Vilniaus Universitetas

Vaizdų klasifikavimas naudojant konvoliucinius neuroninius tinklus

Dirbtinio intelekto pagrindai

4 užduotis

Darbą atliko:

Dovydas Martinkus

Duomenų Mokslas 4 kursas 1gr.

Vilnius, 2022

**Turinys**

[1 Tikslas ir uždaviniai 3](#_Toc115375381)

[2 Duomenys 4](#_Toc115375382)

[3 Užduoties ataskaita 5](#_Toc115375383)

[4 Išvados 19](#_Toc115375384)

# Tikslas ir uždaviniai

Tikslas: Naudojant konvoliucinius neuroninius tinklus sudaryti modelį, gebantį atpažinti vaizdus.

Uždaviniai:

Analizuojamų duomenų paruošimas ir aprašymas.

Vaizdų atpažinimo modelio sudarymas parenkant tinkamą sluoksnių seką.

Modelių apmokymas ir (hiper)parametrų įtakos modelio pasiekiamiems rezultatams tyrimas.

Geriausio modelio parinkimas ir jo klasifikavimo kokybės detalus tyrimas (pagal klases).

# Duomenys

Užduotyje naudotas CIFAR10 duomenų rinkinys. Prieiga prie duomenų per internetą: <https://www.cs.toronto.edu/~kriz/cifar.html>.

Duomenų rinkinį sudaro 60000 nuotraukų, iš kurių kiekvienos dydis yra 32 x 32 pikselių.

Kiekvienas paveiksliukas priklauso tik vienai iš 10 galimų klasių: lėktuvas, automobilis, paukštis, katė, elnias, šuo, varlė, arklys, laivas, sunkvežimis.

Duomenys padalinti testavimo aibei priskiriant 50000 stebėjimų, testavimo aibei – likusius 10000.

# Užduoties ataskaita

Užduotis atlikta naudojant programavimo kalbą „Python“. Naudota „TensorFlow“ biblioteka, patogumo dėlei pasitelkiant „Keras“ API. Naudojant „Keras“, CIFAR10 duomenų rinkinys yra lengvai prieinamas pasitelkiant šios bibliotekos funkcijas. Nuskaičius duomenis pavaizduotas po vienas kiekvienai klasei priklausančio vaizdo pavyzdys.

**import** tensorflow **as** tf

**from** tensorflow.keras **import** layers

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

**import** pandas **as** pd

*# Duomenys iš https://www.cs.toronto.edu/~kriz/cifar.html*

*# Keras prie jų suteikia dar palengvintą prieigą*

(train\_images, train\_labels), (test\_images, test\_labels) **=** tf**.**keras**.**datasets**.**cifar10**.**load\_data()

*# Klasių pavadinimai*

classes\_labels **=** np**.**char**.**title(["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"])

*# Kiekvienos klasės pavyzdys*

fig, ax **=** plt**.**subplots(2,5,figsize**=**(15, 10))

**for** i **in** range(0,10):

indice **=** np**.**where(train\_labels **==** i)[0][0]

ax[i**//**5,i**%5**].imshow(train\_images[indice])

ax[i**//**5,i**%5**].set\_title(classes\_labels[i])



1 pav. Kiekvienos klasės paveikslėlio pavyzdys

Sudarant modelį panaudotas Rescaling sluoksnis, skirtas duomenyse esančias RBG vertes, svyruojančias nuo 0 iki 255, normalizuoti nuo 0 iki 1. Toliau iš eilės naudoti Conv2D, MaxPooling2D, Conv2D, MaxPooling2D, Conv2D, Flatten ir Dense sluoksniai. Sudaryto modelio schema pavaizduota žemiau:

*# modelio sudarymas*

**def** build\_model(activation):

inputs **=** tf**.**keras**.**Input(shape**=**(32,32,3,))

*# x = layers.RandomFlip("horizontal")(inputs),*

*# x = layers.RandomRotation(0.1),*

x **=** layers**.**Rescaling(scale**=**1.0 **/** 255)(inputs)

x **=** layers**.**Conv2D(32, (3, 3), activation**=**activation, input\_shape**=**(32, 32, 3))(x)

x **=** layers**.**MaxPooling2D((2, 2))(x)

x **=** layers**.**Conv2D(64, (3, 3), activation**=**activation)(x)

x **=** layers**.**MaxPooling2D((2, 2))(x)

x **=** layers**.**Conv2D(64, (3, 3), activation**=**activation)(x)

x **=** layers**.**Flatten()(x)

x **=** layers**.**Dense(64, activation**=**activation)(x)

output **=** layers**.**Dense(10)(x)

model **=** tf**.**keras**.**Model(inputs, output)

**return** model

Layer (type) Output Shape Param #

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

input\_1 (InputLayer) [(None, 32, 32, 3)] 0

rescaling (Rescaling) (None, 32, 32, 3) 0

conv2d (Conv2D) (None, 30, 30, 32) 896

max\_pooling2d (MaxPooling2D (None, 15, 15, 32) 0

)

conv2d\_1 (Conv2D) (None, 13, 13, 64) 18496

max\_pooling2d\_1 (MaxPooling (None, 6, 6, 64) 0

2D)

conv2d\_2 (Conv2D) (None, 4, 4, 64) 36928

flatten (Flatten) (None, 1024) 0

dense (Dense) (None, 64) 65600

dense\_1 (Dense) (None, 10) 650

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

Total params: 122,570

Trainable params: 122,570

Non-trainable params: 0

Modeliui mokyti naudota Sparse Categorical Crossentropy nuostolių (angl. loss) funkcija. Modelio rezultatų metrika pasirinktas bendras tikslumas (angl. accuracy). Modelio mokymui pasirinkta naudoti 10 epochų.

*# modelio mokymas*

**def** train\_model(train\_images,train\_labels,

activation **=** "relu",

batch\_size **=** 32,

optimizer**=**"adam",

loss**=**tf**.**keras**.**losses**.**SparseCategoricalCrossentropy(from\_logits**=True**),

metrics**=**["accuracy"],

attempt **=** 1):

model **=** build\_model(activation)

model**.**compile(optimizer**=**optimizer,

loss**=**loss,

metrics**=**metrics)

tensorboard\_callback **=** tf**.**keras**.**callbacks**.**TensorBoard('logs', histogram\_freq**=**1)

modelcheckpoint\_callback **=** tf**.**keras**.**callbacks**.**ModelCheckpoint(

filepath**=**"modeliai/" **+** "hyperparameter\_set\_" **+** str(attempt) **+** "/epoch\_{epoch}",

save\_freq**=**'epoch',

save\_best\_only**=True**)

history **=** model**.**fit(train\_images, train\_labels, epochs**=**10,

validation\_split**=**0.2,

verbose **=** 2, batch\_size **=** batch\_size,

callbacks**=**[tensorboard\_callback,modelcheckpoint\_callback])

**return** model, history

*# modelio įvertinimas*

**def** evaluate\_model(model,history,test\_images,test\_labels,**\*\***kwargs):

plt**.**plot(history**.**history['accuracy'], label**=**'Mokymo duomenys')

plt**.**plot(history**.**history['val\_accuracy'], label **=** 'Validacijos duomenys')

plt**.**xlabel('Epocha')

plt**.**ylabel('Tikslumas')

plt**.**legend(loc**=**'lower right')

plt**.**title(kwargs**.**\_\_str\_\_())

plt**.**show()

test\_loss, test\_accuracy **=** model**.**evaluate(test\_images, test\_labels, verbose**=**2)

print("Modelio tikslumas: ", round(test\_accuracy,2))

**return** test\_loss, test\_accuracy

*# mokymas ir įvertinimas*

**def** pipeline(train\_images,train\_labels,

test\_images,test\_labels,

activation**=**"relu",

batch\_size**=**32,

optimizer**=**"adam",

loss**=**tf**.**keras**.**losses**.**SparseCategoricalCrossentropy(from\_logits**=True**),

metrics**=**["accuracy"],

attempt **=** 1):

model, history **=** train\_model(train\_images,train\_labels,

activation,batch\_size,

optimizer,loss,metrics,attempt)

\_, test\_accuracy **=** evaluate\_model(model,history,test\_images,test\_labels,

activation**=**activation,

batch\_size**=**batch\_size,

optimizer**=**optimizer)

**return** model, history, test\_accuracy

Sudarytas parametrų tinklelis iš ReLu ir sigmoidinės aktyvacijos funkcijų, Adam ir SGD (stochastinio gradientinio nusileidimo) optimizavimo metodų, mokymo metu naudojamų paketų dydžių (angl. batch size) 32, 64 ir 128:

*# hiperparametrai, kurių įtaką tikrinsiu*

**from** sklearn.model\_selection **import** ParameterGrid

grid **=** ParameterGrid({"activation":["relu","sigmoid"],"batch\_size":[32,64,128],"optimizer":["sgd","adam"]})

results **=** pd**.**DataFrame(list(grid))

test\_accuracies **=** []

models **=** []

histories **=** []

Naudojant kiekvieną (hiper)parametrų rinkinį modelis apmokytas naudojant mokymo duomenų aibę. Mokymo metu 0.2 mokymo aibės naudota validacijai. Kiekvienos epochos metu gauti mokymo ir validacijos tikslumai pavaizduoti grafiškai. Iš grafikų galima matyti stipri mokymo epochos įtaka: daugeliu atveju didėjant epochai stipriai gerėja rezultatai vertinimui naudojant tiek mokymo, tiek validacijos aibes.

Kiekvienam modeliui taip pat išsaugotas koks tikslumas gautas naudojant apmokytą modelį testavimo duomenims.

Žemiau esantis kodas kartotas su i=0,1,...,11:

i **=** 0

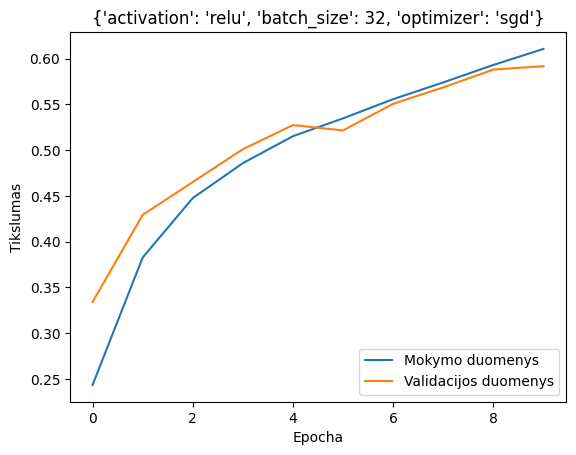
model, history, accuracy **=** pipeline(train\_images,train\_labels,test\_images,test\_labels,

**\*\***grid[i],attempt **=** i)

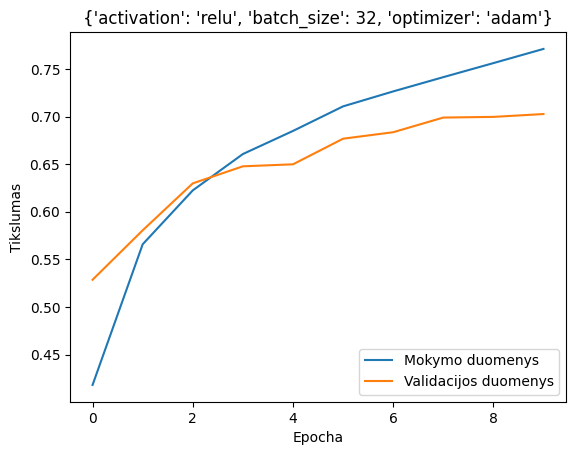
test\_accuracies**.**append(accuracy)

models**.**append(model)

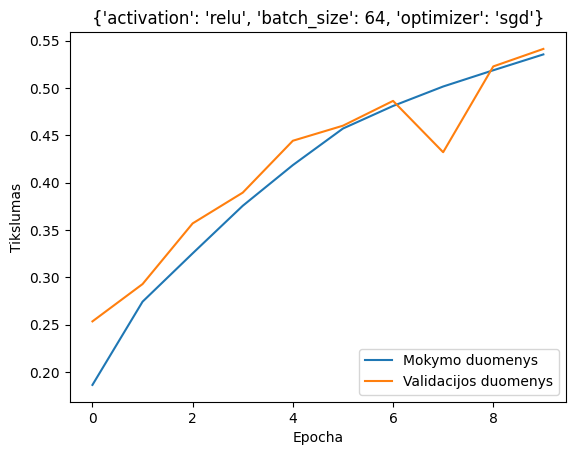
histories**.**append(history)



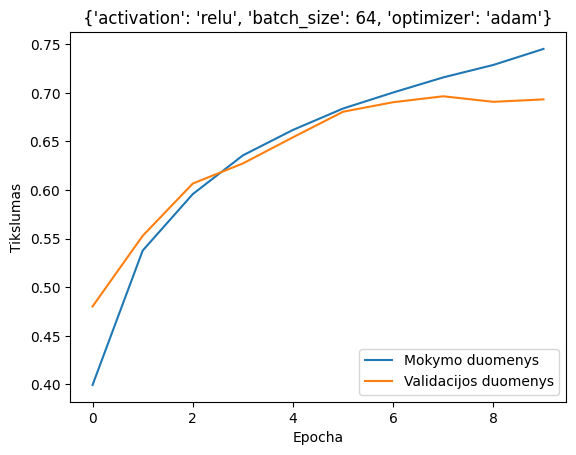
2 pav. Mokymo ir validacijos tikslumas pagal epochą



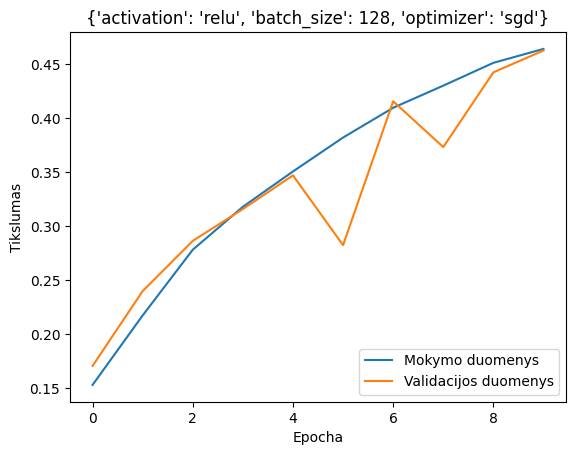
3 pav. Mokymo ir validacijos tikslumas pagal epochą



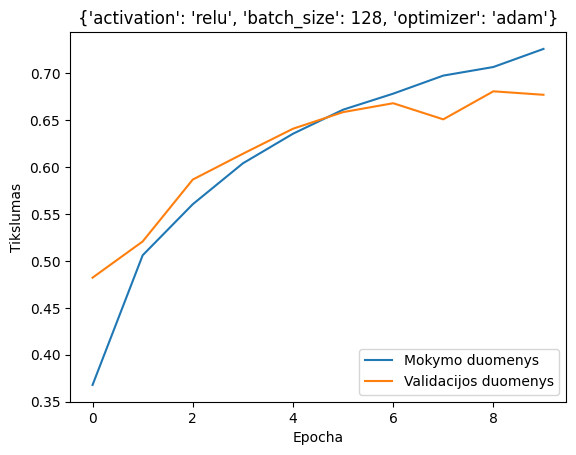
4 pav. Mokymo ir validacijos tikslumas pagal epochą



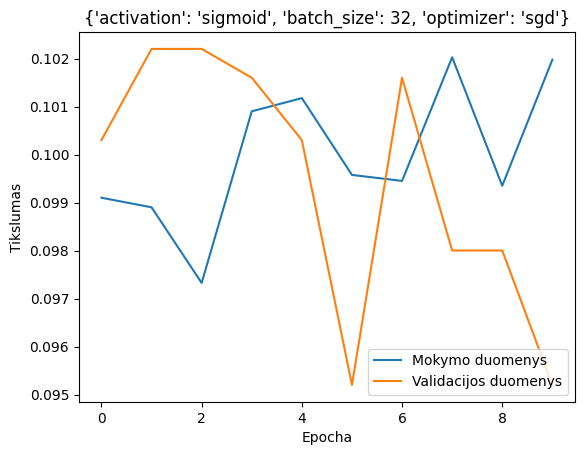
5 pav. Mokymo ir validacijos tikslumas pagal epochą



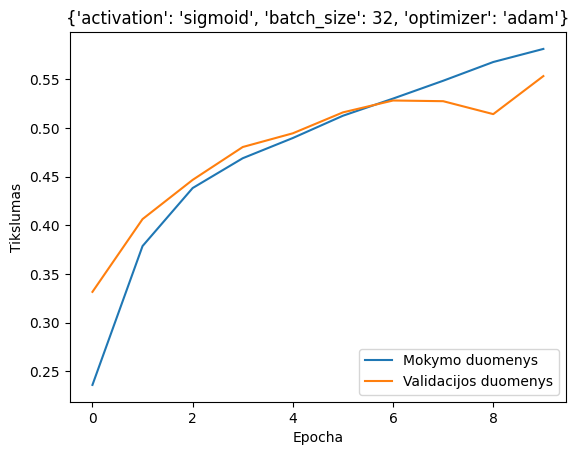
6 pav. Mokymo ir validacijos tikslumas pagal epochą



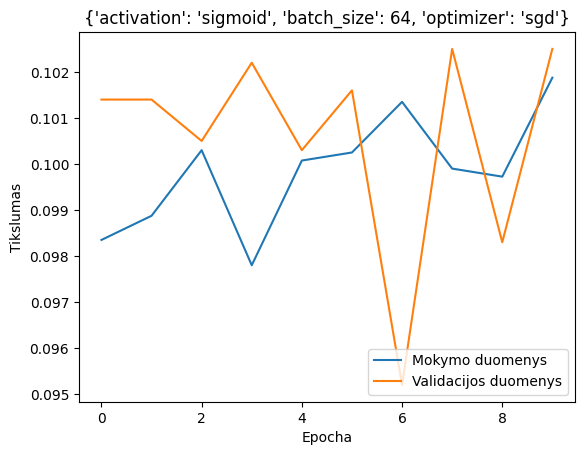
7 pav. Mokymo ir validacijos tikslumas pagal epochą



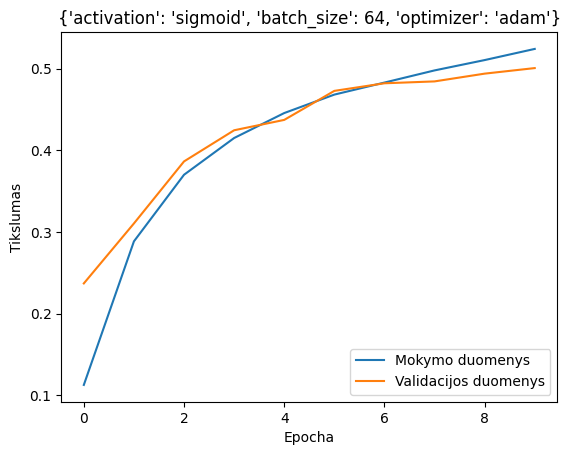
8 pav. Mokymo ir validacijos tikslumas pagal epochą



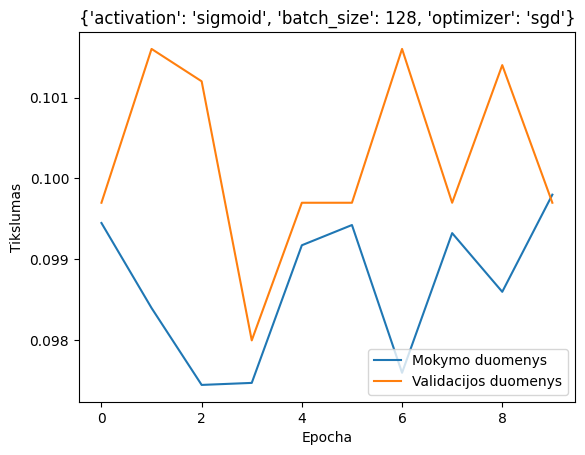
9 pav. Mokymo ir validacijos tikslumas pagal epochą



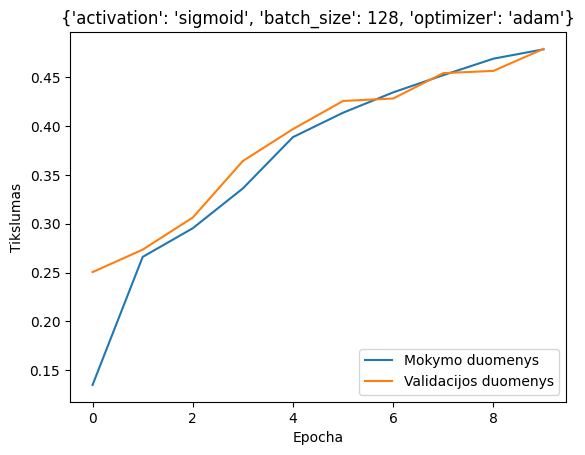
10 pav. Mokymo ir validacijos tikslumas pagal epochą



11 pav. Mokymo ir validacijos tikslumas pagal epochą



12 pav. Mokymo ir validacijos tikslumas pagal epochą



13 pav. Mokymo ir validacijos tikslumas pagal epochą

results["test\_accuracies"] **=** np**.**round\_(test\_accuracies,2)

results

Visų tikrintų modelių rezultatai pateikti lentelėje (1 lentelėje). Ja naudojantis galime matyti, kad geriausi rezultatai gauti naudojant modelį su ReLu aktyvacijos funkcija, imant 32 paketo dydį ir Adam optimizavimo metodą. Šis modelis bus naudojamas tolimesniame tyrime. Apskritai matome, kad naudojant ReLu aktyvacijos funkcija pasiekiami geresni rezultatai negu sigmoidinę. Panašus rezultatų pagerėjimas matomas ir naudojant Adam optimizavimo funkciją vietoje SGD. Tuo tarpu paketo dydžio įtaka nebuvo tokia ryški kaip kitų dviejų parametrų.

1 lentelė Testavimo aibės rezultatai kiekvienam hiperparametrų rinkiniui

|  |  |  |  |
| --- | --- | --- | --- |
| Aktyvacijos funkcija | Paketo dydis | Optimizavimo metodas | Testavimo aibės tikslumas |
| relu | 32 | sgd | 0.59 |
| relu | 32 | adam | 0.7 |
| relu | 64 | sgd | 0.54 |
| relu | 64 | adam | 0.68 |
| relu | 128 | sgd | 0.47 |
| relu | 128 | adam | 0.67 |
| sigmoid | 32 | sgd | 0.1 |
| sigmoid | 32 | adam | 0.55 |
| sigmoid | 64 | sgd | 0.1 |
| sigmoid | 64 | adam | 0.5 |
| sigmoid | 128 | sgd | 0.1 |
| sigmoid | 128 | adam | 0.48 |

model **=** models[np**.**argmax(test\_accuracies)]

*# Geriausias sudarytas modelis išsaugomas*

model**.**save("modeliai/geriausias")

best\_model **=** tf**.**keras**.**models**.**load\_model("modeliai/geriausias")

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix

y\_pred **=** best\_model**.**predict(test\_images)

y\_pred **=** [np**.**argmax(i) **for** i **in** y\_pred]

confusion\_matrix **=** confusion\_matrix(test\_labels, y\_pred)

print(confusion\_matrix)

test\_accuracy **=** accuracy\_score(test\_labels, y\_pred)

print("Geriausio modelio tikslumas: ", round(test\_accuracy,2))

Naudojant maišos matricą (angl. confusion matrix) pavaizduoti geriausio modelio rezultatai testavimo aibėje (2 lentelė):

2 lentelė Maišos matrica testavimo aibės duomenims naudojant geriausią modelį

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Prognozuota  Tikra | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | 746 | 20 | 55 | 16 | 23 | 4 | 11 | 9 | 84 | 32 |
| 1 | 29 | 826 | 7 | 8 | 4 | 1 | 7 | 3 | 35 | 80 |
| 2 | 78 | 6 | 580 | 66 | 122 | 54 | 50 | 16 | 17 | 11 |
| 3 | 21 | 10 | 82 | 543 | 90 | 125 | 49 | 39 | 24 | 17 |
| 4 | 21 | 8 | 56 | 73 | 693 | 38 | 42 | 56 | 10 | 3 |
| 5 | 19 | 7 | 68 | 209 | 52 | 569 | 15 | 36 | 15 | 10 |
| 6 | 11 | 4 | 56 | 92 | 43 | 28 | 740 | 7 | 12 | 7 |
| 7 | 21 | 4 | 52 | 49 | 78 | 56 | 8 | 713 | 7 | 12 |
| 8 | 71 | 35 | 13 | 17 | 9 | 4 | 3 | 2 | 822 | 24 |
| 9 | 43 | 86 | 6 | 17 | 4 | 8 | 9 | 15 | 26 | 786 |

Naudojant geriausią modelį ir testavimo aibės duomenis pateiktos rezultatų kokybės metrikos kiekvienai klasei atskirai (3 lentelė). Matome, kad modeliui blogiau sekėsi klasifikuoti kates ir paukščius. Pasinaudoję maišos matrica matome, kad katės dažnai sumaišytos su šunimis, tuo tarpu paukščiai – su elniais. Geriausiai modeliui sekėsi klasifikuoti automobilius ir sunkvežimius.

**from** sklearn.metrics **import** classification\_report

*# greitas būdas pažiūrėti kaip gerai klasifikuojama kiekviena klasė*

results\_frame **=** pd**.**DataFrame(classification\_report(test\_labels,y\_pred,output\_dict**=True**))

results\_frame**.**columns **=** np**.**concatenate((classes\_labels,results\_frame**.**columns[10:]**.**values))

results\_frame

3 lentelė Geriausio modelio rezultatai testavimo aibėje kiekvienai klasei atskirai

|  |  |  |  |
| --- | --- | --- | --- |
|  | Teigiamas prognostinis dydis | Jautrumas | F1 |
| Lėktuvas | 0.7 | 0.75 | 0.72 |
| Automobile | 0.82 | 0.83 | 0.82 |
| Bird | 0.59 | 0.58 | 0.59 |
| Cat | 0.5 | 0.54 | 0.52 |
| Deer | 0.62 | 0.69 | 0.65 |
| Dog | 0.64 | 0.57 | 0.6 |
| Frog | 0.79 | 0.74 | 0.77 |
| Horse | 0.8 | 0.71 | 0.75 |
| Ship | 0.78 | 0.82 | 0.8 |
| Truck | 0.8 | 0.79 | 0.79 |

Pasirinkta pavaizduoti pavyzdžius vaizdų, kuriuos geriausias modelis klasifikavo klaidingai (14 pav.):

*# Kelių neteisingų prognozių pavyzdžiai*

wrong\_indices **=** np**.**random**.**choice(np**.**where(y\_pred **!=** test\_labels[:,0])[0],10)

fig, ax **=** plt**.**subplots(2,5,figsize**=**(15, 10))

**for** i **in** range(0,10):

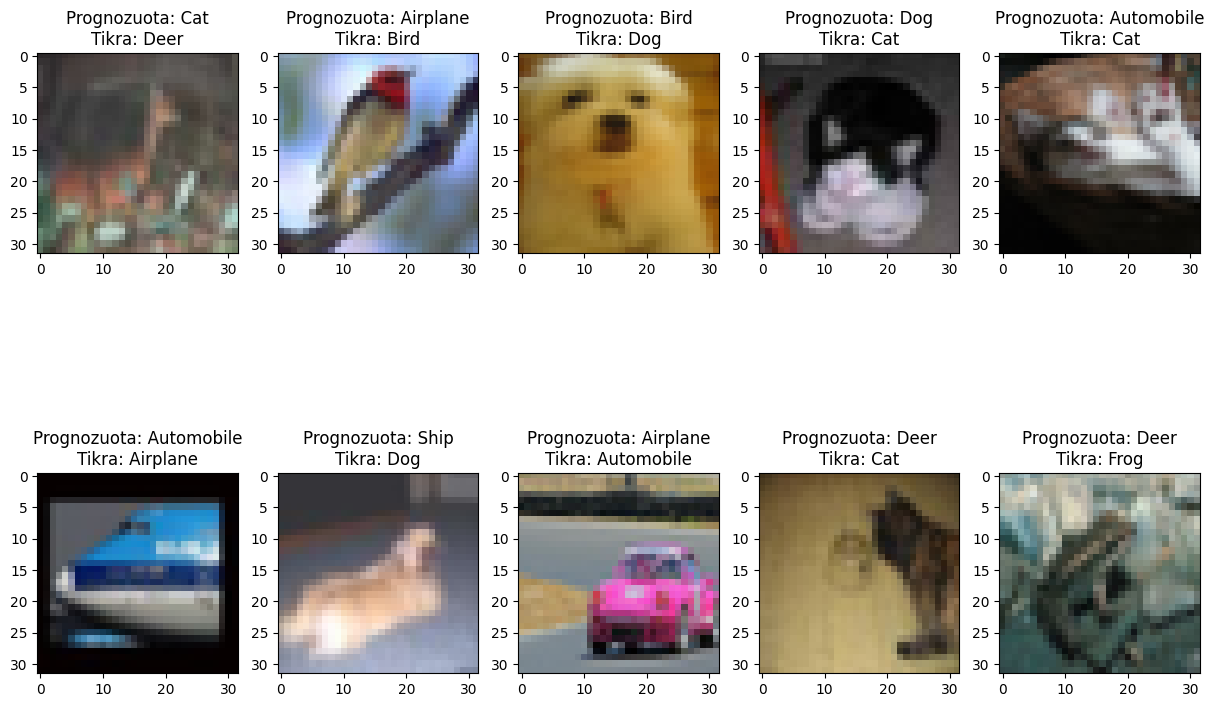
indice **=** wrong\_indices[i]

ax[i**//**5,i**%5**].imshow(test\_images[indice])

ax[i**//**5,i**%5**].set\_title("Prognozuota: " + classes\_labels[y\_pred[indice]] +

"\n" **+**

"Tikra: " **+** classes\_labels[test\_labels[indice][0]])



14 pav. Neteisingai klasifikuotų stebėjimų pavyzdžiai

Lentelėje pavaizduotas dar keletas modelio prognozuotų ir tikrų klasių pavyzdžių (4 lentelė):

print("Keli atsitiktiniai stebėjimai iš testavimo aibės:")

tikra **=** []

prognozuota **=** []

teisinga **=** []

indices **=** np**.**random**.**choice(np**.**where(np**.**array(y\_pred) **==** np**.**array(y\_pred))[0],30)

**for** i **in** indices:

teisinga**.**append(y\_pred[i] **==** test\_labels[i][0]),

prognozuota**.**append(classes\_labels[y\_pred[i]]),

tikra**.**append(classes\_labels[test\_labels[i][0]])

pd**.**DataFrame({"Teisinga":teisinga,"Tikra":tikra,"Prognozuota":prognoz

4 lentelė Klasifikavimo testavimo aibėje naudojant geriausią modelį pavyzdžiai

|  |  |  |
| --- | --- | --- |
| Teisinga | Tikra | Prognozuota |
| True | Dog | Dog |
| True | Horse | Horse |
| True | Cat | Cat |
| True | Ship | Ship |
| False | Cat | Bird |
| True | Ship | Ship |
| True | Truck | Truck |
| True | Deer | Deer |
| True | Truck | Truck |
| True | Ship | Ship |
| False | Cat | Dog |
| True | Deer | Deer |
| True | Ship | Ship |
| False | Deer | Bird |
| True | Dog | Dog |
| False | Deer | Frog |
| True | Deer | Deer |
| False | Ship | Automobile |
| True | Airplane | Airplane |
| False | Automobile | Truck |
| True | Dog | Dog |
| True | Horse | Horse |
| True | Frog | Frog |
| True | Ship | Ship |
| True | Dog | Dog |
| False | Dog | Cat |
| True | Truck | Truck |
| True | Deer | Deer |
| True | Horse | Horse |
| True | Bird | Bird |

# Išvados

Mokymo metu pastebėta stipri mokymo epochos įtaka – didėjant epochai daugeliu atveju gautas ryškus klasifikavimo rezultatų pagerėjimas tiek mokymo, tiek validacijos aibėse.

Didžiausia įtaką modelio pasiekiamiems rezultatams turėjo aktyvacijos funkcija: Naudojant ReLu gauti stipriai geresni rezultatai negu pasitelkiant sigmoidinę aktyvacijos funkciją. Taip pat svarbi naudojamo optimizacijos metodo įtaka: naudojant Adam gauti geresni rezultatai negu su SGD. Paketo dydžio reikšmės įtaka gautiems rezultatams daug mažesnė.

Geriausi rezultatai gauti naudojant modelį su ReLu aktyvacijos funkcija, imant 32 paketo dydį ir Adam optimizavimo metodą.

Geriausias modelis prasčiausiai atpažino kates nuo šunų, taip pat ir paukščius nuo elnių. Geriausiai modeliui sekėsi atpažinti automobilius ir sunkvežimius.

Bendri geriausio modelio pasiekti rezultatai nėra labai geri (didžiausias tikslumas testavimo aibėje – 0.7). Galima šio rezultato priežastis – stipriai sumažinta naudotų paveikslėlių raiška.