

**VILNIAUS UNIVERSITETAS**  
**MATEMATIKOS IR INFORMATIKOS FAKULTETAS**  
**STUDIJŲ PROGRAMA DUOMENU MOKSLAS 4 KURSAS**

**Vaizdų klasifikavimas naudojant konvoliucinius  
neuroninius tinklus**

**Darbo aprašas**

Autorius: Austėja Ona Plančiūnaitė  
VU el. p.: ona.plančiūnaitė@mif.stud.vu.lt

**Vilnius**  
**2024**

# Turinys

<b>1</b>	<b>Ivadas</b>	<b>2</b>
<b>2</b>	<b>Praktinė užduotis</b>	<b>3</b>
2.1	Tikslas . . . . .	3
2.2	Uždaviniai . . . . .	3
2.3	Duomenys . . . . .	3
<b>3</b>	<b>Skaičiavimo Resursai</b>	<b>4</b>
<b>4</b>	<b>Tyrimas</b>	<b>4</b>
4.1	Eksperimentų išvados . . . . .	7
4.2	Testavimo duomenų įrašų klasifikavimas . . . . .	7
<b>5</b>	<b>Hiperparametru tyrimas</b>	<b>9</b>
<b>6</b>	<b>Išvados</b>	<b>10</b>

# 1 Ivadas

## 2 Praktinė užduotis

### 2.1 Tikslas

Praktinės užduoties tikslas yra apmokyti konvoluciinių neuroninių tinklų vaizdams klasifikuoti, atliliki modelio architektūros ir hiperparametru įtakos klasifikavimo modelio tikslumui ir paklaidai tyrimą. Darbe turimas duomenų rinkinys padalijamas į tris dalis: mokymo, validavimo ir testavimo aibę. Apmokius modelius ir nustatius geriausią klasifikavimo modelio atvejį, patikrinami jo rezultatai testavimo duomenų rinkiniui. Programinis kodas parašytas naudojant „Python“ programinę kalbą, „Google Colab“ platformoje. Skaičiavimo resursams pasitelkta „Google Colab“ platformoje teikiama „T4 GPU“. Programinis kodas rašytas remiantis „TensorFlow“ dokumentacija<sup>1</sup>.

### 2.2 Uždaviniai

- Duomenų paruošimas, paskirstymas į mokymo, validavimo ir testavimo aibes.
- Konvoluciino neuroninio tinklo kūrimas vaizdams klasifikuoti.
- Sužinoti, kaip klasifikavimo tikslumas ir paklaida priklauso nuo tinklo architektūros ir hiperparametru reikšmių.

### 2.3 Duomenys

Praktiniam darbui atliliki naudojamas „Muffin vs Chihuahua Image Classification“ duomenų rinkinys, pateiktas Kaggle platformoje<sup>2</sup>. Šiame duomenų rinkinyje yra paveikslėliai, kuriuose pavaizduoti keksiukai ir šuniukai. Šiame duomenų rinkinyje yra 5917 paveikslėlių, kurie buvo surinkti iš „Google Images“. Vaizdai skirtomi į dvi klases: keksiukai (muffins) ir šuniukai (chihuahuas).

Duomenų rinkinys buvo padalintas į mokymo, validavimo ir testavimo aibes. Duomenys buvo padalinti į 80 % mokymo, 10 % validavimo ir 10 % testavimo rinkinių santykį. Galutiniai duomenų aibų dydžiai yra tokie:

- **Mokymo aibė:**
  - Keksiukai (muffin): 2174 paveikslėliai.
  - Šuniukai (chihuahua): 2559 paveikslėliai.
- **Validavimo aibė:**
  - Keksiukai (muffin): 272 paveikslėliai.
  - Šuniukai (chihuahua): 320 paveikslėliai.

---

<sup>1</sup><https://www.tensorflow.org/>

<sup>2</sup><https://www.kaggle.com/datasets/samuelcortinhas/muffin-vs-chihuahua-image-classification/data>

- Testavimo aibė:

- Keksiukai (muffin): 272 paveikslėliai.
- Šuniukai (chihuahua): 320 paveikslėliai.



1 pav.: Duomenų pavyzdys

Vaizdai buvo transformuoti į 128x128 pikselių dydžio matricas. Taip pat buvo atlikta jų normalizacija, transformuojant spalvų reikšmes į intervalą [0,1] naudojant „Rescaling“ sluoksnį. Šis paruošimas užtikrina vienodą vaizdų dydį ir intensyvumą, o tai leidžia neuroniniams tinklui greičiau ir tiksliau konverguoti.

### 3 Skaičiavimo Resursai

Skaičiavimams buvo naudojama Google Colab platforma, kuri suteikė NVIDIA Tesla T4 GPU. Programinei įrangai buvo naudotos TensorFlow (2.x versija) ir scikit-learn bibliotekos. TensorFlow buvo naudojama modelio kūrimui ir mokymui, o scikit-learn – duomenų padalijimui ir rezultatų vertinimui. Deja dėl ribotos nemokamos Google Colab GPU prieigos dalis modelio mokymo procesų vyko naudojant tik Google Colab CPU.

### 4 Tyrimas

Tyrimo metu buvo lyginamos trys skirtingos tinklų architektūros. Visuose tinkluose buvo naudojamas „rescaling“ sluoksnis, kuris paveikslėlio RGB reikšmes suveda į intervalą (0; 1). Mokymo metu visuose tinkluose naudotas „adam“ optimizavimo metodas, „ReLU“ aktyvacijos funkcija, ir paketų dydis 32. Architektūrų tinkamumas buvo vertinamas pagal klasifikavimo tikslumą ir paklaidą mokymo bei validavimo duomenų rinkiniuose.

Pirmasis modelis buvo sudarytas iš 9 sluoksninių, kuriuos sudaro 3 konvoliucijos sluoksniai (turintys po 16, 32, ir 64 filtrų), o po kiekvieno konvoliucijos sluoksnio pridėtas maksimaliojo sujungimo (*MaxPooling*) sluoksnis. Tankusis sluoksnis turėjo 128 neuronų.

Model 1 Summary:  
Model: "sequential"

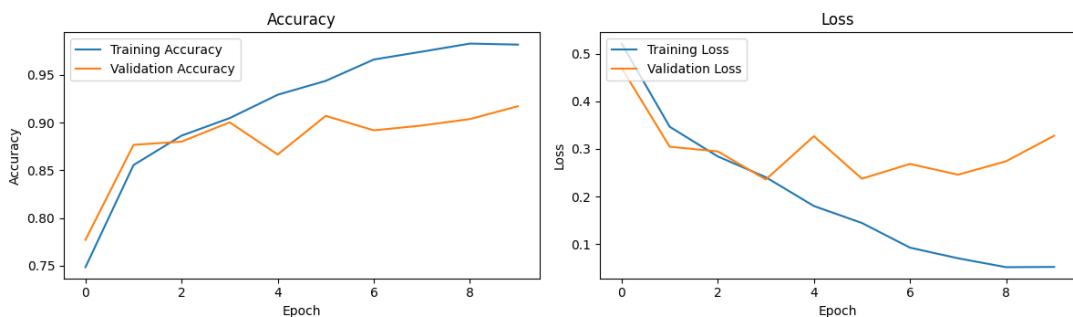
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 16)	448
max_pooling2d (MaxPooling2D)	(None, 64, 64, 16)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	4,640
max_pooling2d_1 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	18,496
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 128)	2,097,280
dense_1 (Dense)	(None, 2)	258

Total params: 6,363,368 (24.27 MB)  
Trainable params: 2,121,122 (8.09 MB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 4,242,246 (16.18 MB)

2 pav.: Pirmoji architektūra

Pirmojo modelio tikslumo (accuracy) ir nuostolio (loss) kreivės rodo, kad mokymo duomenų tikslumas nuolat augo ir pasiekė beveik 100 %, o mokymo paklaida tolygiai mažėjo per visas epochas. Validavimo duomenų tikslumas augo iki 4-osios epochos, tada pasyvyravo ir nuo 8 epochos tikslumas vėl pradėjo didėti, tačiau ir nuostolis pradėjo didėti kas rodo modelio persimokymą.

Model 1



3 pav.: Pirmosios architektūros rezultatai

Antrasis modelis buvo sudarytas iš 11 sluoksninių, išskaitant 6 konvoliucijos sluoksnius su (8, 16, 32, 64) filtrais ir maksimaliojo sujungimo sluoksniais po kiekvieno konvoliucinio sluoksnio. Tankusis sluoksnis turėjo 256 neuronų. Ši architektūra buvo gilesnė ir pasiekė didesnį tikslumą, tačiau kai kuriose epochose validavimo duomenų nuostoliai šiek tiek išaugo dėl modelio sudėtingumo.

Model 2 Summary:  
Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 128, 128, 8)	224
max_pooling2d_3 (MaxPooling2D)	(None, 64, 64, 8)	0
conv2d_4 (Conv2D)	(None, 64, 64, 16)	1,168
max_pooling2d_4 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_5 (Conv2D)	(None, 32, 32, 32)	4,640
max_pooling2d_5 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_6 (Conv2D)	(None, 16, 16, 64)	18,496
max_pooling2d_6 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_2 (Dense)	(None, 256)	1,048,832
dense_3 (Dense)	(None, 2)	514

Total params: 3,221,624 (12.29 MB)

Trainable params: 1,073,874 (4.10 MB)

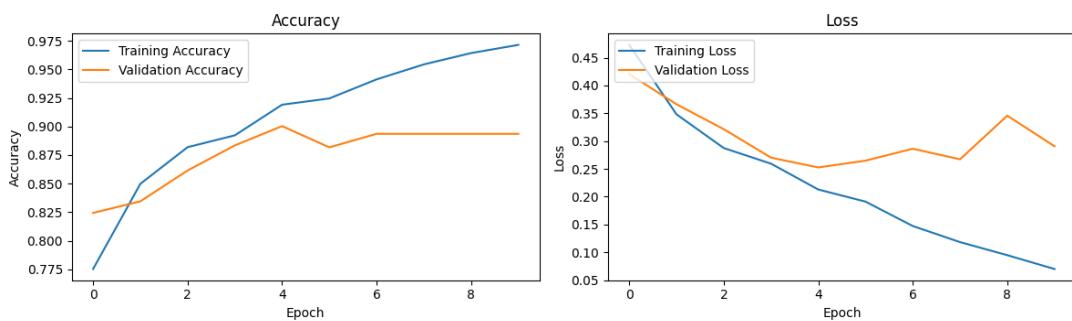
Non-trainable params: 0 (0.00 B)

Optimizer params: 2,147,750 (8.19 MB)

4 pav.: Antroji architektūra

Modelio tikslumo (accuracy) kreivė rodo stabilų augimą tiek mokymo, tiek validavimo duomenų rinkiniuose, pasiekusi apie 97% mokymo tikslumą ir 90% validavimo tikslumą. Nuostolio (loss) kreivė rodo, kad mokymo nuostolis nuolat mažėjo, tačiau validavimo nuostolis pradėjo svyruoti ir šiek tiek padidėjo po tam tikros epochos, kas rodo, jog modelis pradėjo persimokyti.

Model 2



5 pav.: Antrosios architektūros rezultatai

Trečiasis modelis buvo sudarytas iš 7 sluoksnių, įskaitant 2 konvoluciinius sluoksnius su 8 ir 16 filtrų bei maksimaliojo sujungimo sluoksnius po kiekvieno konvoluciinio sluoksnio. Po konvoluciinių sluoksnių ir maksimaliojo sujungimo sluoksnių sekė išlyginimo sluoksnis, kuris transformavo duomenis į 1D vektorių. Modelio tankusis sluoksnis turėjo 64 neuronų ir buvo atsakingas už klasifikaciją į dvi klasses. Išvesties sluoksnis turėjo 2 neuronus, atitinkančius dvi klasses.

Ši architektūra buvo mažiausiai sudėtinga ir greičiausia mokymo metu, tačiau dėl mažes-

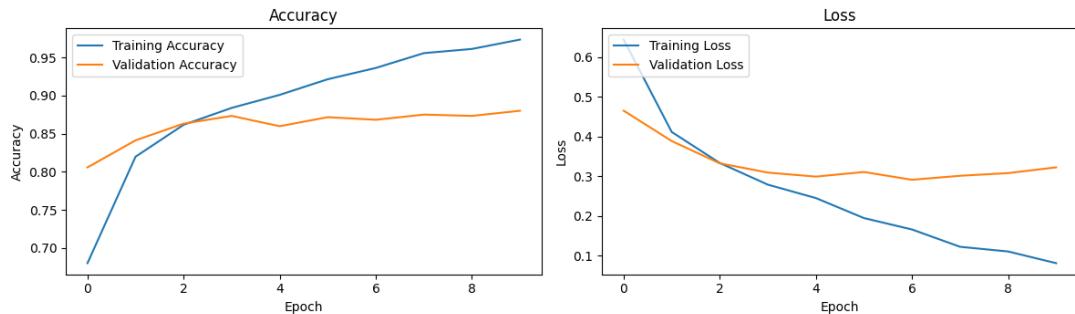
Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 128, 128, 8)	224
max_pooling2d_7 (MaxPooling2D)	(None, 64, 64, 8)	0
conv2d_8 (Conv2D)	(None, 64, 64, 16)	1,168
max_pooling2d_8 (MaxPooling2D)	(None, 32, 32, 16)	0
flatten_2 (Flatten)	(None, 16384)	0
dense_4 (Dense)	(None, 64)	1,048,640
dense_5 (Dense)	(None, 2)	130

Total params: 3,150,488 (12.02 MB)  
Trainable params: 1,050,162 (4.01 MB)  
Non-trainable params: 0 (0.00 B)  
Optimizer params: 2,100,326 (8.01 MB)

6 pav.: Trečioji architektūra

nio sluoksnį skaičiaus ir parametrų, klasifikavimo tikslumas buvo mažesnis nei pirmųjų dvieju modelių. Nors modelis greitai išmokė duomenis, jis pasiekė žemesnį validavimo tikslumą, palyginti su kitais modeliais, nes jo galimybės apdoroti sudėtingesnes duomenų savybes buvo ribotos.

Model 3



7 pav.: Trečiosios architektūros rezultatai

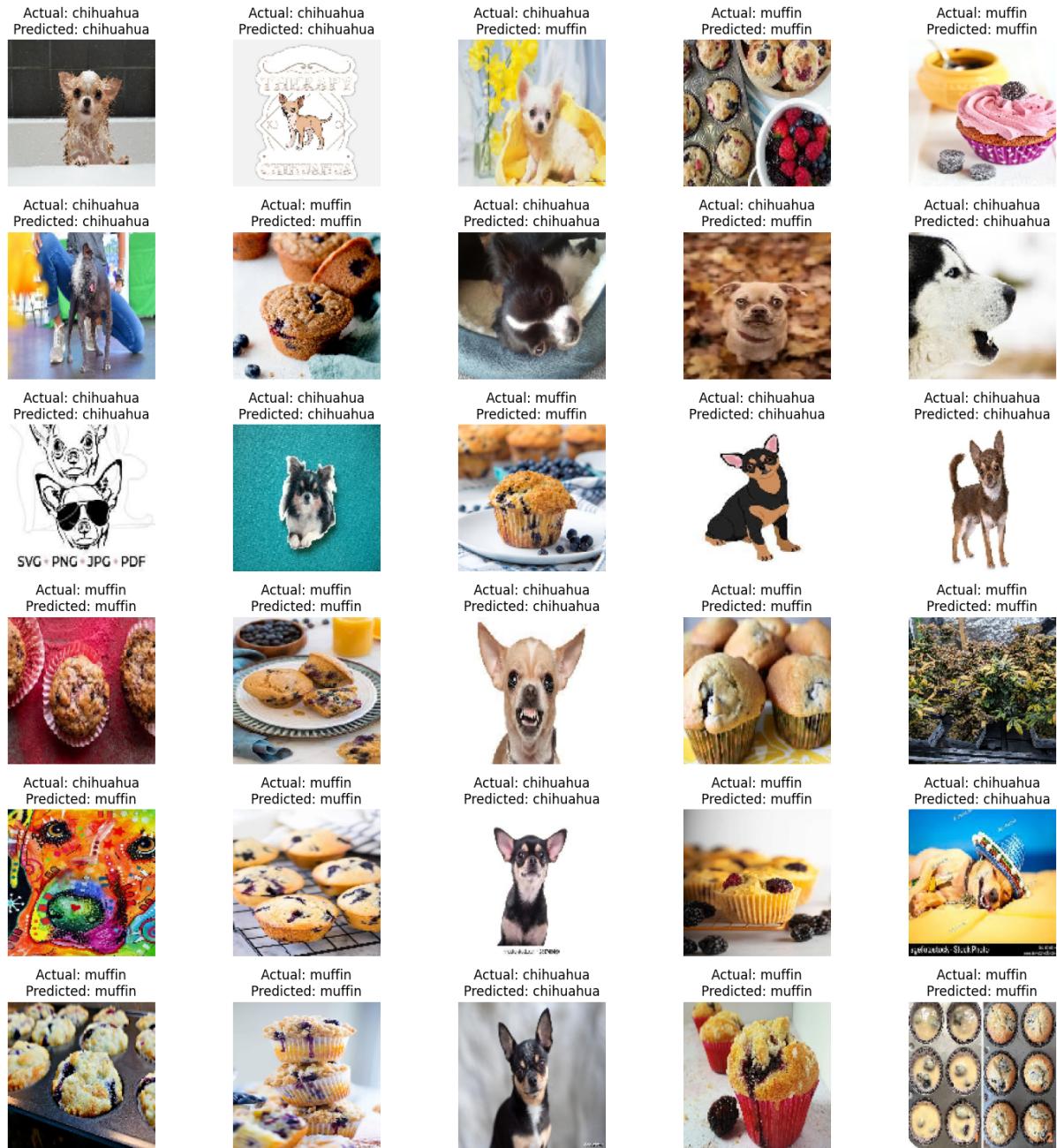
## 4.1 Eksperimentų išvados

Geriausias modelis buvo pirmasis, kuris pasiekė aukščiausią klasifikavimo tikslumą tiek mokymo, tiek validavimo duomenų rinkiniuose. Šis modelis, turintis 3 konvoluciinius sluoksnius sugebėjo geriausiai išmokti duomenų savybes ir pasiekti stabilų tikslumą, kuris buvo aukštessnis nei kitų modelių. Nors antrasis modelis buvo sudėtingesnis, jo tikslumas buvo mažesnis, o trečiasis modelis, nors greitas, pasiekė mažiausią tikslumą.

## 4.2 Testavimo duomenų įrašų klasifikavimas

Norint vizualiai įvertinti geriausio modelio gebėjimą klasifikuoti testavimo duomenų įrašus, buvo atrinkti 30 testavimo rinkinio pavyzdžių. Kiekviename pavyzdžyje pateikiama tikroji klasė ir modelio prognozė. Remiantis geriausiu modeliu, testavimo pavyzdžių klasifikavimo rezultatai

pateikiami 8 paveiksle. Matome, kad modelis suklydo tik vienoje nuotraukoje, kurioje yra neįprastų spalvų šuns vaizdas.



8 pav.: Testavimo duomenų aibės įrašų klasifikavimas

## 5 Hiperparametru tyrimas

Buvo tiriama, kaip klasifikavimo tikslumas ir paklaida gali pakisti pridedant kelis išmetimo (dropout) sluoksnius. Be to, buvo tikrinama, kaip šiuos rezultatus gali pakeisti paketų normalizavimas, naudojama aktyvacijos funkcija ir optimizavimo algoritmas.

Pasirinkta daryti tyrimą, kai dropout gali įgyti tikimybę 0 arba 0,25. Jei tikimybė yra 0, tai reiškia, kad išmetimo sluoksnis nėra pridedamas. Tikrinami rezultatai naudojant sigmoidinę ir „ReLU“ aktyvacijos funkcijas bei „adam“ ir stochastinio gradientinio nusileidimo optimizavimo metodus. Tam buvo sudaromi hiperparametru rinkiniai, iš viso turime 16 skirtinių parametru reikšmių kombinacijų.

```
{'dropOut': 0, 'batchNormalization': True, 'activationFunction': 'relu', 'optimizer': 'adam'}
{'dropOut': 0, 'batchNormalization': True, 'activationFunction': 'relu', 'optimizer': 'sgd'}
{'dropOut': 0, 'batchNormalization': True, 'activationFunction': 'sigmoid', 'optimizer': 'adam'}
{'dropOut': 0, 'batchNormalization': True, 'activationFunction': 'sigmoid', 'optimizer': 'sgd'}
{'dropOut': 0, 'batchNormalization': False, 'activationFunction': 'relu', 'optimizer': 'adam'}
{'dropOut': 0, 'batchNormalization': False, 'activationFunction': 'relu', 'optimizer': 'sgd'}
{'dropOut': 0, 'batchNormalization': False, 'activationFunction': 'sigmoid', 'optimizer': 'adam'}
{'dropOut': 0, 'batchNormalization': False, 'activationFunction': 'sigmoid', 'optimizer': 'sgd'}
{'dropOut': 0.25, 'batchNormalization': True, 'activationFunction': 'relu', 'optimizer': 'adam'}
{'dropOut': 0.25, 'batchNormalization': True, 'activationFunction': 'relu', 'optimizer': 'sgd'}
{'dropOut': 0.25, 'batchNormalization': True, 'activationFunction': 'sigmoid', 'optimizer': 'adam'}
{'dropOut': 0.25, 'batchNormalization': True, 'activationFunction': 'sigmoid', 'optimizer': 'sgd'}
{'dropOut': 0.25, 'batchNormalization': False, 'activationFunction': 'relu', 'optimizer': 'adam'}
{'dropOut': 0.25, 'batchNormalization': False, 'activationFunction': 'relu', 'optimizer': 'sgd'}
{'dropOut': 0.25, 'batchNormalization': False, 'activationFunction': 'sigmoid', 'optimizer': 'adam'}
{'dropOut': 0.25, 'batchNormalization': False, 'activationFunction': 'sigmoid', 'optimizer': 'sgd'}
```

9 pav.: Parametru reikšmių rinkiniai

Kiekvieno sudaryto modelio klasifikavimo tikslumo ir paklaidos mokymo bei validavimo duomenų rinkiniams rezultatai paskutinėje mokymo epochoje pateikiami 1 lentelėje. Prasciausiai rezultatai gaunami, kai nėra naudojamas paketų normalizavimas, pasirenkama sigmoidinė aktyvacijos funkcija ir stochastinio gradientinio nusileidimo metodas. Ši parametru variacija labai išskiria savo rezultatais, kadangi gaunamas vos 0,10 klasifikavimo tikslumas mokymo duomenų rinkiniui. Pridedant išmetimo sluoksnį su 0,25 išmetimo tikimybe, gauname didesnes paklaidų reikšmes. Be to, galima pastebėti, jog geresni klasifikavimo tikslumai gaunami tada, kai naudojamas paketų normalizavimas. Šiuo atveju gauname, kad „adam“ optimizavimo metodu būdingas didesnis klasifikavimo tikslumas ir mažesnė paklaida, nes yra net 2 atvejai, kada naudojant stochastinio gradientinio nusileidimo metodą gaunamas 0,10 tikslumas mokymo ir validavimo duomenims. Analogiskai gaume ir su pasirinkta aktyvacijos funkcija: naudojant sigmoidinę funkciją, dviejų tinklų klasifikavimo tikslumas buvo vos 0,10. Tad galime teigti, kad geresni rezultatai gaunami naudojant „ReLU“ aktyvacijos funkciją. Geriausius rezultatus gaume naudojant 1 parametru rinkinį, kai nėra pridedamas išmetimo sluoksnis, yra naudojamas paketų normalizavimas, „ReLU“ aktyvacijos funkcija ir „adam“ optimizavimo metodas.

1 lentelė: Hiperparametru tyrimo rezultatai

Mokymosi duomenų aibė		Validavimo duomenų aibė	
Tikslumas	Paklaida	Tikslumas	Paklaida
0,96	0,10	0,92	0,26
0,94	0,15	0,91	0,28
0,96	0,10	0,91	0,25
0,92	0,20	0,90	0,25
0,94	0,13	0,91	0,26
0,90	0,30	0,90	0,30
0,90	0,28	0,88	0,30
0,10	2,30	0,09	2,30
0,94	0,15	0,91	0,25
0,92	0,20	0,90	0,25
0,94	0,13	0,91	0,20
0,90	0,28	0,90	0,28
0,92	0,19	0,90	0,24
0,84	0,38	0,85	0,36
0,90	0,35	0,86	0,35
0,10	2,30	0,11	2,30

## 6 Išvados

Geriausi klasifikavimo tikslumo ir paklaidos rezultatai buvo pasiekti naudojant architektūrą su trimis konvoluciujos sluoksniais ir 128 neuronų tankiuoju sluoksniu. Pastebėta, kad paketų normalizavimo, „ReLU“ aktyvacijos funkcijos ir „adam“ optimizavimo metodo naudojimas užtikrino geresnius rezultatus nei kiti deriniai. Pridėjus išmetimo sluoksnį (*dropout*) su 0,25 tikimybe, paklaida išaugo, nors kai kuriais atvejais tai padėjo sumažinti persimokymo efektą.

Modelis dažniausiai klydo klasifikuodamas nuotraukas, kurios buvo stipriai modifikuotos, turėjo keistas spalvas ar formas. Be to, duomenų rinkinyje pasitaikė nuotraukų, kuriose vaizduojama nei šuniukas, nei keksiukas, tokie atvejai taip pat prisidėjo prie klaidingos klasifikacijos.

Tyrimas parodė, kad tam tikri hiperparametrai, tokie kaip sigmoidinė aktyvacijos funkcija, stochasticinio gradientinio nusileidimo metodas ir paketų normalizavimo nenaudojimas, gali duoti labai blogus rezultatus. Pavyzdžiui, kai šie parametrai buvo taikomi, klasifikavimo tikslumas siekė vos 0,10. Todėl, siekiant geriausių rezultatų, būtina tinkamai suderinti hiperparametrus ir tinklo architektūrą.

## Literatúra

- [1] [https://colab.research.google.com/github/rses-dl-course.github.io/blob/master/notebooks/python/L03\\_image\\_classification\\_with\\_cnn.ipynb](https://colab.research.google.com/github/rses-dl-course/rses-dl-course.github.io/blob/master/notebooks/python/L03_image_classification_with_cnn.ipynb) scrollTo = t9FDsUlxCaWW
- [2] [https://www.datacamp.com/tutorial/cnn-tensorflow-python?dc\\_referrer=https://www.datacamp.com](https://www.datacamp.com/tutorial/cnn-tensorflow-python?dc_referrer=https://www.datacamp.com)