šiek tiek pagerino rezultatus palyginus su 50 epochų.

4.4. Papildomas klasifikatorius

Įjungus papildomą klasifikatorių, rezultatas kiekvienam iš algoritmų buvo priešingas: klasifikavimo teisingumas NAO algoritmui pakilo iki 89% nuo įprastų 88%, o CNAS algoritmui sumažėjo nuo 85% iki 71%.

Tikėtina, kad tai nutiko dėl pačių algoritmų specifikos, tačiau tai reikėtų patikrinti: NAO algoritmas naudoja svorių pasidalijimo komponentę, kuri bando aproksimuoti architektūros teisingumą. Papildomo klasifikatoriaus pridėjimas prie visų operacijų minėtąją aproksimacijos paklaidą sumažina, tad ir padeda rasti geresnę architektūrą – geriau „išmokyti“ neuroninius tinklus, kurie ir ieško architektūros. Kita vertus CNAS algoritmas bando optimizuoti/rasti tokias operacijas, kurios daugiausiai „prisideda“ prie klasifikavimo. Dėl šios priežasties, kai pridedamas klasifikatorius, didžiąją dalį klasifikavimo „teisingumo“ suteikia pridėtas klasifikatorius, tad atitinkamai kitos operacijos yra mažiau įvertinamos, kad prisideda prie rezultato. Vis dėlto šias hipotezės reikėtų tikrinti, tačiau tai turėtų būti skirta ateities tyrimams.

Nepaisant to, žiūrint į 5 pav. esantį grafiką, o tiksliau, geltoną bei raudoną stulpelius, galima teigti, kad NAO algoritmas gali geriau pasirodyti negu CNAS įjungus papildomą klasifikatorių. Taip pat galima pastebėti, kad įjungus papildomą klasifikatorių NAO algoritmas tik dar labiau patobulėjo, kai CNAS algoritmas gerokai suprastėjo.



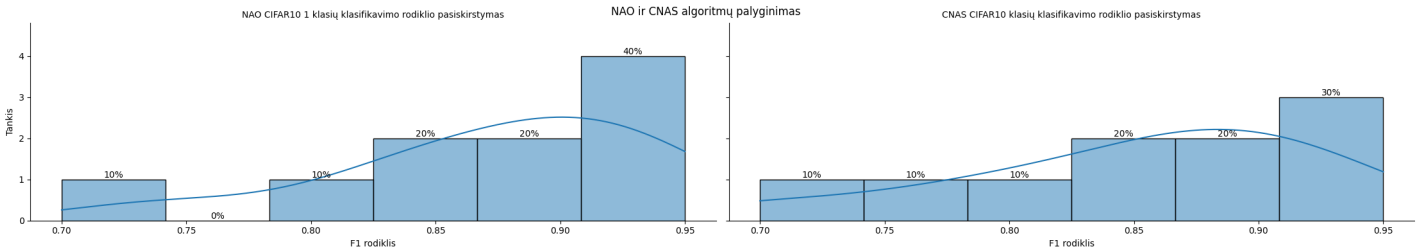
5 pav. Surastų architektūrų klasių klasifikavimas pagal F1 rodiklį

4.5. Duomenų rinkiniai

Išbandžius abu algoritmus su kitais duomenų rinkiniais gautas žymiai prastesnis klasifikavimo teisingumas negu su CIFAR10 duomenų rinkiniu. Vis dėlto šitoks teisingumo įverčio suprastėjimas nebuvo netikėtas, kadangi ir pačių klasių daugiau, o ir nuotraukų per klasę yra mažiau. Dėl šių faktorių galimybė, kad pavyks įsisavinti kiekvienos klasės savybes per 50 epochų bei su kitais nustatytais tyrimo parametrais, yra labai maža.

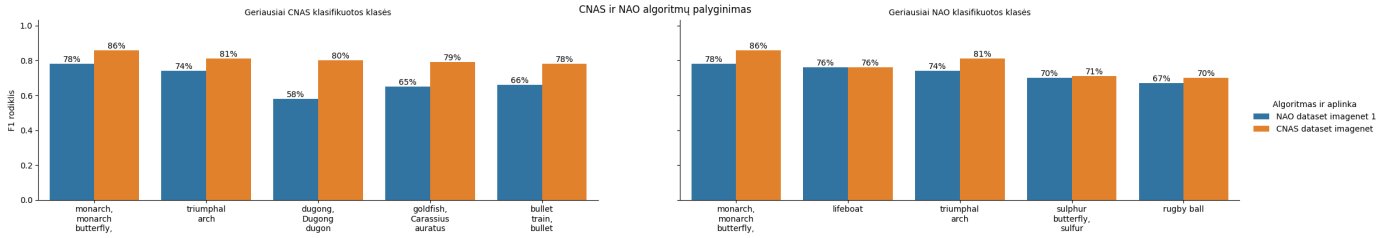
Atlikus paieška su CIFAR100 duomenų rinkiniu gauti tokie klasifikavimo įverčiai: NAO ir CNAS algoritmas atitinkamai gavo 62% bei 61%. Kitu atveju, atlikus paieška su Tiny ImageNet rinkiniu, gauti tokie įverčiai: 39% bei 46%.

Galima pastebėti, kad CNAS rasta architektūra su kitais duomenų rinkiniais yra žymiai arčiau arba lenkia NAO rastą architektūrą teisingumo įverčio požiūriu. Vis dėlto, reikėtų daugiau tyrimų su šiais duomenų rinkiniais, kadangi gali būti, kad NAO algoritmas su kitokiais parametrais – daugiau epochų, architektūrų, celių, kitokiu paketo dydžiu ar kitais panašiais parametrais galėtų pranokti CNAS algoritmą. Atsižvelgiant į tai būtų galima teigti, kad CNAS algoritmas tam tikra prasme yra adaptyvesnis skirtingų duomenų rinkinių atžvilgiu negu NAO. Ypač lyginant 6, 12 bei 9 pav. esančius grafikus, kuriuose galima matyti kiekvieno algoritmo rastos architektūros su skirtingais duomenų rinkiniais F1 įverčio pasiskirstymus. NAO algoritmas su CIFAR10 rinkiniu surenka daugumą įverčių dešinėje (žr 6 pav.), kai CNAS algoritmo įverčiai labiau pasiskirstę tarp 70% ir 95%. Vis tik ši tendencija pasikeičia ir abu algoritmai, tiksliau, jų klasifikavimo įverčių pasiskirstymas, susilygina panaudojus CIFAR100 rinkinį (žr. 12 pav.). Galiausiai, pasižiūrėjus į 9 pav. esančius grafikus, galima pastebėti, kad Tiny ImageNet rinkinyje CNAS algoritmas surenka daugiau įverčių dešinėje negu NAO algoritmas.

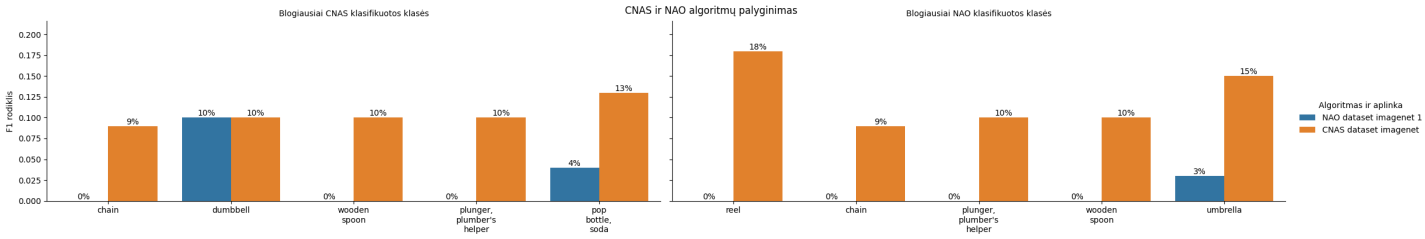


6 pav. F1 įverčio pasiskirstymas

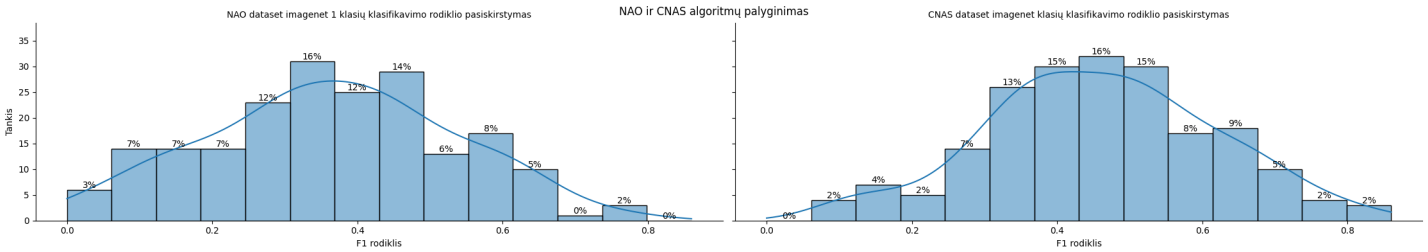
4.5.1. Tiny ImageNet



7 pav. Geriausiai klasifikuojamos 5 klasės pagal F1 rodiklį



8 pav. Blogiausiai klasifikuojamos 5 klasės pagal F1 rodiklį



9 pav. F1 įverčio pasiskirstymas

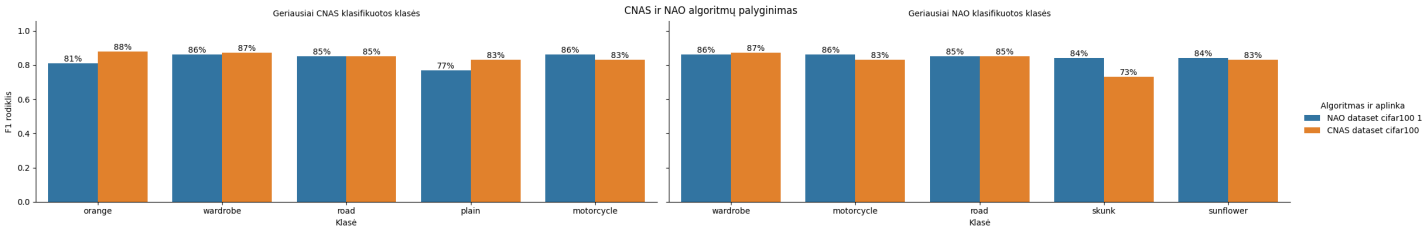
Atsižvelgus į 8 pav. esančius grafikus, galima pamatyti, kad CNAS rasta architektūra šiame rinkinyje net ir blogiausiu atveju surinko bent 10% F1 įvertinimo. Kita vertus daugelyje klasių NAO rastos architektūros įvertinimas buvo paprasčiausiai 0%.

Atkreipus dėmesį į 7 pav. kairėje esantį grafiką, galima pastebėti, kad tarp NAO 5-iose geriausiai įvertintų klasių 4-iose iš jų CNAS rasta architektūra gerokai lenkia kito algoritmo rastą architektūrą.

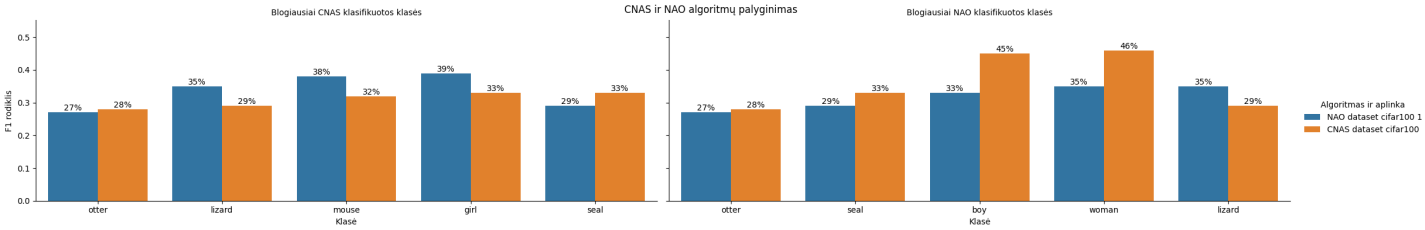
Pažiūrėjus į 9 pav. esančius grafikus, galima pastebėti, kad dešinėje esantis grafikas, reprezentuojantis NAO architektūros F1 įverčių pasiskirstymą, turi mažesnį kiekį elementų kairėje negu dešinėje esantis grafikas. Tai reiškia, kad CNAS algoritmas daugiau klasių suklasifikavo geriau.

Galiausiai galima būtų teigti, kad su tiek epochų ir kitais numatytais parametrais CNAS algoritmas pranoksta NAO. Vis dėlto, vertinant absoliučiai pagal teisingumo ir kitus įverčius, CNAS rasta architektūra pasirodė pakankamai prastai. Taip pat CNAS algoritmas užtruko dvigubai ilgiau negu NAO ieškodamas architektūros. CNAS algoritmas užtruko 3 dienas ir 5 valandas, o NAO algoritmas 1 dieną ir 12 valandų.

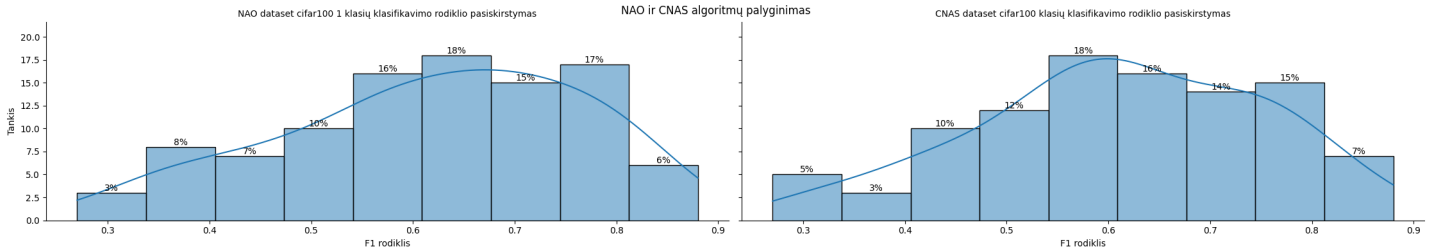
4.5.2. CIFAR100



10 pav. Geriausiai klasifikuojamos 5 klasės pagal F1 rodiklį



11 pav. Blogiausiai klasifikuojamos 5 klasės pagal F1 rodiklį



12 pav. F1 įverčio pasiskirstymas

Pažvelgus į 11 pav. esančius grafikus, galima pamatyti, kad blogiausiai klasifikuojamosiose klasėse nėra daug išsiskiriančių dalykų: kur viena iš architektūrų pasirodo prasčiau, ją aplenkia kita, išskyrus paskutinius atvejus. Paskutinėse klasėse kitos architektūros įvertinimas būna prastesnis.

Pažiūrėjus į 10 pav. esančius grafikus, pasirodo įdomus dalykas: pirmoje klasėje, kurią geriausiai suklasifikavo NAO rasta architektūra, vis tik lenkia CNAS architektūra, nors kitose klasėse NAO algoritmas arba pasirodo taip pat, arba šiek tiek geriau. Vis dėlto, panašus dalykas nutinka ir su CNAS algoritmu jo geriausiai klasifikuojamose klasėse, tačiau tai nutinka tik paskutinėje klasėje. Visur kitur CNAS algoritmas šiek tiek lenkia arba turi panašų įvertį kaip NAO rastos architektūros.

Apibendrinus galima teigti, kad šie algoritmai per daug nesiskiria nei geriausiai, nei blogiausiai klasifikuojamose klasėse. Vis tik atsižvelgus į 12 pav. esančius grafikus, galima pamatyti, kad NAO rastos architektūros įverčiai yra šie tiek labiau pasislinkę link dešinės, kas reiškia, kad daugiau klasių pavyko suklasifikuoti geriau negu CNAS architektūrai. Tai kažkiek patvirtina ir suskaičiuotas aritmetinis vidurkis, mediana bei moda, atitinkamai CNAS ir NAO algoritmai gavo: 0.6111, 0.61, 0.61; 0.6193, 0.64, 0.67, tačiau šie įverčiai gali būti labiau triukšmas negu reikšti

realų pokytį. Taip pat laikas, kurį užtrunka įvykdyti šiuos algoritmus, skiriasi daugiau negu dvigubai: CNAS algoritmas užtruko 22 valandas, o NAO algoritmas 10 valandų. Vis tik galima būtų teigti, kad CIFAR100 rinkinyje su numatytais tyrimo parametrais NAO algoritmas randa panašią architektūrą kaip ir CNAS algoritmas klasifikavimo įverčių prasme.

Rezultatai ir išvados

Atlikus architektūros paieškos algoritmų apžvalgą, konvoliucinių neuroninių tinklų apžvalgą, sukūrus aplinką bei įvykdžius eksperimentus gauti tokie rezultatai:

- Gauti kiekvieno iš algoritmo rastų architektūrų įverčiai panaudojus papildomus įrankius bei keičiant parametrų reikšmes: paketo dydžio, epochų dydį, įjungus papildomą klasifikatorių, keičiant duomenų rinkinius bei galiausiai naudojant nuotraukų iškarpas.

Iš šių rezultatų gautos tokios išvados:

- Labiau rekomenduojama naudoti NAO algoritmą negu CNAS, kai norima naudoti 32 paketo dydį su kitais numatytais parametrais. Tyrimo metu daugumoje klasių rastų architektūrų teisingumo įvertis bei F1 rodiklis buvo geresni naudojantis NAO negu CNAS algoritmu. Kitais atvejais, kai paketo dydis 64, tikėtina, kad CNAS algoritmas pasirodys geriau. Atlikus tyrimus pastebėta, kad architektūros teisingumo įvertis bei F1 rodiklis buvo geresni pasinaudojus CNAS algoritmu. Nustačius paketo dydį į 16, abi architektūros sugeba pasirodyti panašiai. Atlikti tyrimai parodė, kad jų įverčiai yra labai panašūs. Vis tik CNAS algoritmas užtrunka dvigubai ilgiau.
- NAO algoritmo rezultatai sutapo su [ML18] straipsnio gautais rezultatais: mažinant paketo dydį neuroninis tinklas gali geriau generalizuoti klasės savybes. Atlikus tyrimus pastebėta, kad klasifikavimo teisingumas bei kiekvienos klasės F1 įvertis gerėjo vis mažinant paketo dydį nuo 64 iki 16. Vis tik CNAS atveju šie rezultatai skyrėsi ir nesutapo su straipsnio gautais rezultatais. Su paketo dydžiu 32 CNAS algoritmas pasirodė prasčiau, negu kai paketo dydis buvo 64. CNAS atveju reikėtų atlikti daugiau tyrimų, kurie padėtų išsiaiškinti šį neatitikimą.
- Siūloma nenaudoti nuotraukų iškarpų be originalių nuotraukų šiuose algoritmuose, kadangi tai tik suprastina nuotraukų klasifikavimą. Kiekvienam iš algoritmų rastos architektūros teisingumo įvertis suprastėjo: NAO nuo 88% iki 85%, o CNAS 85% iki 79%. Taip pat, net jei ir naudojamos tokios nuotraukų iškarpos, labiau patartina naudoti NAO algoritmą, kadangi jo įvertis nukrito mažiau – tik apie 3 punktus, kai CNAS nukrito netgi 6.
- Tikėtina, kad geriau naudoti NAO algoritmą negu CNAS, kai didinamas epochų kiekis su šiame tyrime numatytais parametrais. NAO algoritmas prieš CNAS pasirodė šiek tiek geriau ir teisingumo įverčio požiūriu, jų įverčiai atitinkamai 88% ir 86%, ir kiekvienos klasės F1 įverčiu. Vis tik tai galėjo būti labiau triukšmas ir reikėtų atlikti daugiau tyrimų, kad būtų galima teigti labiau užtikrintai, kad NAO algoritmas pasirodo geriau.
- Įjungus papildomą klasifikatorių rekomenduojama naudoti NAO algoritmą. NAO algoritmas pasirodė žymiai geriau su įjungtu papildomu klasifikatoriumi negu CNAS: CNAS gavo teisingumo įvertį 71%, o NAO 89%.
- Papildomas klasifikatorius pagerina NAO algoritmą, o CNAS blogina. Rastos NAO architektūros teisingumo įvertis pakilo nuo 88% iki 89%. CNAS atveju klasifikatorius gerokai pablogino teisingumo įvertį nuo 85% iki 71%.
- CIFAR100 duomenų rinkinio atveju abu algoritmai pasirodo panašiai su numatytais para-

metrais. Tyrimo metu abi architektūros gavo panašius įverčius: NAO – 62%, o CNAS – 61%. Atsižvelgus ir į įverčių pasiskirstymą, ir į vidurio taško įverčius, galima matyti, kad reikšmingo skirtumo nėra.

- Tiny ImageNet duomenų rinkinio atveju CNAS algoritmas pasirodo geriau negu NAO algoritmas. CNAS algoritmas įvertintas 46%, o NAO tik 39%. Taip pat atsižvelgus į F1 įverčių pasiskirstymą, galima pastebėti, kad CNAS algoritmo atveju įverčiai yra labiau pasislinkę link 100%, o NAO atveju labiau link 0%.
- Keičiant duomenų rinkinius, kurie didina klasių skaičių ir mažina nuotraukų skaičių per klases su tais pačiais parametrais, galima pastebėti, kad CNAS algoritmas yra adaptyvesnis. Tyrimų metu pastebėta, kad CNAS algoritmo rastų architektūrų teisingumo įverčiai tarp skirtingų duomenų rinkinių skyrėsi mažiau negu NAO algoritmo atveju.

Reziumė

The paper compares two algorithms: NAO and CNAS. Both algorithms search for convolutional neural network architectures. Comparison is made by evaluating the found architecture on a dataset and then deriving an F1-score from classification metrics while also considering time. In the paper, the comparison is made by changing packet size and epoch parameters, using additional tools: auxiliary classificator and image cutouts, and different datasets. The results show that the NAO algorithm performed better using the CIFAR10 dataset with cutouts, auxiliary classificator, and likely with smaller packet size and more epochs. However, the CNAS algorithm was superior when using the CIFAR10 dataset with a larger packet size and the Tiny ImageNet dataset. Also, the findings show that CNAS is more adaptive to changes in the number of classes and the number of images per class. In other cases, both algorithms performed similarly.

Literatūra

[AMA17a] Saad Albawi, Tareq Abed Mohammed ir Saad Al-Zawi. Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, p.p. 1–6, 2017. DOI: 10.1109/ICEngTechnol.2017.8308186.

[AMA17b] Saad Albawi, Tareq Abed Mohammed ir Saad Al-Zawi. Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*, p.p. 1–6. Ieee, 2017.

[Dič05] Valdas Dičiūnas. Algoritmų analizės pagrindai : mokymo priemonė, 2005. URL: http://www.mif.vu.lt/katedros/cs/Asmen/algoritmu_analize.pdf. 700 kB.

[EMH19] Thomas Elsken, Jan Hendrik Metzen ir Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[HSA+19] Jie Hu, Li Shen, Samuel Albanie, Gang Sun ir Enhua Wu. Squeeze-and-Excitation Networks, 2019. arXiv: 1709.01507 [cs.CV].

[HZR+15] Kaiming He, Xiangyu Zhang, Shaoqing Ren ir Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015. arXiv: 1502.01852 [cs.CV].

[KMN+16] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy ir Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, 2016. DOI: 10.48550/ARXIV.1609.04836. URL: https://arxiv.org/abs/1609.04836.

[LBB+98] Y. Lecun, L. Bottou, Y. Bengio ir P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. DOI: 10.1109/5.726791.

[LH16] Ilya Loshchilov ir Frank Hutter. SGDR: Stochastic Gradient Descent with Warm Restarts, 2016. DOI: 10.48550/ARXIV.1608.03983. URL: https://arxiv.org/abs/1608.03983.

[LMK15] Zhouhan Lin, Roland Memisevic ir Kishore Konda. How far can we go without convolution: Improving fully-connected networks. *arXiv preprint arXiv:1511.02580*, 2015.

[LSY19] Hanxiao Liu, Karen Simonyan ir Yiming Yang. DARTS: Differentiable Architecture Search, 2019. arXiv: 1806.09055 [cs.LG].

[LTQ+19] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen ir Tie-Yan Liu. Neural Architecture Optimization, 2019. arXiv: 1808.07233 [cs.LG].

[LVG+20] Walter Hugo Lopez Pinaya, Sandra Vieira, Rafael Garcia-Dias ir Andrea Mechelli. Chapter 10 - Convolutional neural networks. Andrea Mechelli ir Sandra Vieira, redaktoriai, *Machine Learning*, p.p. 173–191. Academic Press, 2020. ISBN: 978-0-12-815739-8. DOI: https://doi.org/10.1016/B978-0-12-815739-8.00010-9. URL: https://www.sciencedirect.com/science/article/pii/B9780128157398000109.

[ML18] Dominic Masters ir Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks, 2018. DOI: 10.48550/ARXIV.1804.07612. URL: https://arxiv.org/abs/1804.07612.

[ON15] Keiron O'Shea ir Ryan Nash. An Introduction to Convolutional Neural Networks, 2015. arXiv: 1511.08458 [cs.NE].

[PGZ+18] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le ir Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing, 2018. arXiv: 1802.03268 [cs.LG].

[RXC+20] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Xiaojiang Chen ir Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *arXiv preprint arXiv:2006.02903*, 2020.

[SD08] SN Sivanandam ir SN Deepa. Genetic algorithm optimization problems. *Introduction to genetic algorithms*, p.p. 165–209. Springer, 2008.

[SZ15] AlaEldin S. ir Yun Zhang. A Review on Back-Propagation Neural Networks in the Application of Remote Sensing Image Classification. *Journal of Earth Science and Engineering*, 5, 2015-01. DOI: 10.17265/2159-581X/2015.01.004.

[Tha20] Alaa Tharwat. Classification assessment methods. *Applied Computing and Informatics*, 2020.

[WAR+20] Yuan Wen, Andrew Anderson, Valentin Radu, Michael F.P. O'Boyle ir David Gregg. TASO: Time and Space Optimization for Memory-Constrained DNN Inference. *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, p.p. 199–208, 2020. DOI: 10.1109/SBAC-PAD49847.2020.00036.

[WZL+19] Yu Weng, Tianbao Zhou, Lei Liu ir Chunlei Xia. Automatic Convolutional Neural Architecture Search for Image Classification Under Different Scenes. *IEEE Access*, 7:38495–38506, 2019. DOI: 10.1109/ACCESS.2019.2906369.

[ZZL+17] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin ir Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices, 2017. arXiv: 1707.01083 [cs.CV].