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

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).

2 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.

3 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 = []
```

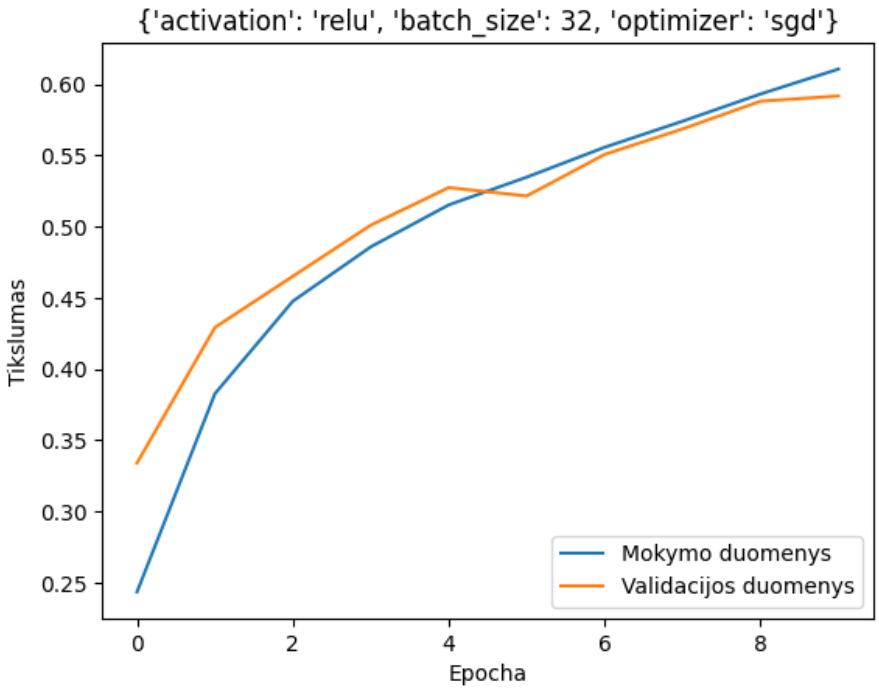
```
models = []
histories = []
```

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.

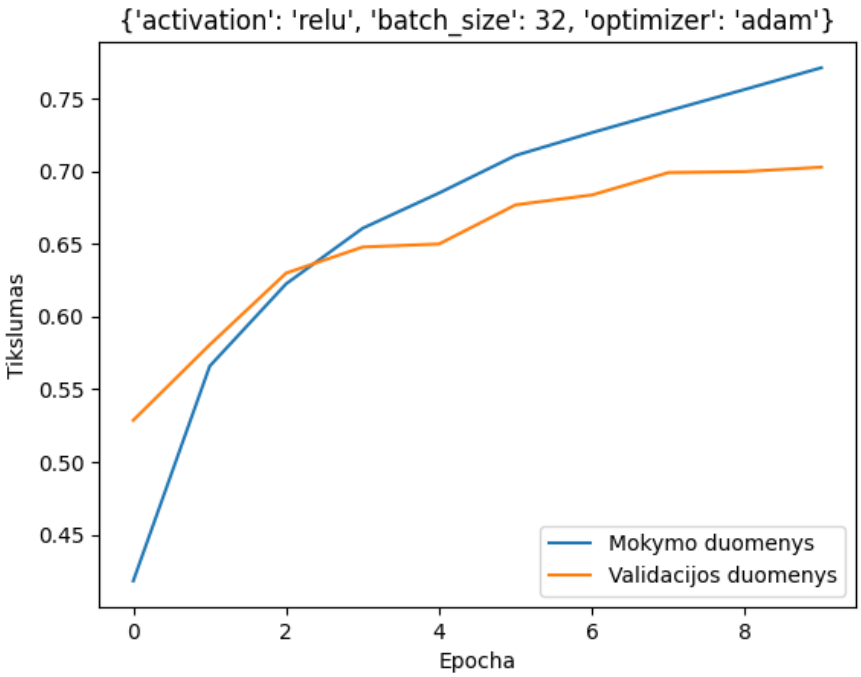
Kiekvienam modeliui taip pat išsaugotas koks tikslumas gautas naudojant apmokytą modelį testavimo duomenims.

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

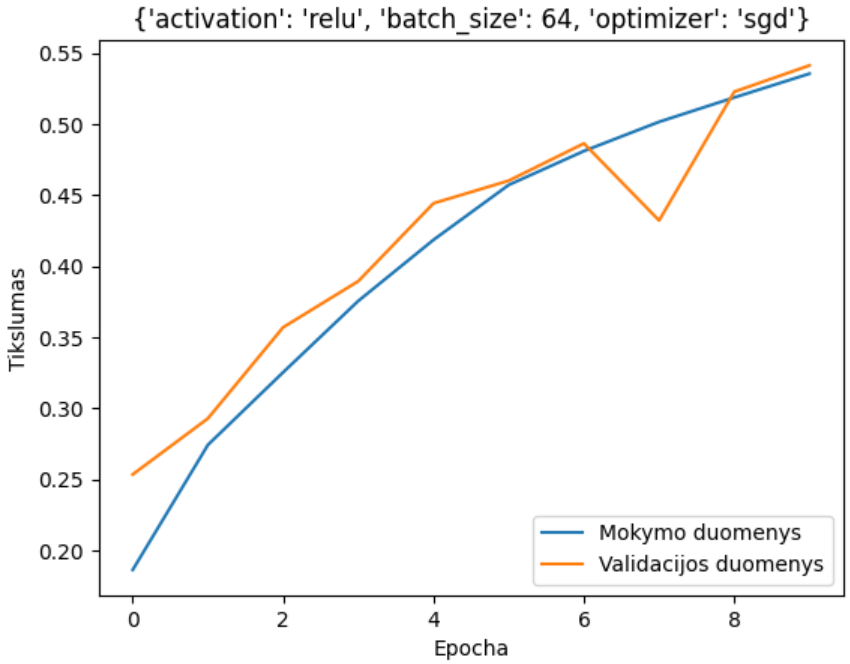
```
i = 0

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)
```



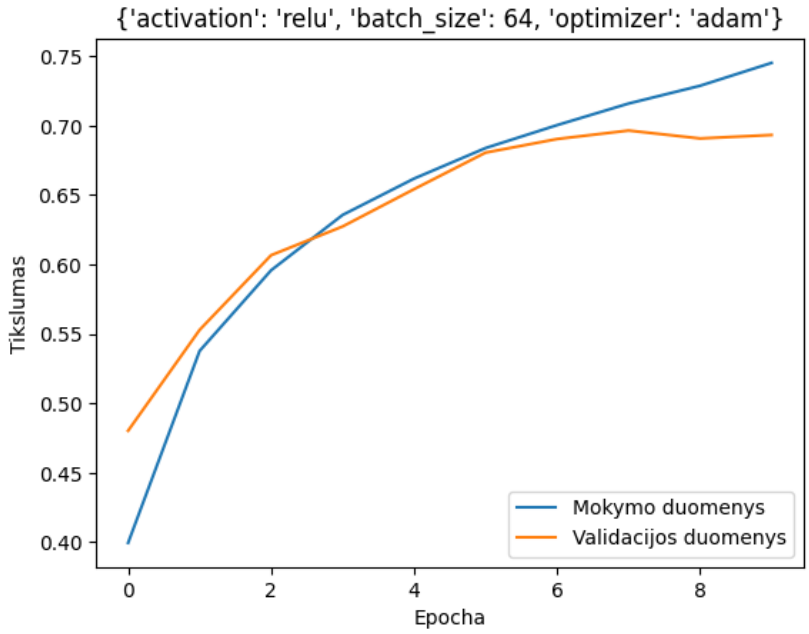
2 pav. Mokymo ir validacijos tikslumas pagal epochą



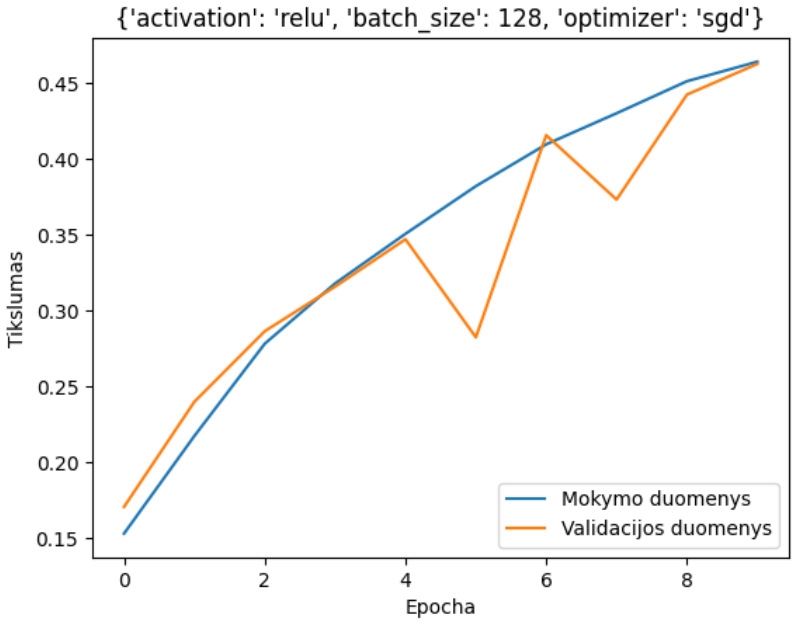
3 pav. Mokymo ir validacijos tikslumas pagal epochą



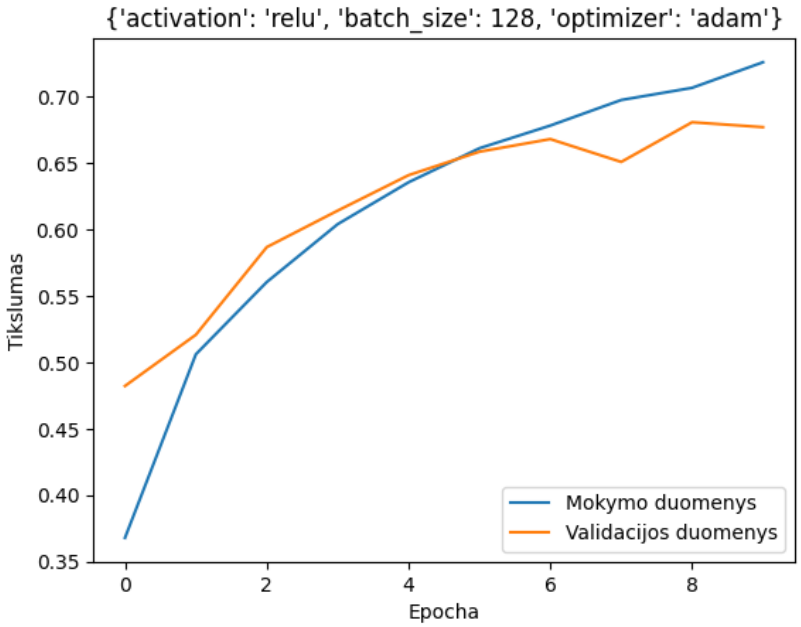
4 pav. Mokymo ir validacijos tikslumas pagal epochą



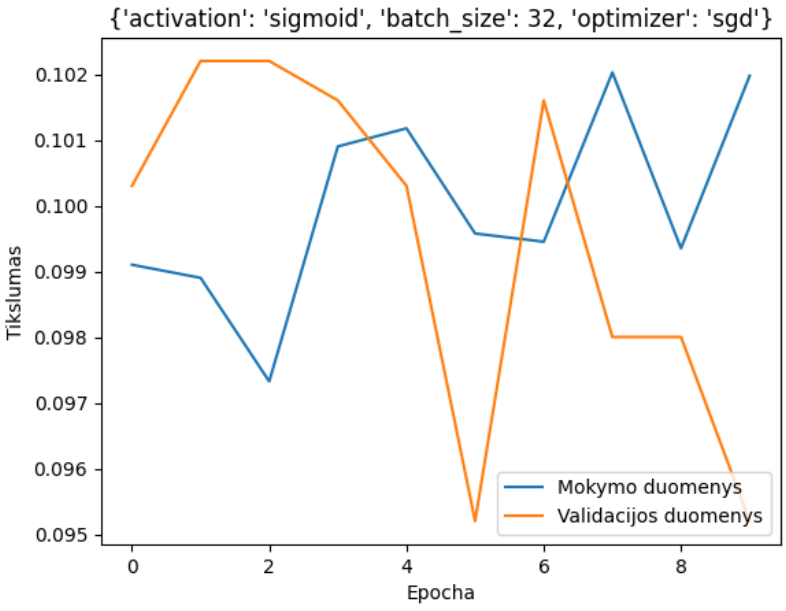
5 pav. Mokymo ir validacijos tikslumas pagal epochą



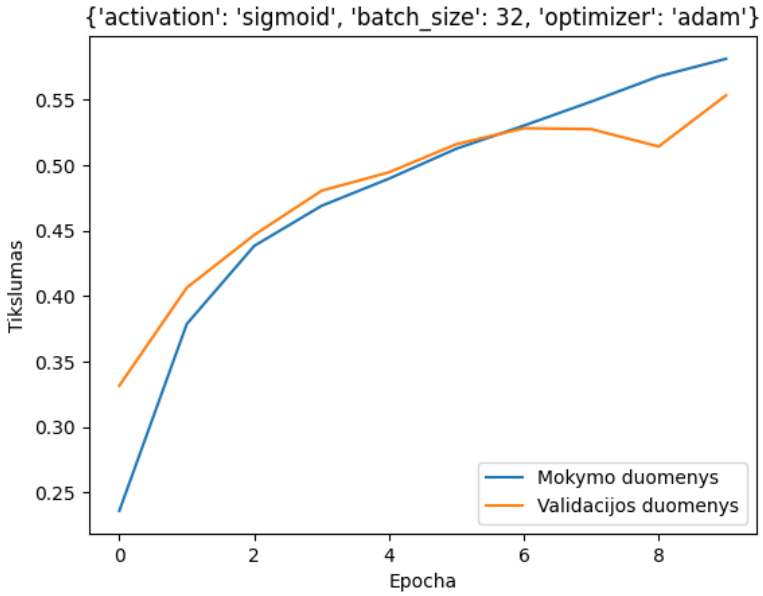
6 pav. Mokymo ir validacijos tikslumas pagal epochą



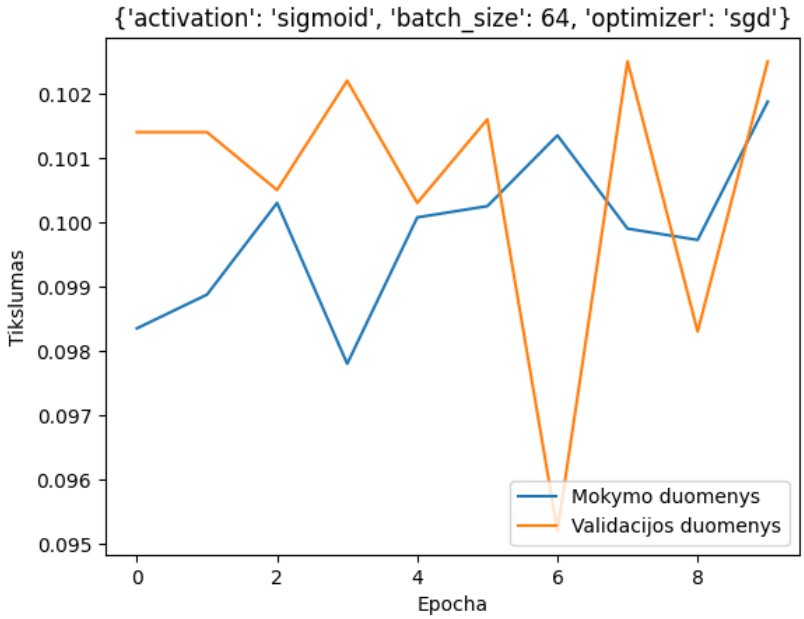
7 pav. Mokymo ir validacijos tikslumas pagal epochą



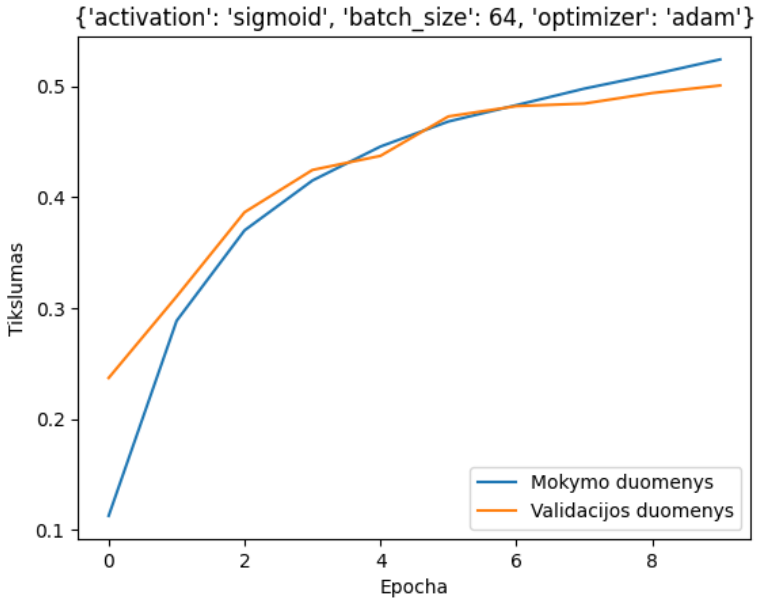
8 pav. Mokymo ir validacijos tikslumas pagal epochą



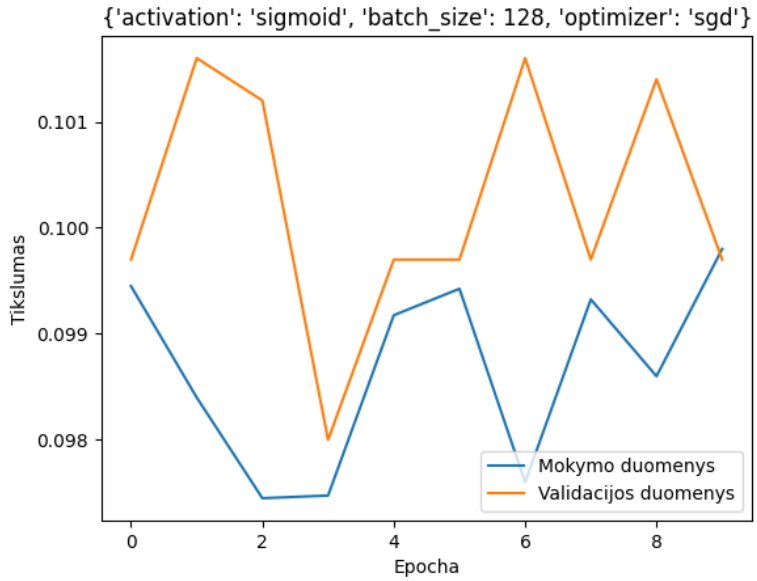
9 pav. Mokymo ir validacijos tikslumas pagal epochą



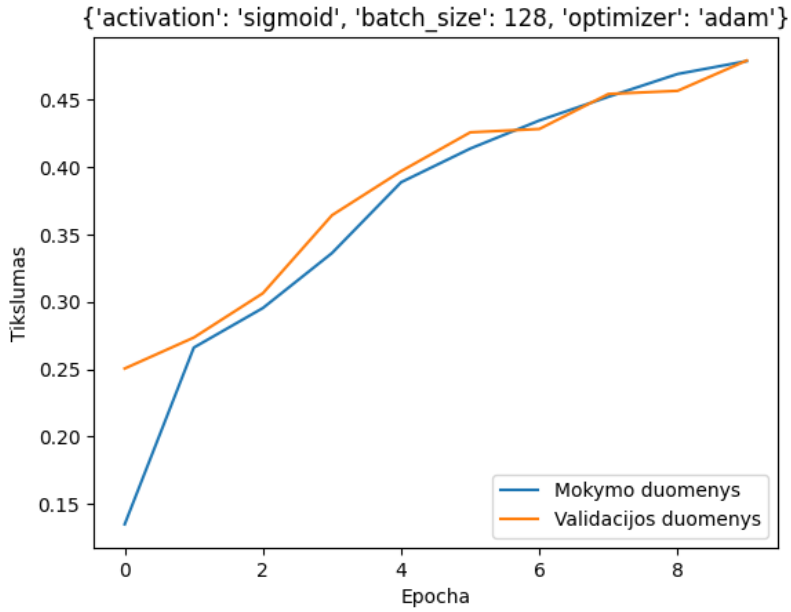
10 pav. Mokymo ir validacijos tikslumas pagal epochą



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
```

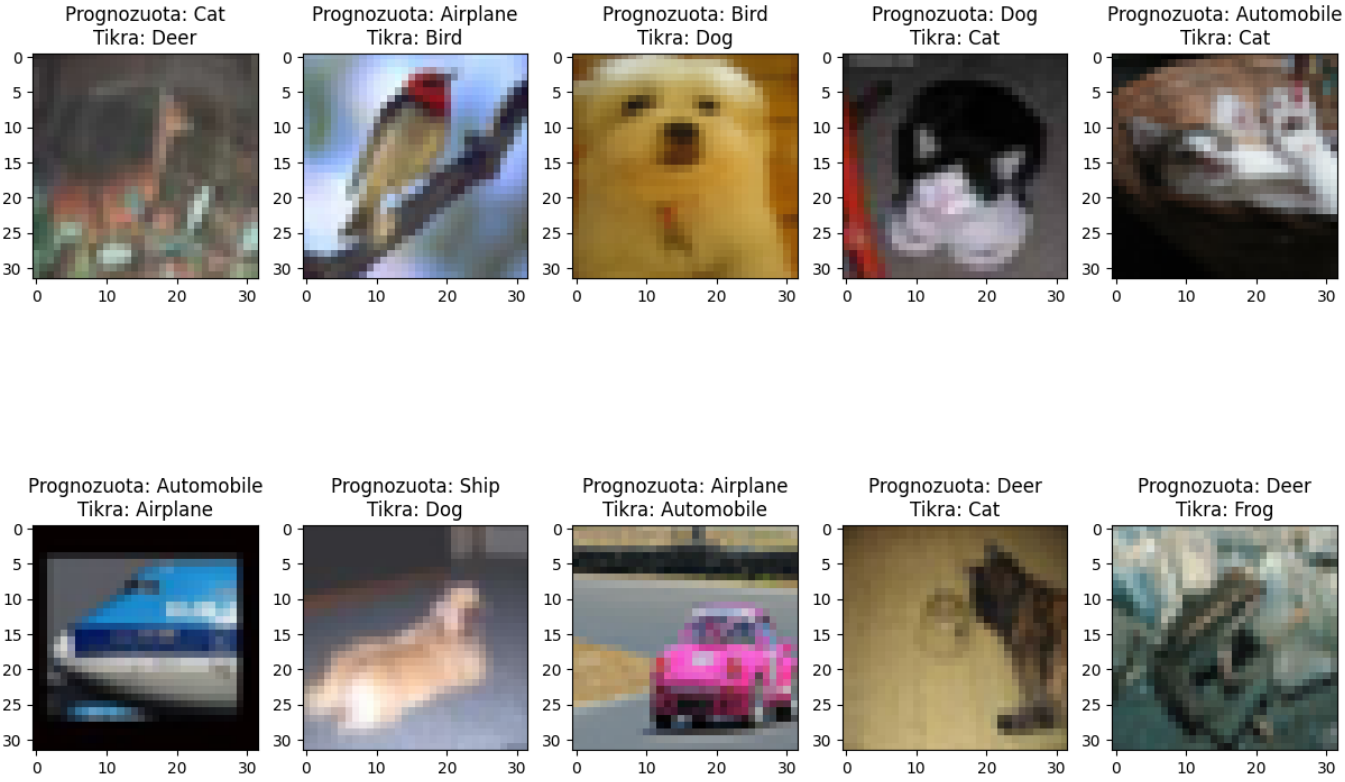
3 lentelė Geriausio modelio rezultatai testavimo aibėje kiekvienai klasei atskirai

	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

4 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.