

**KAUNO TECHNOLOGIJOS UNIVERSITETAS**

**MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS**

**Vytautas Kraujalis**

Studijų modulio

P160M121 Duomenų tyrybos metodai ir programinės priemonės

**Laboratorinis darbas nr. 3**

Dėstytojas: dr. M. Kavaliauskas

**KAUNAS, 2022**

# DUOMENŲ RINKINYS

Pasirinkite duomenų rinkinį su vaizdais priklausančiais vienai iš kelių klasių. Rekomenduojamas vaizdų dydis nuo 50x50 iki 250x250 taškų dydžio ir šimto ar daugiau vaizdų kiekvienai klasei. Pasirinktą duomenų rinkinį pasižymėkite Moodle pateiktame dokumente. Duomenų rinkiniai tarp studentų negali sutapti.

# UŽDUOTIS

1. **Neuroniai tinklai**

Savo pasirinktam duomenų rinkiniui iš bandykite bent 2 tiesioginio sklidimo (feed-forward) ir bent 4 skirtingas sąsūkų neuroninių tinklų architektūras. Stenkitės parinkti tinklą tokį, kad gautumėte kuo geresnį klasifikavimo tikslumą. Pateikite gautus rezultatus kiekvienam iš neuroninių tinklų. Tinklą nusakykite sluoksnių kiekiu, neuronų kiekiu sluoksniuose, filtrų kiekiu ar dydžiu, aktivacijos funkcijos naudojimu, apjungimo sluoksniu, etc., o jų gautą rezultatą nusakykite klasifikavimo tikslumu (ant test imties), sumaišymo matrica ir apmokymo trukme. Pateikite tikslumo kitimo mokymosi metu grafiką. Apibūdinkite, ar tinklai linkę persimokyti. Aprašykite sukumus su kuriais susidūrėte ir kaip juos sprendėte.

1. **Neuroninių tinklų perpanaudojimas**

Sudarykite neuroninį tinklą savo pasirinktiems vaizdamas klasifikuoti. Tinklą sudarykite naudodami *imagenet* tinklo sąsūkų sluoksnius bei papildomai apmokydami savo sukurtą pilnai sujungtą tinklo dalį. Išbandykite dvi pilnai sujungto tinklo dies alternatyvas. Analogiškai, kaip ir pirmoje užduotyje, pateikite gautus rezultatus.

Apibendrinkite gautus rezultatus, pateikite išvadas.

# BENDRI REIKALAVIMAI

1. Pasižymėkite savo naudojamą duomenų rinkinį Moodel dokumente.
2. Projektas ataskaita turi būti patalpinama Moodle aplinkoje.
3. Projekto atskaitos prieduose turi būti pateikiamas pilnas R kodas, kurį galima būtų įvykdyti be klaidų ir gauti projekto aprašyme pateiktus rezultatus.

Galimi duomenų rinkinių šaltiniai:

* https://www.kaggle.com/tags/image-data

**ATASKAITA**

**DUOMENŲ RINKINYS**

Duomenų rinkinys paimtas iš: <https://www.kaggle.com/datasets/crowww/a-large-scale-fish-dataset>

Duomenų rinkinys susideda iš 9 klasių:

0 – black sea sprat,

1 - gilt head bream,

2 – hourse mackerel,

3 – red mullet,

4 – red sea bream,

5 – sea bass,

6 – shrimp,

7 – striped red mullet,

8 – trout.

Kiekviena klasė turi po 1000 vaizdų. Šiuose vaizduose jau yra atlikta vaizdų augmentacija, kai tie patys vaizdai yra pasukti.

lentelė 1 Duomenų rinkinio klasių pavyzdžiai

|  |  |  |
| --- | --- | --- |
| Klasė |  |  |
| 0 |  |  |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |

**UŽDUOTIS 1. Sąsūkų neuroniniai tinklai**

**Tiesioginio sklidimo pilnai sujungto neuroninio tinklo architektūra nr. 1.**

Sudarėme paprastą tiesioginio sklidimo neuroninį tinklą, sudarytą iš 4 sluoksnių (lentelė žemiau). Pirmasis sluoksnis mūsų duomenis tik paverčia 1d vektoriumi, todėl jis parametrų ir neturi. Kadangi mūsų duomenys yra vaizdai, o juos visus konvertuoju į 150x150 dimensijas turinčius spalvotus vaizdus. Todėl gauname, kad mūsų įėjimo sluoksnyje yra 150 \* 150 \* 3 (kadangi RGB vaizdai) = 67500 įvestys.

Antrame sluoksnyje turime 32 neuronus, kurie yra pilnai sujungti su įvesties neuronais. Todėl šiame žingsnyje turime 32 (neuronai) \* (67500 (įvesties neuronai) + 1 (konstanta)) = 2160032 parametrus. Naudojome “ReLu” aktyvacijos funkciją.

Trečiame sluoksnyje turime 16 neuronų, kurie yra pilnai sujungti su 2 sluoksniu. Šiame sluoksnyje turime 16 (neuronai) \* (32 (2 sluoksnio neuronai kaip įvestys) + 1 (konstanta)) = 528 parametrus. Naudojome “ReLu” aktyvacijos funkciją.

Ketvirtas sluoksnis yra išėjimo sluoksnis, kuris susideda iš 9 neuronų (kadangi turime 9 klases), šis sluoksnis taip pat pilnai sujungtas su 3 sluoksniu, todėl turėsime 9 \* (16 + 1) = 153 parametrus. Naudojome “Softmax” aktyvacijos funkciją.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

flatten\_11 (Flatten) (None, 67500) 0

dense\_28 (Dense) (None, 32) 2160032

dense\_27 (Dense) (None, 16) 528

dense\_26 (Dense) (None, 9) 153

===========================================================

Total params: 2,160,713

Trainable params: 2,160,713

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Neuroninis tinklas iš architektūros yra labai paprastas, tačiau net ir paprastas tinklas su tokiu įvesties formatu turi net 2160713 parametrus.

Apmokydami tinklą, eigoje gauname:

Chart, line chart

Description automatically generated

pav. Pirmo tiesioginio sklidimo pilnai sujungto neuroninio tinklo mokymosi procesas

Apmokymas truko 565,45 sek. (9,4 min.). Kaip matome, eigoje tinklui visiškai nepavyksta rasti tinkamų parametrų reikšmių. Kiekviena epocha beveik niekuo nesiskiria nuo kitos.

Išbandę modelį ant patvirtinamosios imties, gauname 11,11% tikslumą, tai jau yra žymiai blogiau nei atsitiktinis spėjimas. Kaip matome iš sumaišymo matricos (paveiksliukas žemiau), tinklas kiekvieną vaizdą klasifikuoja kaip 1 klasę (Gilt-Head Bream). Todėl net neapsimoka skaičiuoti kitų statistikų, kai tinklas yra visiškai nepanaudojamas.

Calendar

Description automatically generated

pav. Sumaišymo matrica nr. 1

Sekantį tinklą pamėginsime sudaryti sudėtingesnį.

**Tiesioginio sklidimo pilnai sujungto neuroninio tinklo architektūra nr. 2.**

Sudarėme tiesioginio sklidimo pilnai sujungtą neuroninį tinklą su 4 sluoksniais. Pirmasis sluoksnis sudarytas tik iš vaizdų konvertavimo į vektorių, o paskutinis sluoksnis – išėjimo sluoksnis. Iš viso šiame tinkle turime 17314313 parametrų. Viduriniuose sluoksniuose naudojome „ReLu“ aktyvacijos funkcijas, o išėjimo sluoksnyje – „Softmax“.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

flatten\_14 (Flatten) (None, 67500) 0

dense\_40 (Dense) (None, 256) 17280256

dense\_39 (Dense) (None, 128) 32896

dense\_38 (Dense) (None, 9) 1161

===========================================================

Total params: 17,314,313

Trainable params: 17,314,313

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Stebėdami mokymosį procesą (paveikslas žemiau), matome žymiai geresnį vaizdą nei prieš tai naudotojo architektūroje. Matome, kad tikslumas ant apmokymo imties vis didėja, tačiau tikslumas ant patvirtinamosios imties po kelių epochų pradeda mažėt, dar ankščiau pradeda didėti patvirtinamosios imties nuostolių funkcijos reikšmės. Galim teigti, kad modelis per daug apsimoko ant apmokymo imties.

Chart, line chart

Description automatically generated

pav. Antro tiesioginio sklidimo pilnai sujungto neuroninio tinklo mokymosi procesas

Šį tinklą apmokyti prireikė 614,39 sek. (10,24 min.). Modelis sudėtingesnis, nei pirmoje dalyje naudota architektūra, apmokymas truko tik šiek tiek ilgiau, o tikslumas ant patvirtinamosios imties siekia 54,8%. Tačiau toks tikslumas yra tik šiek tiek geresnis nei atsitiktinis sprendimas.

Calendar

Description automatically generated

pav. Sumaišymo matrica nr. 2

7 klasės teisingų spėjimų buvo tik 21 (iš 200), šios klasės jautrumas siekia tik 0,10. Šią klasę tinklas dažniausiai sumaišė su 5 klase.

A fish on a blue surface

Description automatically generated with medium confidence

pav. 7 klasės pavyzdys

A picture containing text, fish

Description automatically generated

pav. 5 klasės pavyzdys

A picture containing text, plaque

Description automatically generated

pav. Klasifikavimo statistikos nr. 2

Apskritai, šis tinklas rodo gana didelį specifiškumą, kadangi visų klasių specifiškumas siekia >0,94 ir tik 2 klasė (0,81) ir 5 klasė (0,77) turi mažesnį specifiškumą.

**Tiesioginio sklidimo pilnai sujungtų neuroninių tinklų išvada.**

Tokio tipo architektūra neparodė gerų rezultatų, sudėtingesnės architektūros nepavyko sudaryti dėl resursų trūkumo, kuomet R programa neleido sudaryti sudėtingesnio tinklo su daugiau neuronų per kelis sluoksnis dėl atminties trūkumo. Kas buvo pastebėta, jog tinklas sugeba klasifikuoti teisinga linkme tik tuomet, kada sluoksnyje prieš išėjimą turime mažiausiai 128 neuronų sluoksnį (jei naudoju tik neuronų skaičių, kurie gaunami keliant 2 laipsniu). Taip pat rezultatas nemažai priklausė ir nuo „batch size“ parinkimo, kuomet naudojame didesnį vaizdų kiekį, rezultatai yra prastesni, bei nuo vaizdų dydžio. Buvo bandyta naudoti 200x200 RGB vaizdus, tačiau apmokymas trūko pernelyg ilgai.

**Sąsūkų neuroninio tinklo architektūra nr. 1.**

Sudarėmė paprastą sąsūkų neuroninį tinklą sudarytą iš 10 sluoksnių (tik 4 sluoksniuose turime parametrus). Tinklas sudarytas iš 2 sąsūkų sluoksnių (32 ir 16 dydžių, abu naudoja 3x3 filtro dydžius, „ReLu“ aktyvacijos funkcijas, vaizdų kraštus užpildysim nuliais ir judėsime tarp pikselių vienu žingsniu), 2 apjungimo sluoksnių (abiejuose sluoksniuose vaizdus mažinsime 2x2, kraštų neužpildysim ir judėsime tarp pikselių 2 žingsniais), 3 išmetimo sluoksnių (atitinkamai 0.3, 0.3 ir 0.5 dalis išmesim), 1 išlyginimo sluoksnis (pereinam iš vaizdo į vektorių) ir pilnai sujungto sluoksnio (su 16 neuronų ir „ReLu“ aktyvacijos funkcija) su išėjimo sluoksniu.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

conv2d\_1 (Conv2D) (None, 150, 150, 32) 896

max\_pooling2d\_1 (MaxPooling2D) (None, 75, 75, 32) 0

dropout\_2 (Dropout) (None, 75, 75, 32) 0

conv2d (Conv2D) (None, 75, 75, 16) 4624

max\_pooling2d (MaxPooling2D) (None, 37, 37, 16) 0

dropout\_1 (Dropout) (None, 37, 37, 16) 0

flatten (Flatten) (None, 21904) 0

dense\_1 (Dense) (None, 16) 350480

dropout (Dropout) (None, 16) 0

dense (Dense) (None, 9) 153

===========================================================

Total params: 356,153

Trainable params: 356,153

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iš viso tinkle turime 356153 parametrus. Prieš išlyginant vaizdus, paskutiniame apjungimo sluoksnyje gauname 37x37 matricas.

Tinklas gana paprastas, todėl ir iš mokymosi proceso (paveiklas žemiau) matome, kad tikslumas yra gana mažas. Taip pat matosi, jog tinklas per mažai apsimoko, turime patvirtinamosios imties tikslumą didesnį nei apmokymo imties.

Chart, line chart

Description automatically generated

pav. Sąsūkų neuroninio tinklo mokymosi procesas nr. 1

Apmokymas truko 839,75 sek. (13,9 min.). Apmokymo tikslumas siekia 37,6%, o patvirtinamosios imties – 70,4%.

Iš patvirtinamosios imties tikslumo ir sumaišymo matricos (paveiklas žemiau) iš pirmo žvilgsnio atrodo kaip neblogas tinklas, tačiau apmokymo imties tikslumas yra labai prastas, o taip pat ir su patvirtinamąja imtimi turime, jog 6 klasės jautrumas yra lygus 0, tinklui visiškai nepavyko atpažinti 6 klasės teisingai.

Calendar

Description automatically generated

pav. Sąsūkų NN sumaišymo matrica nr. 1

**Sąsūkų neuroninio tinklo architektūra nr. 2.**

Sudarėmė šiek tiek sudėtingesnį sąsūkų neuroninį tinklą sudarytą iš 12 sluoksnių (tik 5 sluoksniuose turime parametrus). Tinklas sudarytas iš 2 sąsūkų sluoksnių (padidinome dydžius ir apkeitėme didėjimo tvarką iki 32 ir 64 dydžių, abu naudoja 3x3 filtro dydžius, „ReLu“ aktyvacijos funkcijas, vaizdų kraštus užpildysim nuliais ir judėsime tarp pikselių vienu žingsniu), 2 apjungimo sluoksnių (abiejuose sluoksniuose vaizdus mažinsime 2x2, kraštų neužpildysim ir judėsime tarp pikselių 2 žingsniais), 4 išmetimo sluoksnių (atitinkamai 0.3, 0.3, 0.5 ir 0.5 dalis išmesim), 1 išlyginimo sluoksnis (pereinam iš vaizdo į vektorių) ir 2 pilnai sujungtų sluoksnių (padidinome neuronų skaičių iki 128 ir 64 neuronų ir „ReLu“ aktyvacijos funkcija) su išėjimo sluoksniu.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

conv2d\_3 (Conv2D) (None, 150, 150, 32) 896

max\_pooling2d\_3 (MaxPooling2D) (None, 75, 75, 32) 0

dropout\_6 (Dropout) (None, 75, 75, 32) 0

conv2d\_2 (Conv2D) (None, 75, 75, 64) 18496

max\_pooling2d\_2 (MaxPooling2D) (None, 37, 37, 64) 0

dropout\_5 (Dropout) (None, 37, 37, 64) 0

flatten\_1 (Flatten) (None, 87616) 0

dense\_4 (Dense) (None, 128) 11214976

dropout\_4 (Dropout) (None, 128) 0

dense\_3 (Dense) (None, 64) 8256

dropout\_3 (Dropout) (None, 64) 0

dense\_2 (Dense) (None, 9) 585

===========================================================

Total params: 11,243,209

Trainable params: 11,243,209

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Kadangi turėjome palyginus mažesnį kiekį parametrų, padidinome paskutinio pilnai sujungto sluoksnio neuronų skaičių, jog gautume didesnį kiekį parametrų. Kadangi tinklas „underfittino“, bandėmę sudaryti sudėtingesnį tinklą. Gavome tinklą su 11243209 parametrais.

Palyginę šio tinklo mokymosi procesą (paveikslas žemiau) su prieš tai esančiu tinklu, matome jog šiam tinklui daug geriau sekėsi mokytis. Tačiau kaip ir prieš tai, matome, jog tinklas ir dabar „underfittino“.

Chart, line chart

Description automatically generated

pav. Sąsūkų neuroninio tinklo mokymosi procesas nr. 2

Šio tinklo apmokymo procesas truko 961,98 sek. (16 min.). Apmokymo imties tikslumas – 74,5%, o patvirtinamosios imties – 87,5%.

Vėlgi, patvirtinamosios imties tikslumas yra gana neblogas, tačiau mažesnis nei apmokomosios imties, iš sumaišymo matricos (paveikslas žemiau) matosi, kad tinklui gana neblogai sekasi atpažinti vaizdus. Turime tinklą su labai aukštu specifiškumu (visos klasės turi >0.96).

Calendar

Description automatically generated

pav. Sąsūkų NN sumaišymo matrica nr. 2

A picture containing text, plaque

Description automatically generated

pav. Sąsūkų NN klasifikavimo statistikos nr. 2

Kadangi turime “underfittinimą“, bandysime šią problemą spręsti bandydami sudaryti dar sudėtingesnį tinklą. Viena iš priežaščių gali būti ir tai, jog pasižiūrėję į duomenis, turime ne tiek daug pavyzdžių kiekvienai klasei, tačiau šioje imtyje jau atlikta vaizdų augmentacija, turime tuos pačius vaizdus, tik pasuktus, todėl padidinti imtį užtruktų pernelyg ilgai.

**Sąsūkų neuroninio tinklo architektūra nr. 3.**

Sudarėmė sąsūkų neuroninį tinklą sudarytą iš 15 sluoksnių (8 sluoksniuose turime parametrus). Tinklas sudarytas iš 4 sąsūkų sluoksnių (atitinkamai 64, 64, 32 ir 32 dydžių, visi naudoja 3x3 filtro dydžius, „ReLu“ aktyvacijos funkcijas, vaizdų kraštus užpildysim nuliais ir judėsime tarp pikselių vienu žingsniu), 4 apjungimo sluoksnių (visuose sluoksniuose vaizdus mažinsime 2x2, kraštų neužpildysim ir judėsime tarp pikselių 2 žingsniais), 2 išmetimo sluoksnių (palikome išmetimo sluoksnius tik tarp pilnai sujungtų paskutinių sluoksnių atitinkamai 0.2 ir 0.2 dalis išmesim), 1 išlyginimo sluoksnis (pereinam iš vaizdo į vektorių) ir 3 pilnai sujungtų sluoksnių (padidinome neuronų skaičių iki 512, 128 ir 128 neuronų ir „ReLu“ aktyvacijos funkcija) su išėjimo sluoksniu.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

conv2d\_6 (Conv2D) (None, 150, 150, 64) 1792

max\_pooling2d\_6 (MaxPool ing2D) (None, 75, 75, 64) 0

conv2d\_5 (Conv2D) (None, 75, 75, 64) 36928

max\_pooling2d\_5 (MaxPool ing2D) (None, 37, 37, 64) 0

conv2d\_4 (Conv2D) (None, 37, 37, 32) 18464

max\_pooling2d\_4 (MaxPool ing2D) (None, 18, 18, 32) 0

conv2d\_3 (Conv2D) (None, 18, 18, 32) 9248

max\_pooling2d\_3 (MaxPool ing2D) (None, 9, 9, 32) 0

flatten\_1 (Flatten) (None, 2592) 0

dense\_7 (Dense) (None, 512) 1327616

dense\_6 (Dense) (None, 128) 65664

dropout\_3 (Dropout) (None, 128) 0

dense\_5 (Dense) (None, 128) 16512

dropout\_2 (Dropout) (None, 128) 0

dense\_4 (Dense) (None, 9) 1161

===========================================================

Total params: 1,477,385

Trainable params: 1,477,385

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Turime 1477385 parametrus.

Iš mokymosi proceso (paveikslas žemiau), matosi, jog pavyko sudaryti tinklą, kuris išsprendė „underfittinimo“ problemą, tačiau dabar tinklas persimoko.

Chart, line chart

Description automatically generated

pav. Sąsūkų neuroninio tinklo mokymosi procesas nr. 3

Šio tinklo mokymas užtruko 1173,98 sek. (19,5 min.). Apmokymo imties tikslumas – 99,1%, o patvirtinamosios imties – 84%. Iš sumaišymo matricos (paveikslas žemiau) vėl matosi, jog tinklas gana neblogai atpažįsta vaizdus, turime ir vėl aukšto specifiškumo tinklą. Tačiau kaip ir keliuose tinkluose prieš tai, turime labai prastą 7 klasės jautrumą, siekiantį tik 0,58, tinklas net 51 (iš 200) 7 klasės vaizdų priskyrė 6 klasei.

Calendar

Description automatically generated

pav. Sąsūkų NN sumaišymo matrica nr. 3

A picture containing text, plaque

Description automatically generated

pav. Sąsūkų NN klasifikavimo statistikos nr. 3

Matosi, jog sudėtingesnis tinklas davė geresnius rezultatus (apmokymo imties tikslumas >99%), todėl sekančiame tinkle pabandysime dar sudaryti sudėtingesnį tinklą, kuris gebėtų geriau atpažinti nematytus vaizdus.

**Sąsūkų neuroninio tinklo architektūra nr. 4.**

Sudarėmė sąsūkų neuroninį tinklą sudarytą iš 15 sluoksnių (8 sluoksniuose turime parametrus). Tinklas sudarytas iš 4 sąsūkų sluoksnių (atitinkamai 64, 64, 128 ir 128 dydžių, visi naudoja 3x3 filtro dydžius, „ReLu“ aktyvacijos funkcijas, vaizdų kraštus užpildysim nuliais ir judėsime tarp pikselių vienu žingsniu), 4 apjungimo sluoksnių (visuose sluoksniuose vaizdus mažinsime 2x2, kraštų neužpildysim ir judėsime tarp pikselių 2 žingsniais), 2 išmetimo sluoksnių (atitinkamai 0.2 ir 0.2 dalis išmesim), 1 išlyginimo sluoksnis (pereinam iš vaizdo į vektorių) ir 3 pilnai sujungtų sluoksnių (padidinome neuronų skaičių iki 1024, 512 ir 512 neuronų ir „ReLu“ aktyvacijos funkcija) su išėjimo sluoksniu.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

conv2d\_14 (Conv2D) (None, 150, 150, 64) 1792

max\_pooling2d\_14 (MaxPooling2D) (None, 75, 75, 64) 0

conv2d\_13 (Conv2D) (None, 75, 75, 64) 36928

max\_pooling2d\_13 (MaxPooling2D) (None, 37, 37, 64) 0

conv2d\_12 (Conv2D) (None, 37, 37, 128) 73856

max\_pooling2d\_12 (MaxPooling2D) (None, 18, 18, 128) 0

conv2d\_11 (Conv2D) (None, 18, 18, 128) 147584

max\_pooling2d\_11 (MaxPooling2D) (None, 9, 9, 128) 0

flatten\_3 (Flatten) (None, 10368) 0

dense\_15 (Dense) (None, 1024) 10617856

dense\_14 (Dense) (None, 512) 524800

dropout\_7 (Dropout) (None, 512) 0

dense\_13 (Dense) (None, 512) 262656

dropout\_6 (Dropout) (None, 512) 0

dense\_12 (Dense) (None, 9) 4617

===========================================================

Total params: 11,670,089

Trainable params: 11,670,089

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iš viso turime 11670089 parametrus.

Iš tinklo mokymosi proceso (paveikslas žemiau) matome kad tinklas gana greitai pasiekia neblogą apmokymo imties tikslumą kurį ir išlaiko tolimesnėse epochose. Mokymosi proceso metu matosi, jog tinklas persimoko.

Chart, line chart

Description automatically generated

pav. Sąsūkų neuroninio tinklo mokymosi procesas nr. 4

Tinklas mokėsi 1376,36 sek. (22,9 min.). Apmokymo imties tikslumas – 98,7%, o patvirtinamosios imties – 92,6%. Gavome modelį su mažesniu apmokymo imties tikslumu, tačiau aukštesniu patvirtinamosios imties tikslumu. Iš sumaišymo matricos (paveikslas žemiau), matome kad tinklas tikrai gana gerai sugeba klasifikuoti vaizdus. Gavome tinklą su aukštu jautrumu ir specifiškumu, išsiskiria tik ta pati 7 klasė, kurios jautrumą sugebėjom pakelti tik iki 0,65.

Calendar

Description automatically generated

pav. Sąsūkų NN sumaišymo matrica nr. 4

A picture containing text, plaque

Description automatically generated

pav. Sąsūkų NN klasifikavimo statistikos nr. 4

**Sąsūkų neuroninių tinklų išvada.**

Sąsūkų neuroniniai tinklai parodė geresnius rezultatus. Tačiau mažesnio sudėtingumo tinklai „underfittino“, o padidinę sudėtingumą, susidūrėme su nedideliu persimokymu. Persimokyma šiuo atveju bandėme spręsti tik darydami sudėtingesnį tinklą, kitas variantas, kuris nebuvo išbandytas, būtų susikurti daugiau augumentuotų vaizdų, kadangi tarp kai kurių vaizdų matosi nevienodas/vienodas apšvietimas, kas padaro vaizdus sunkiau atpažįstamus ir su žmogaus akimi. Sąsūkų neuroninių tinklų architektūroje, tuose pavyzdžiuose kuriuos bandžiau, išmetimo sluoksnis tarp sąsūkų sluoksnių pablogindavo rezultatus, nepriklausomai nuo kokio dydžio nustatome išmetimus. Tačiau išmetimo sluoksnio įtraukimas paskutiniuose, pilnai sujungtuose sluoksniuose, padėjo gauti geresnius rezultatus. Nebuvo išbandyta parinkti skirtingo dydžio filtrus dėl laiko stokos ir kompiuterio resursų taupymo, nors manau tai nebūtų dave geresnių rezultatų. Galutinis klasifikavimo tinklas atrodė gana neblogai, pažvelgus kurios klasės jautrumas yra mažas, ir peržvelgus tos klasės pavyzdžius su kitų klasių, su kuriomis tinklas sumaišo, pavyzdžiais, galima pamatyti, kad kai kurie vaizdai yra tikrai panašūs dėl apšvietimo, spalvos ar formos.

**UžDUOTIS 2. Neuroninių Tiklų perpanaudojimas**

**Sąsūkų neuroninio tinklo architektūra su perpanaudojimu nr. 1.**

Sudarėmė sąsūkų neuroninį tinklą, kurio sąsūkų dalį pasiemėme iš „imagenet“ tinklo, o savo pilnai sujungtą dalį prijungėme.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

dense\_7 (Dense) (None, 512) 1049088

dropout\_5 (Dropou t) (None, 512) 0

dense\_6 (Dense) (None, 256) 131328

dropout\_4 (Dropou t) (None, 256) 0

dense\_5 (Dense) (None, 128) 32896

dropout\_3 (Dropou t) (None, 128) 0

dense\_4 (Dense) (None, 9) 1161

===========================================================

Total params: 24,802,185

Trainable params: 1,214,473

Non-trainable params: 23,587,712

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iš viso turime 24802185 parametrus, iš kurių reikės apmokyti tik 1214473.

Iš mokymosi proceso (paveikslas žemiau) matom, jog tinklui nepavyksta gerai apsimokyti, tačiau nuostolių funkcija rodo, kad galimai padėtų epochų didinimas ir leidimas tinklui apsimokyti ilgiau, kadangi nuostolių funkcija vis mažėja. Taip pat nematom stipraus persimokymo.

Chart, line chart

Description automatically generated

pav. 19 Perpanaudotų sąsūkų neuroninio tinklo mokymosi procesas nr. 1

Apmokyti tinklą prireikė 1598,68 sek. (26,6 min.). Apmokymo imties tikslumas siekia 55,4%, o patvirtinamosios imties – 45,8%. Tiek iš tikslumų, tiek iš sumaišymo matricos (paveikslas žemiau), matom, jog tinklui nelabai gerai sekasi atpažinti vaizdus, šiuo atveju tinklo net negalime laikyti kaip aukšto specifiškumo modelio.

Calendar

Description automatically generated

pav. 20 Perpanaudotų sąsūkų neuroninio tinklo sumaišymo matrica nr. 1

Text

Description automatically generated

pav. 21 Perpanaudotų sąsūkų neuroninio tinklo klasifikavimo statistikos nr. 1

Didinti epochų skaičiaus nelabai galime dėl resursų stokos, kadangi ir taip CPU išnaudojame maksimaliu pajėgumu virš 20 minučių. Todėl pamėginsime tik šiek tiek pasunkinti paskutinius sluoksnius.

**Sąsūkų neuroninio tinklo architektūra su perpanaudojimu nr. 2.**

Dabar paskutiniamąjame sluoksnyje padidinome neuronų skaičių:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

===========================================================

dense\_11 (Dense) (None, 1024) 2098176

dense\_10 (Dense) (None, 512) 524800

dropout\_7 (Dropout) (None, 512) 0

dense\_9 (Dense) (None, 512) 262656

dropout\_6 (Dropout) (None, 512) 0

dense\_8 (Dense) (None, 9) 4617

===========================================================

Total params: 26,477,961

Trainable params: 2,890,249

Non-trainable params: 23,587,712

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Iš viso turime 26477961 parametrus iš kurių turėsime apmokyti 2890249.

Iš mokymosi proceso (paveikslas žemiau) matom, jog tinklui nelabai gerai pavyksta apsimokyti, nuostolių funkcija rodo akivaizdų persimokymą, apmokymo imties funkcija mažėja, kai tuo tarpu patvirtinimo imties funkcija nežymiai didėja.

Chart, line chart

Description automatically generated

pav. Perpanaudotų sąsūkų neuroninio tinklo mokymosi procesas nr. 2

Apmokyti tinklą prireikė 1585,13 sek. (26,4 min.). Apmokymo imties tikslumas siekia 70,8%, o patvirtinamosios imties – 51,4%. Tiek iš tikslumų, tiek iš sumaišymo matricos (paveikslas žemiau), matom, jog tinklui nelabai gerai sekasi atpažinti vaizdus. Tinklas yra tinkamas tiek pat, kiek atsitiktinis spėjimas.

Calendar

Description automatically generated

pav. Perpanaudotų sąsūkų neuroninio tinklo sumaišymo matrica nr. 2

A picture containing text, plaque, screenshot

Description automatically generated

pav. Perpanaudotų sąsūkų neuroninio tinklo klasifikavimo statistikos nr. 2

**Perpanaudotų sąsūkų neuroninių tinklų išvada.**

Naudodamiesi jau sukurtą tinklą, gerų rezultatų gauti nepavyko, mokymosi procesas tapo gana ilgu ir eksperimentuoti su sudėtingesniais tinklais tapo problematiška. Susidaro įspūdis, kad norint pasiekti patenkinamų rezultatų, reikėtų leisti tinklui mokytis ilgesnį laiką.

**PRIEDAS 1. R kodas**

library(keras)

library(tictoc)

library(imager)

set.seed(123)

# path to dataset

path <- paste0(getwd(), "/Data")

# We removed "GT" folders from the dataset

# We'll divide image pixels by 255 and fill border pixels by the nearest pixel

imageBatchGen <- image\_data\_generator(

rescale = 1./255, validation\_split = 0.2, fill\_mode = "nearest"

)

# Train dataset

train\_gen <- flow\_images\_from\_directory(

paste0(path, "/Fish\_Dataset/Fish\_Dataset"), target\_size = c(150,150),

class\_mode = "categorical", batch\_size = 32,

color\_mode = "rgb", shuffle = TRUE,

generator = imageBatchGen, subset = "training",

seed = 123

)

# Validation dataset

val\_gen <- flow\_images\_from\_directory(

paste0(path, "/Fish\_Dataset/Fish\_Dataset"), target\_size = c(150,150),

class\_mode = "categorical", batch\_size = 32,

color\_mode = "rgb", shuffle = FALSE,

generator = imageBatchGen, subset = "validation",

seed = 123

)

#--------------- feed forward 1 ---------------#

model <- keras\_model\_sequential()

model %>%

layer\_flatten(input\_shape = c(150, 150, 3)) %>%

layer\_dense(32, activation = "relu") %>%

layer\_dense(16, activation = "relu") %>%

layer\_dense(9, activation = 'softmax')

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

# 565.45 sec elapsed | 11%

#-----------------------------------------------#

#--------------- feed forward 2 ---------------#

model <- keras\_model\_sequential()

model %>%

layer\_flatten(input\_shape = c(150, 150, 3)) %>%

layer\_dense(256, activation = "relu") %>%

layer\_dense(128, activation = "relu") %>%

layer\_dense(9, activation = 'softmax')

# be drop out labai overfittina (pasiekia beveik 100%)

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

# 614.39 sec elapsed | 54%

#-----------------------------------------------#

#-------------------- CNN 1 --------------------#

# Building a model

model <- keras\_model\_sequential()

model %>%

layer\_conv\_2d(32, c(3, 3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_dropout(0.3) %>%

layer\_conv\_2d(16, c(3,3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_dropout(0.3) %>%

layer\_flatten() %>%

layer\_dense(16, activation = "relu") %>%

layer\_dropout(0.5) %>%

layer\_dense(9, activation = 'softmax')

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

history$metrics$accuracy[length(history$metrics$accuracy)]

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

#---------------------------------------------#

#-------------------- CNN 2 --------------------#

# Building a model

model <- keras\_model\_sequential()

model %>%

layer\_conv\_2d(32, c(3, 3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_dropout(rate = 0.3) %>%

layer\_conv\_2d(64, c(3,3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_dropout(0.3) %>%

layer\_flatten() %>%

layer\_dense(128, activation = "relu") %>%

layer\_dropout(0.5) %>%

layer\_dense(64, activation = "relu") %>%

layer\_dropout(0.5) %>%

layer\_dense(9, activation = 'softmax')

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

history$metrics$accuracy[length(history$metrics$accuracy)]

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

#---------------------------------------------#

#-------------------- CNN 3 --------------------#

# Building a model

model <- keras\_model\_sequential()

model %>%

layer\_conv\_2d(64, c(3, 3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_conv\_2d(64, c(3, 3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_conv\_2d(32, c(3,3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_conv\_2d(32, c(3,3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_flatten() %>%

layer\_dense(512, activation = "relu") %>%

layer\_dense(128, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(128, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(9, activation = 'softmax')

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

history$metrics$accuracy[length(history$metrics$accuracy)]

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

#---------------------------------------------#

#-------------------- CNN 4 --------------------#

# Building a model

model <- keras\_model\_sequential()

model %>%

layer\_conv\_2d(64, c(3, 3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_conv\_2d(64, c(3, 3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_conv\_2d(128, c(3,3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_conv\_2d(128, c(3,3), activation = "relu", input\_shape = c(150, 150, 3), padding = "same") %>%

layer\_max\_pooling\_2d(2, 2) %>%

layer\_flatten() %>%

layer\_dense(1024, activation = "relu") %>%

layer\_dense(512, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(512, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(9, activation = 'softmax')

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

history$metrics$accuracy[length(history$metrics$accuracy)]

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

#---------------------------------------------#

#----------- Transfer learning 1---------------#

base\_model <- application\_resnet50(weights="imagenet", include\_top=FALSE, pooling="max", input\_shape = c(150, 150, 3))

freeze\_weights(base\_model)

predictions <- base\_model$output %>%

layer\_dense(512, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(256, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(128, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(9, activation = 'softmax')

model <- keras\_model(base\_model$input, predictions)

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

history$metrics$accuracy[length(history$metrics$accuracy)]

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

#---------------------------------------------#

#----------- Transfer learning 2---------------#

base\_model <- application\_resnet50(weights="imagenet", include\_top=FALSE, pooling="max", input\_shape = c(150, 150, 3))

freeze\_weights(base\_model)

predictions <- base\_model$output %>%

layer\_dense(1024, activation = "relu") %>%

layer\_dense(512, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(512, activation = "relu") %>%

layer\_dropout(0.2) %>%

layer\_dense(9, activation = 'softmax')

model <- keras\_model(base\_model$input, predictions)

summary(model)

model %>% compile(

loss = 'categorical\_crossentropy',

optimizer = optimizer\_adam(),#optimizer\_rmsprop(),

metrics = c('accuracy')

)

steps <- train\_gen$n / train\_gen$batch\_size

val\_steps <- val\_gen$n / val\_gen$batch\_size

tic()

history <- model %>%

fit(

train\_gen, epochs = 10,

steps\_per\_epoch = steps,

validation\_data = val\_gen,

validation\_steps = val\_steps

)

toc()

history$metrics$accuracy[length(history$metrics$accuracy)]

y\_true <- val\_gen$classes %>%

as.factor()

y\_pred <- model %>%

predict(val\_gen, steps = val\_steps + 1) %>%

max.col() %>%

-1 %>%

as.factor()

caret::confusionMatrix(y\_pred, y\_true)

#---------------------------------------------#