VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Dirbtinio Intelekto 4 laboratorinis darbas

Darbą atliko: VU MIF ISI 3 kurso, 2 grupės, 1 pogrupio studentas Ignas Biekša

Vilnius, 2022

Turinys

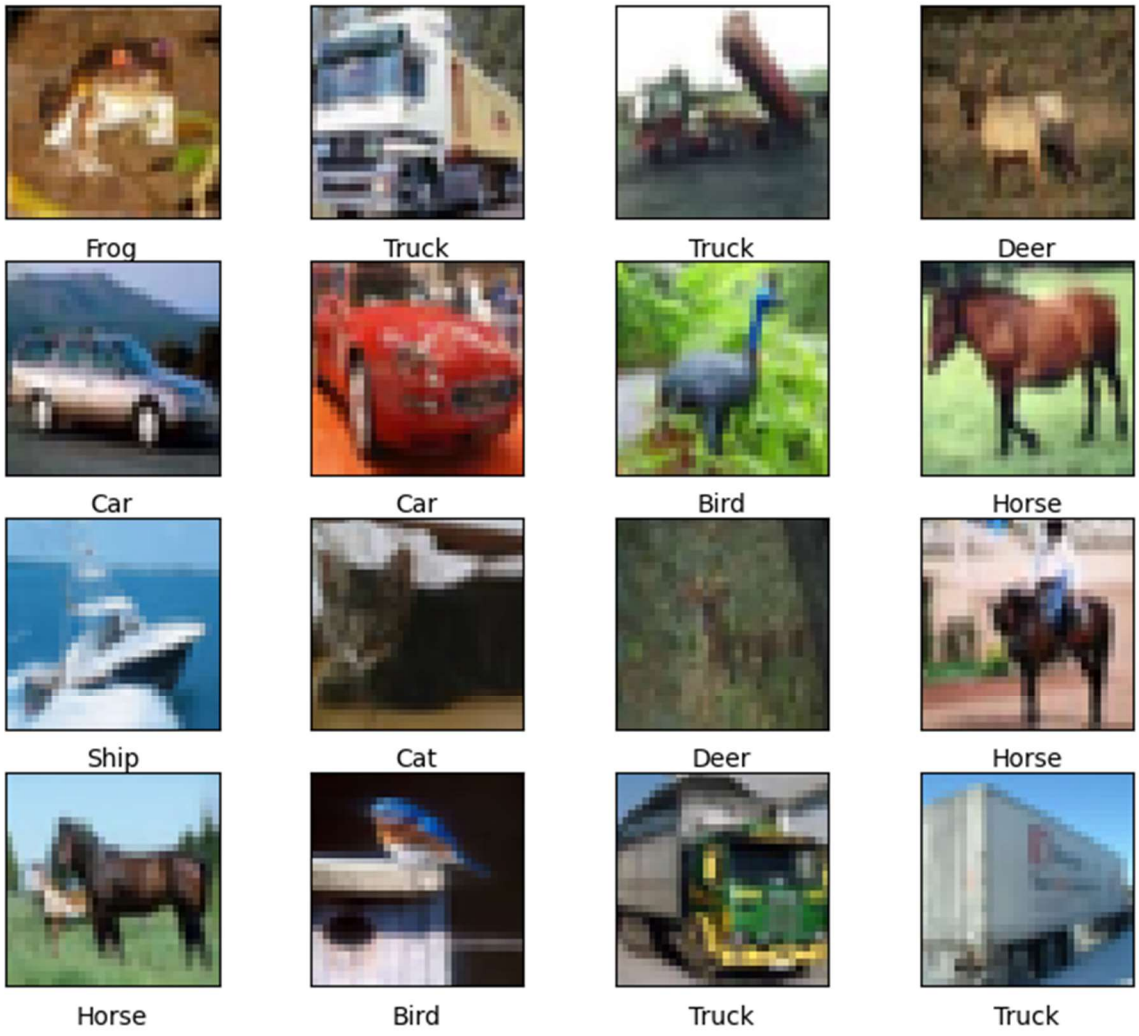
Dirbtinio Intelekto 4 laboratorinis darbas 1
Darbo tikslas 3
Duomenų aprašymas 3
Konvoliucinio neuroninio tinklo architektūros aprašymas, naudojami resursai 4
Tyrimo rezultatai: 5
Klasifikavimo matrica (confusion matrix) 17
30 Atsitiktinai pasirinktų įrašų klasifikavimo rezultatai 17
Išvados 18
Programos kodas: 18

Darbo tikslas

Apmokyti konvoliucinį neuroninį tinklą vaizdams klasifikuoti, atlikti tyrimą.

Duomenų aprašymas

Buvo naudojami CIFAR10 rinkinio vaizdai iš https://www.cs.toronto.edu/~kriz/cifar.html. Rinkinys susidaro iš 60000 32x32 formato nuotraukų. Rinkinį sudaro 10 klasių, kiekvienai po 6000 nuotraukų. Klasių pavadinimai: 'Plane', 'Car', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck'. Mokymo ir testavimo duomenys padalinti santykiu 1:5.



Pav. 1 Rinkinyje esančių vaizdų pavyzdys

Konvoliucinio neuroninio tinklo architektūros aprašymas, naudojami resursai

Modelis kuriamas naudojant TensorFlow biblioteką. Modelis sudarytas sekos formatu, jį sudaro sluoksniai: rescaling – kuris sunormalizuoja duomenų reikšmes į intervalą [0; 1], conv2d – konvoliucinio tinklo dvidimensinis sluoksnis, max_pooling2d – dvidimensinė skaičiavimo operacija, kuri sukompresuoja rezultato formatą. Flatten – paverčia rezultatus į vienos dimensijos masyvą. Dense – NT sluoksnis kuris pagal aktyvacijos funkciją sumažina rezultatų skaičių.

Hiperparametrai, kurie buvo keičiami norint rasti tiksliausią modelį per 10 epochų yra: aktyvacija – tiesinė (relu) ir sigmoidinė (sigmoid), paketo dydis – 32; 64; 128, optimizavimo algoritmas – gradiento nusileidimo su pagreičiu (sgd), gradiento nusileidimo su adaptyviu nuspėjimu (adam).

Naudojami resursai: CPU – Ryzen 5 3600, atmintis – 4 GB.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 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
```

Pav. 2 Konvoliucinio neuroninio tinklo modelio architektūra

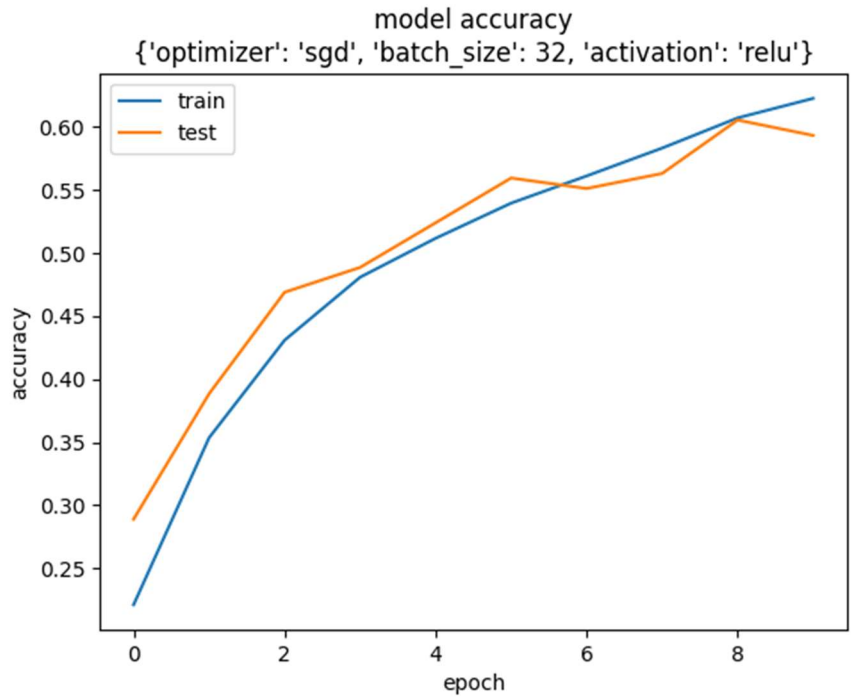
Tyrimo rezultatai:

Ištestuoti modeliai su skirtingais hiperparametrais, nustatyta, kad tiksliausia kombinacija yra 3 – su tiesine aktyvacija, 64 paketo dydžiu ir nuspėjančio gradiento nusileidimo optimizavimu.

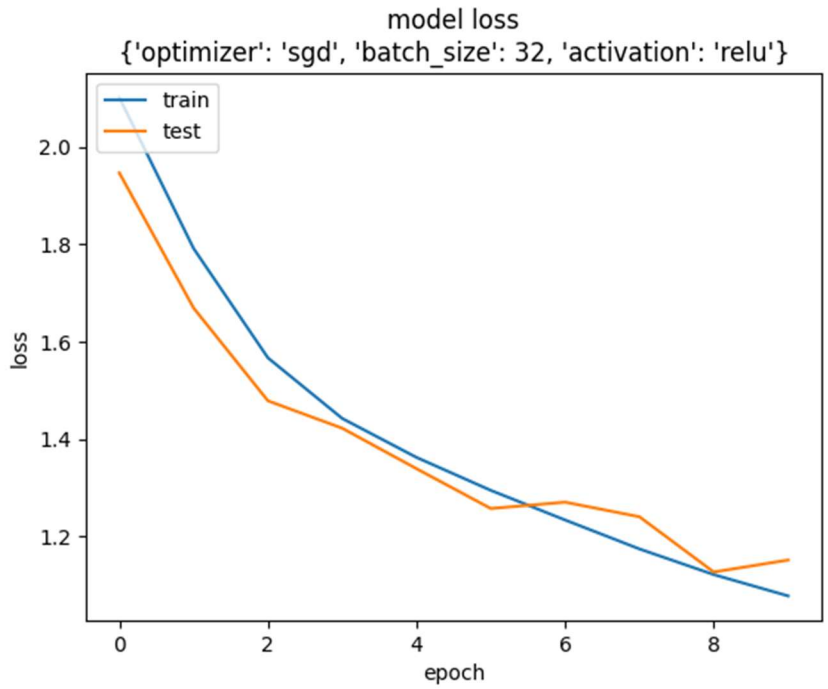
0	relu	32	sgd	0.5973
1	relu	32	adam	0.6873
2	relu	64	sgd	0.5292
3	relu	64	adam	0.6988
4	relu	128	sgd	0.4509
5	relu	128	adam	0.6833
6	sigmoid	32	sgd	0.1000
7	sigmoid	32	adam	0.5497
8	sigmoid	64	sgd	0.1000
9	sigmoid	64	adam	0.5129
10	sigmoid	128	sgd	0.1000
11	sigmoid	128	adam	0.4819

Pav. 3 12 hiperparametrų kombinacijų. Paskutiniame stulpelyje - jų tikslumas.

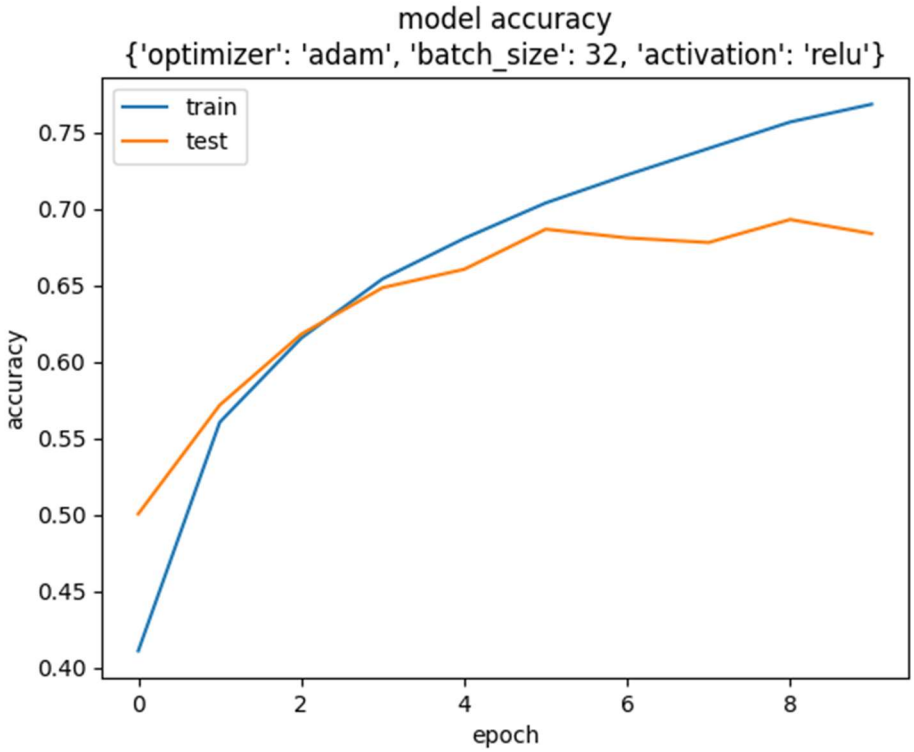
Žemiau yra pateikti grafikai su modelio tikslumo ir paklaidos pokyčiu kiekvienos epochos metu. Galime matyti, kad modelis nepersimoko.



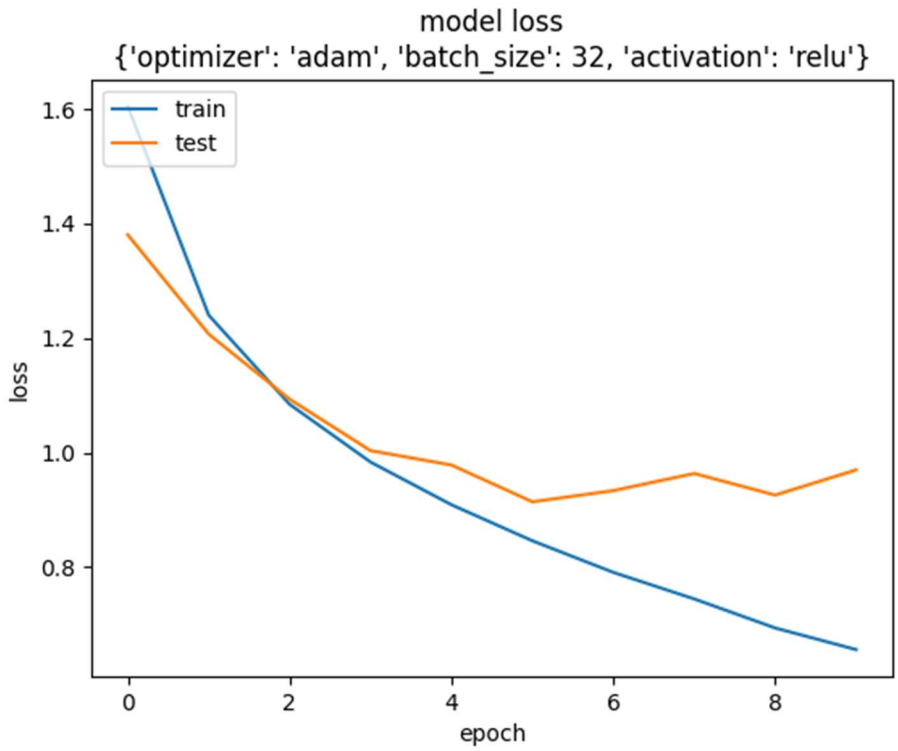
Pav. 4 Modelio tikslumas per 10 epochų, hiperparametrai įvardinti grafiko pavadinime



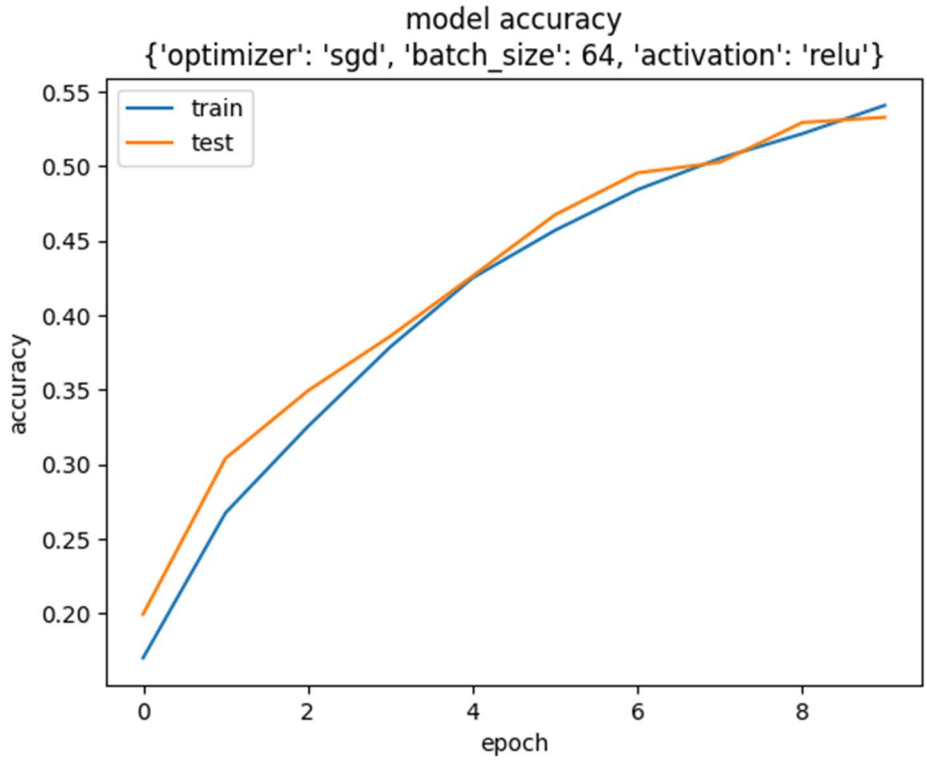
Pav. 5 Modelio klaida per 10 epochų



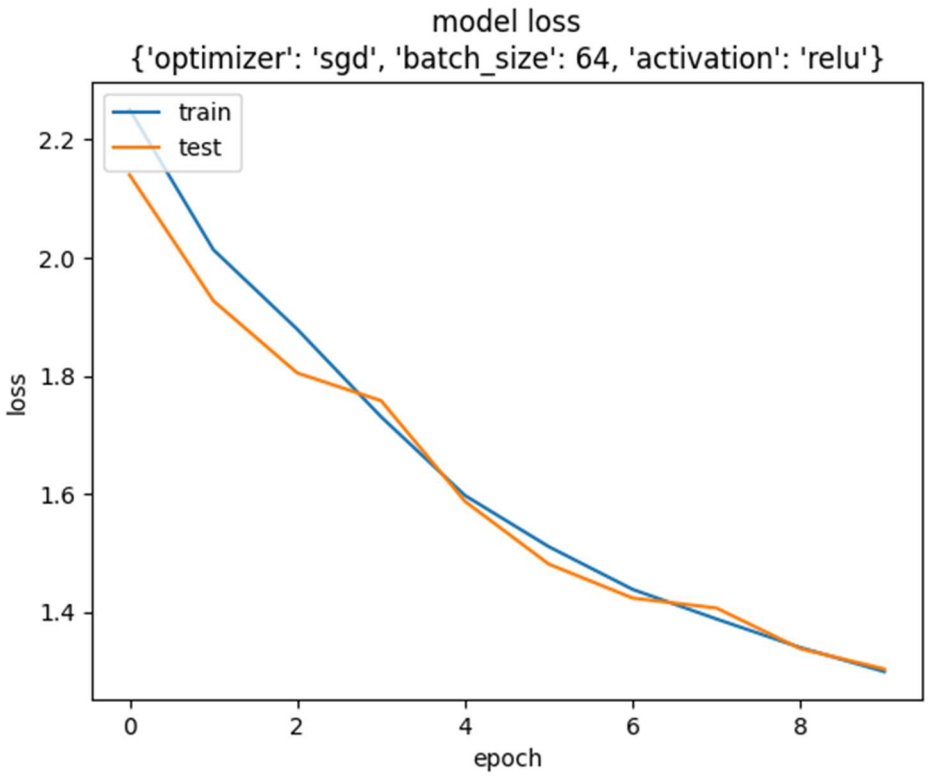
Pav. 6 Modelio tikslumas per 10 epochų



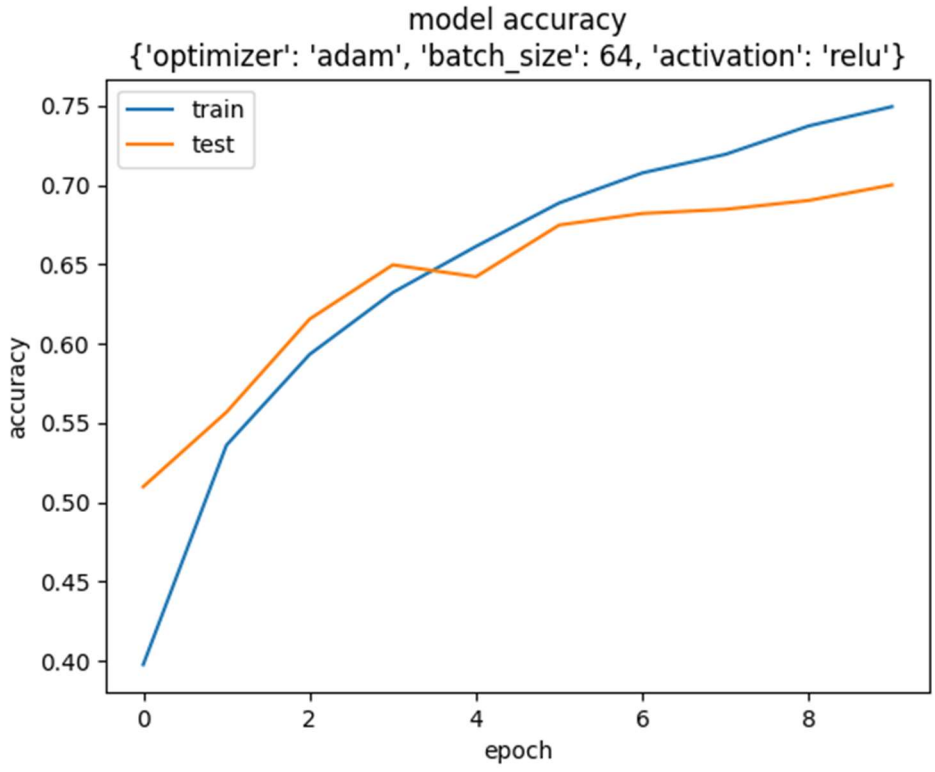
Pav. 7 Modelio klaida per 10 epochų



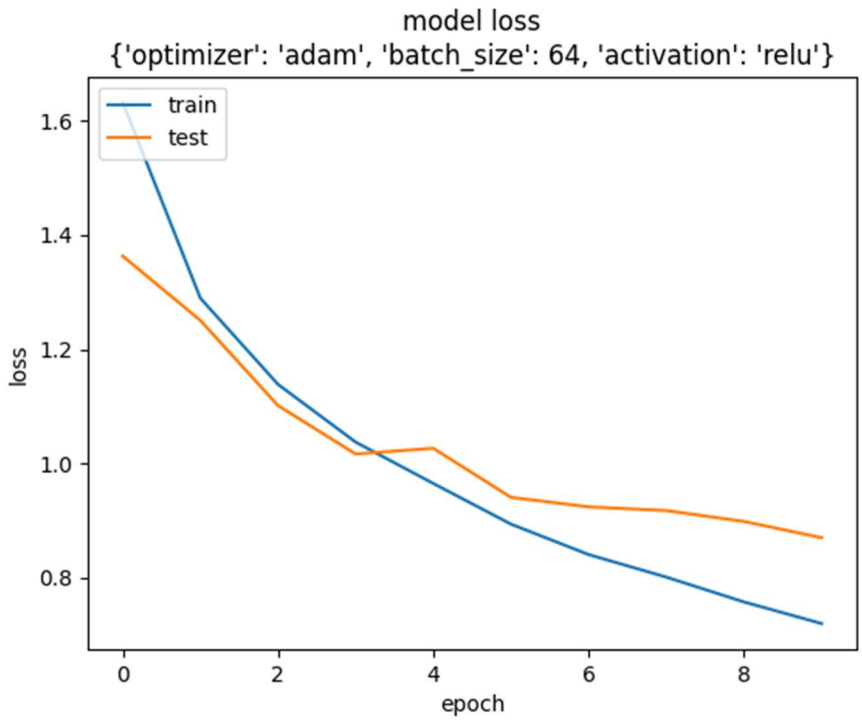
Pav. 8 Modelio tikslumas per 10 epochų



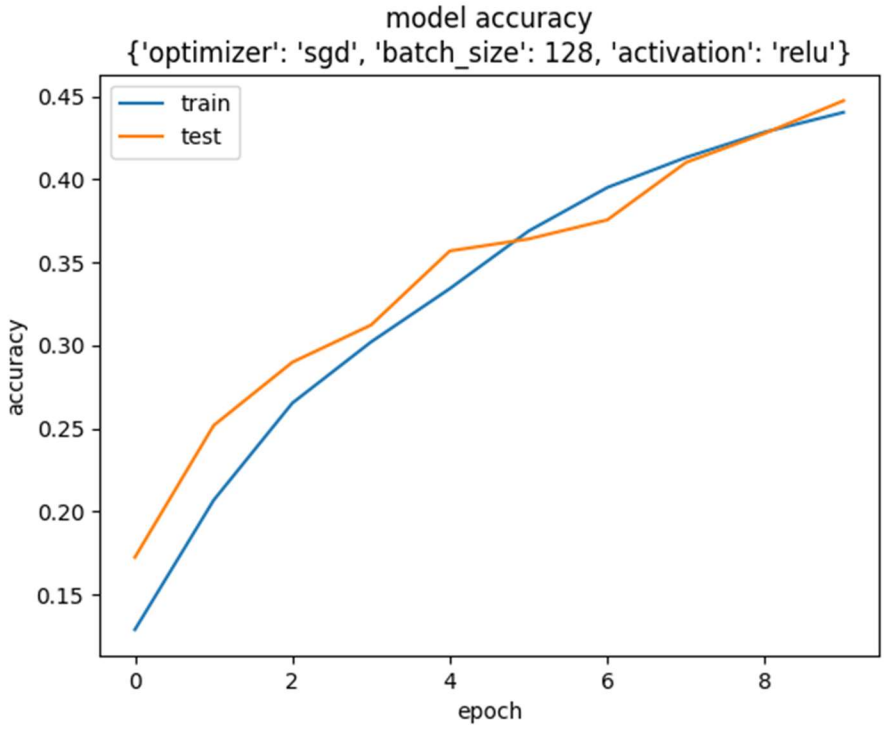
Pav. 9 Modelio klaida per 10 epochų



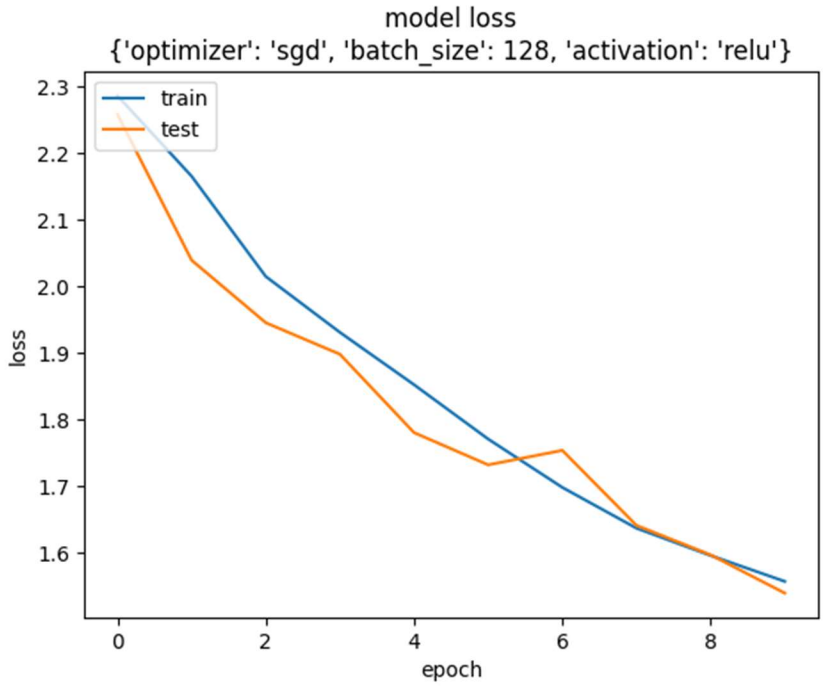
Pav. 10 Modelio tikslumas per 10 epochų



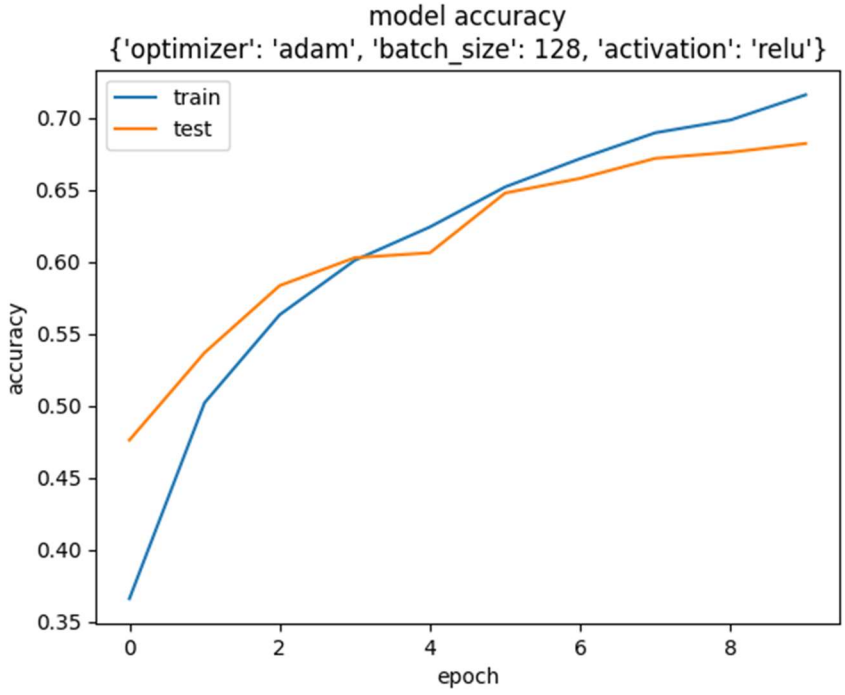
Pav. 11 Modelio klaida per 10 epochų



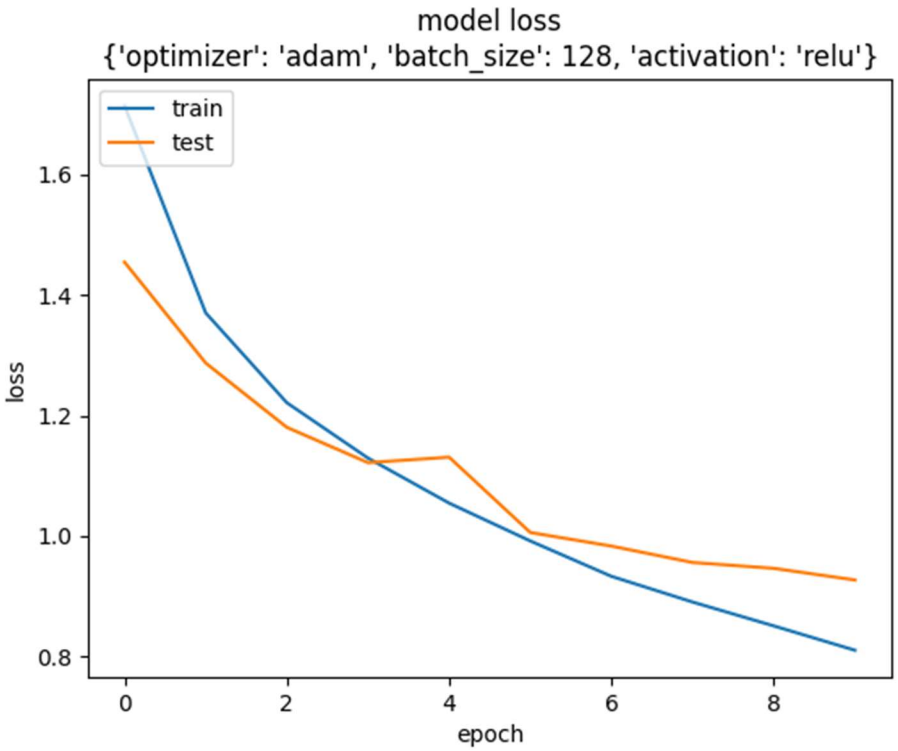
Pav. 12 Modelio tikslumas per 10 epochų



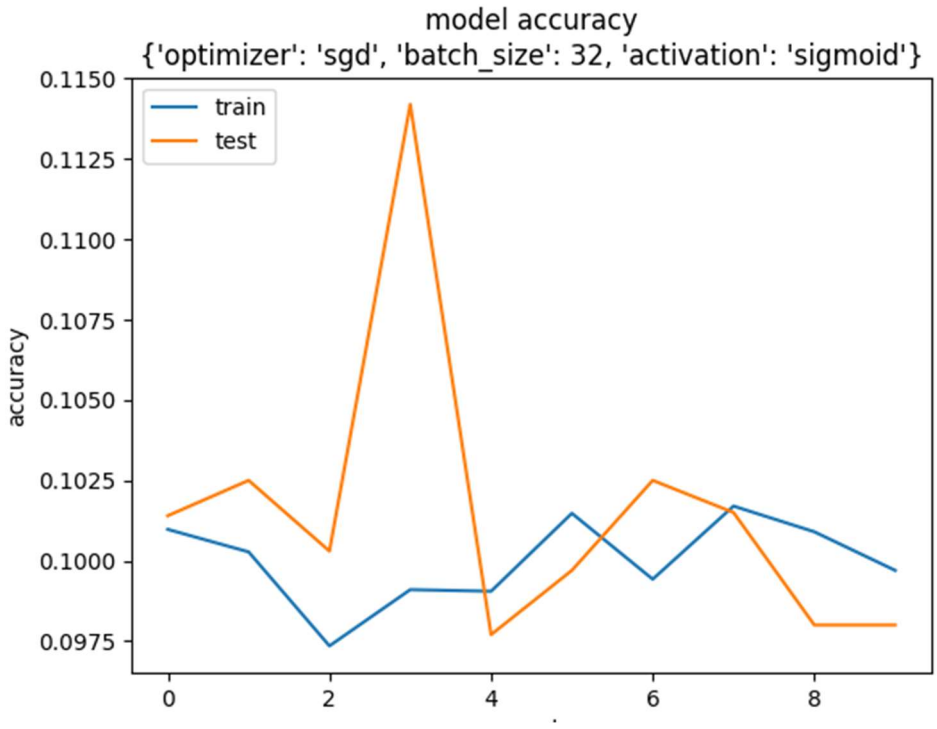
Pav. 13 Modelio klaida per 10 epochų



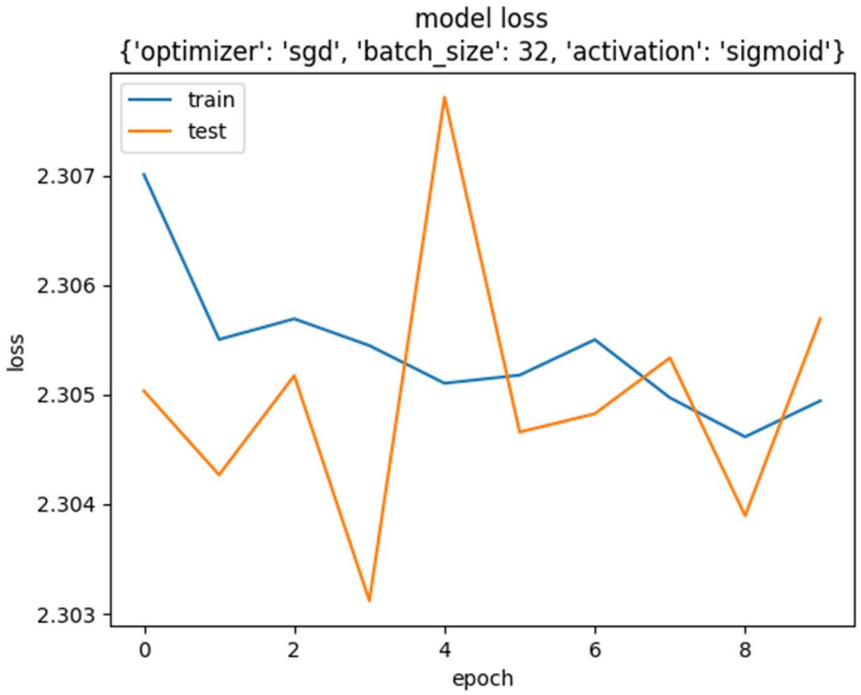
Pav. 14 Modelio tikslumas per 10 epochų



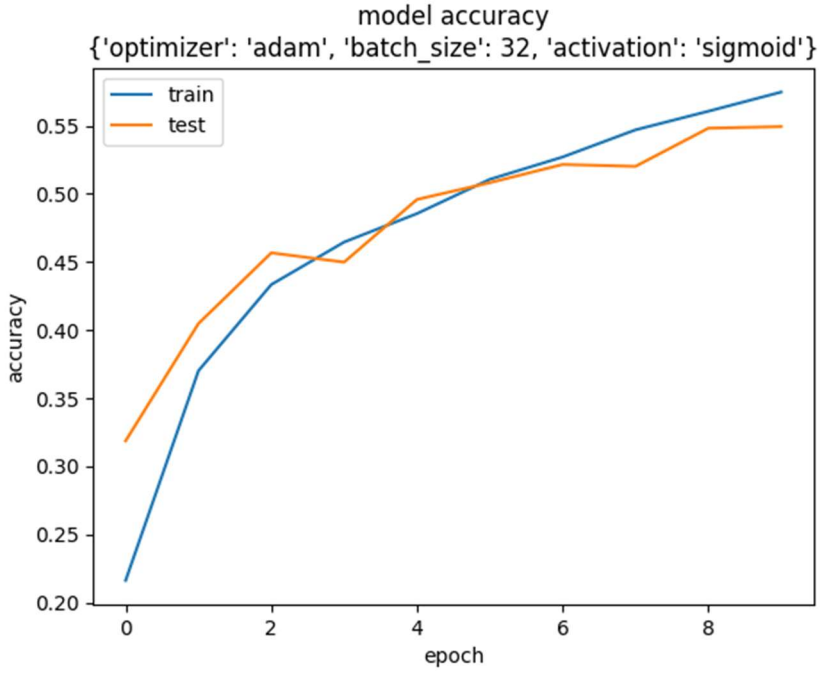
Pav. 15 Modelio klaida per 10 epochų



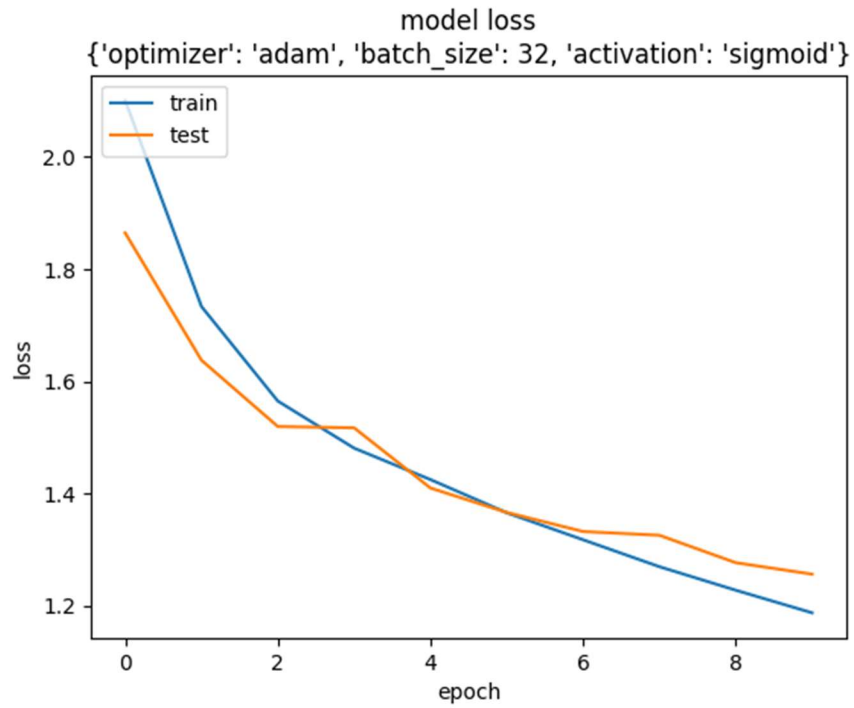
Pav. 16 Modelio tikslumas per 10 epochų



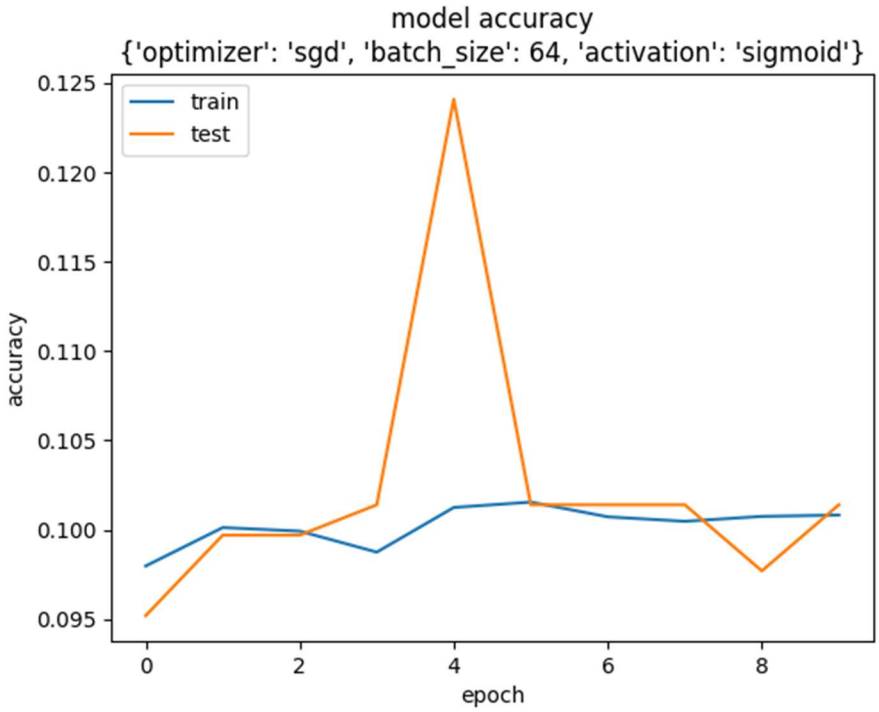
Pav. 17 Modelio klaida per 10 epochų



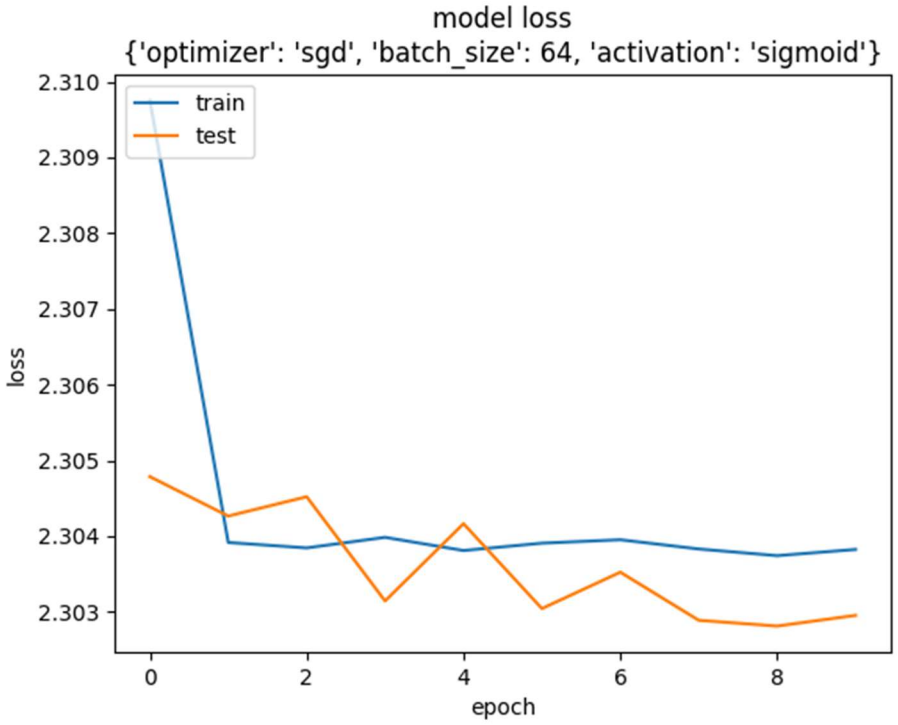
Pav. 18 Modelio tikslumas per 10 epochų



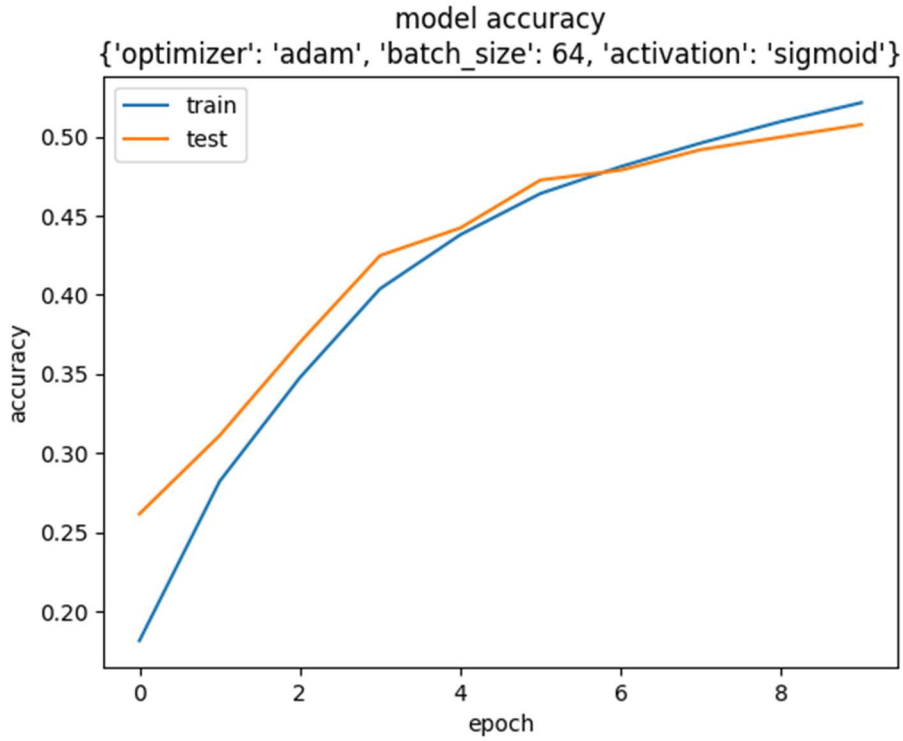
Pav. 19 Modelio klaida per 10 epochų



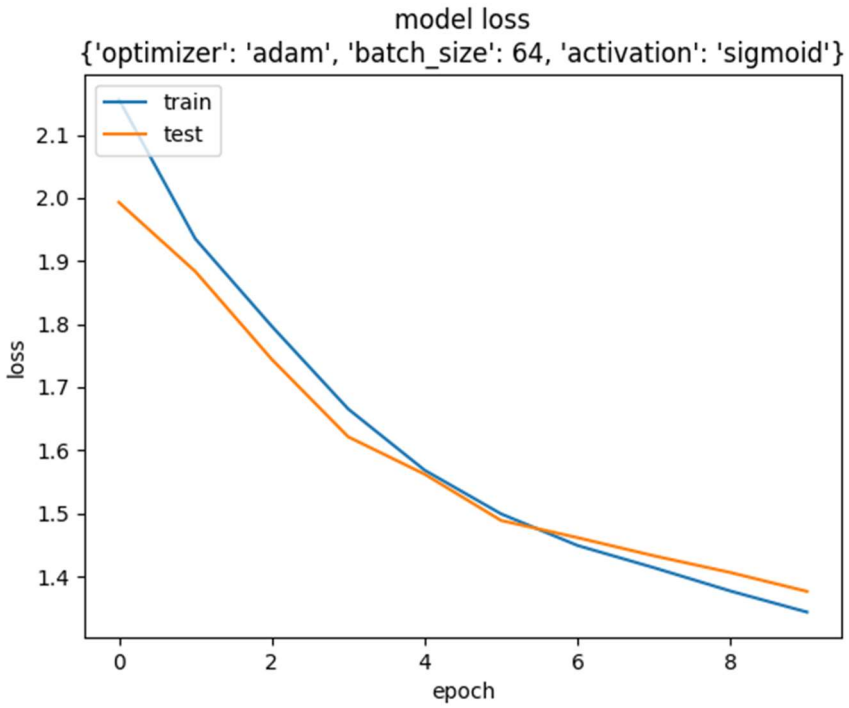
Pav. 20 Modelio tikslumas per 10 epochų



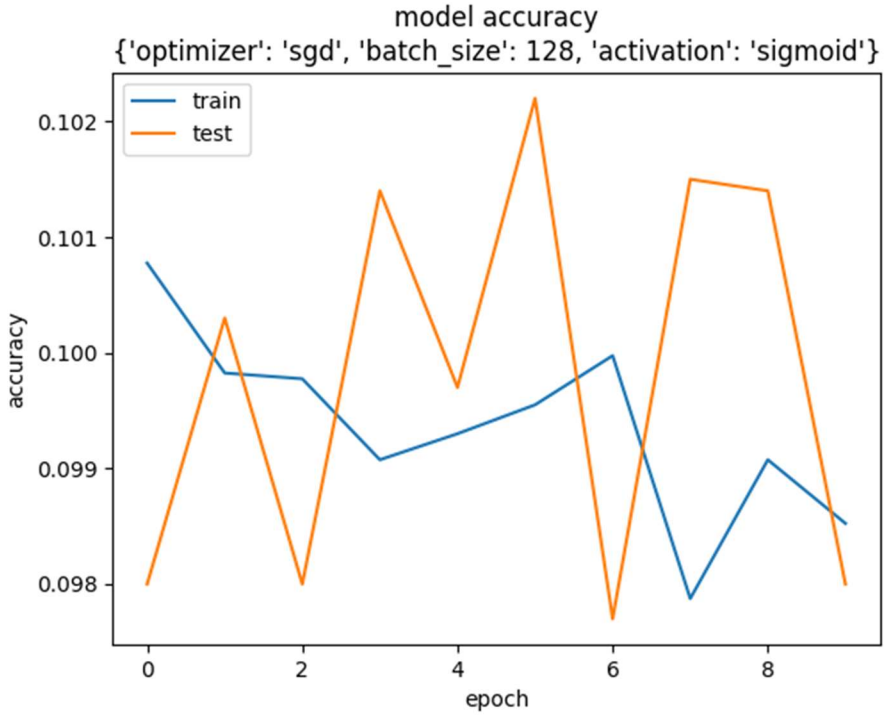
Pav. 21 Modelio klaida per 10 epochų



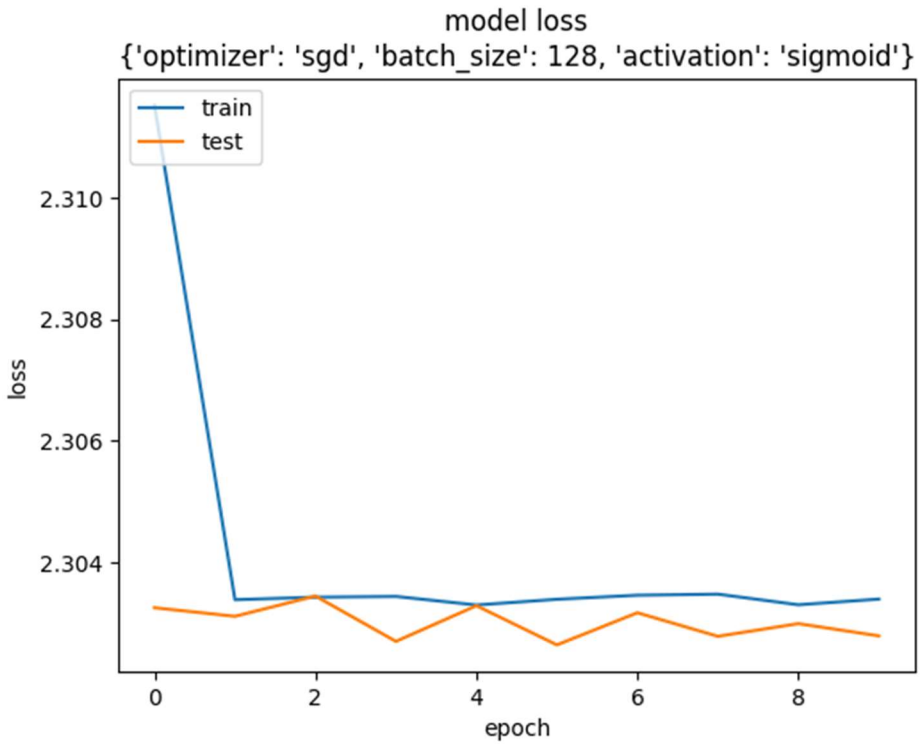
Pav. 22 Modelio tikslumas per 10 epochų



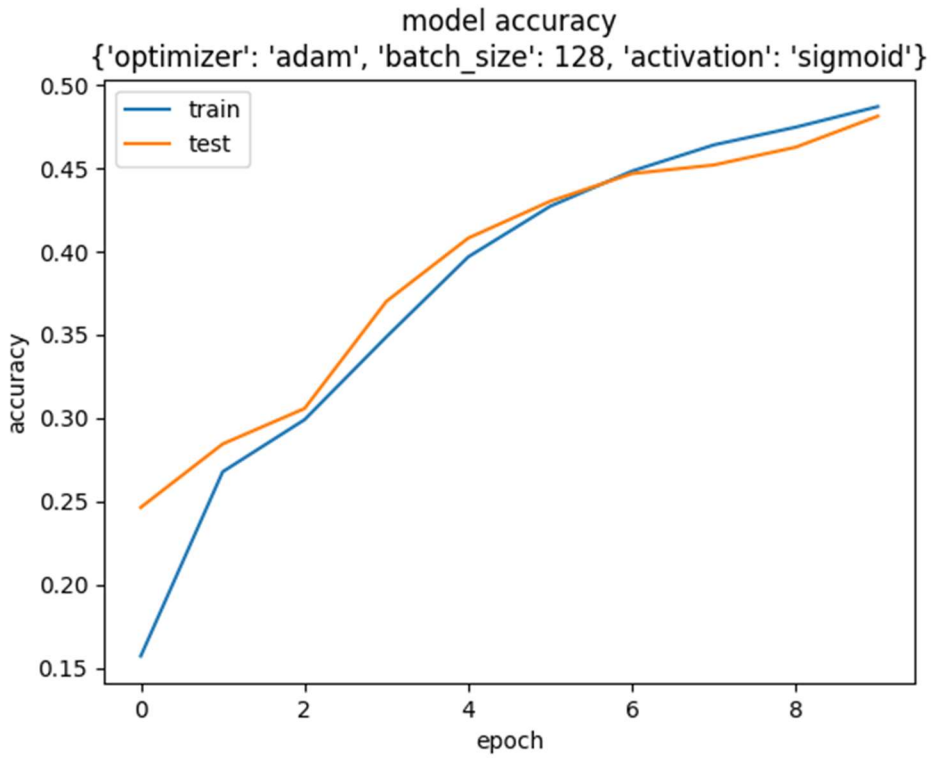
Pav. 23 Modelio klaida per 10 epochų



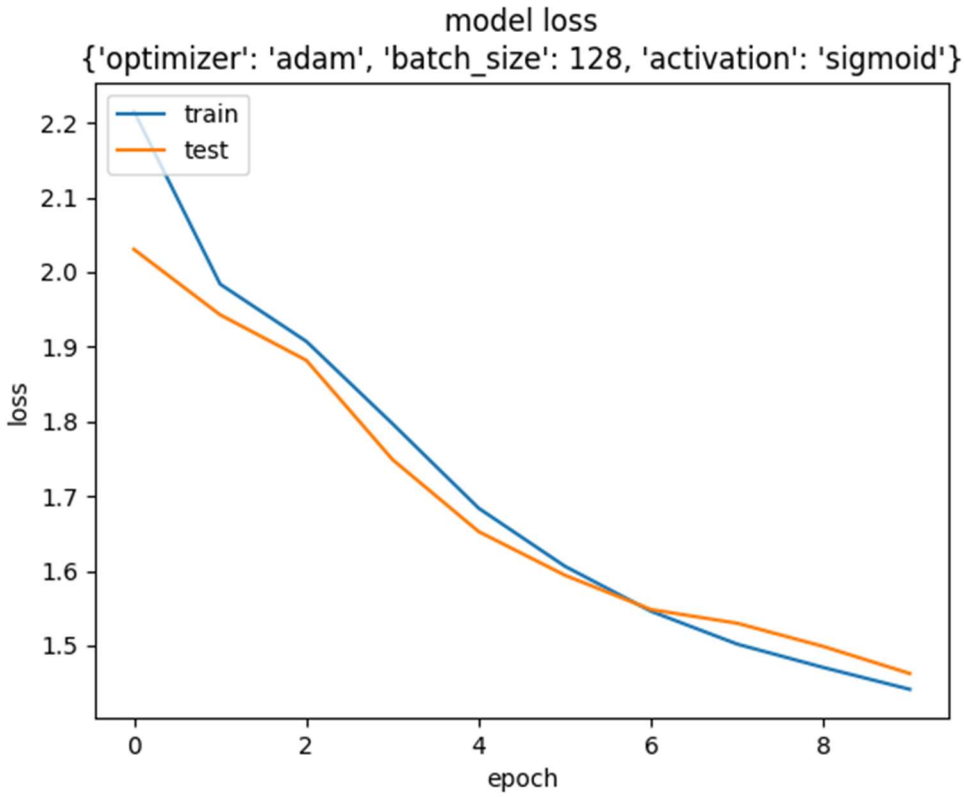
Pav. 24 Modelio tikslumas per 10 epochų



Pav. 25 Modelio klaida per 10 epochų



Pav. 26 Modelio tikslumas per 10 epochų



Pav. 27 Modelio klaida per 10 epochų

Klasifikavimo matrica (confusion matrix)

Klasifikavimo matricoje matyti kad geriausiai atpažįstami yra automobiliai (803), varlės (823), ir laivai (847). Lėktuvai yra dažnai klaidinami su laivais (105).

```
[[759  16  39   9  19   5  11   9 105  28]
 [ 32 803   3   8   5   4  10   2  48  85]
 [ 89   9 550  49 101  59  71  32  32   8]
 [ 21  15  81 489  85 144  83  27  42  13]
 [ 24   4  80  38 676  22  67  60  26   3]
 [ 13   2  71 196  60 538  35  49  27   9]
 [ 10   3  56  43  28  10 823   4  18   5]
 [ 26   3  30  43  87  52  14 721   7  17]
 [ 76  28   5   5   8   5   3   1 847  22]
 [ 25  91   8  12   6   3  12  10  51 782]]
```

Pav. 28 Apskaičiuota klasifikavimo matrica su tiksliausiu modeliu (nr. 3) modelio tikslumas – 0.6988

30 Atsitiktinai pasirinktų įrašų klasifikavimo rezultatai

Buvo atsitiktinai parinkti įrašai ir naudotas modelis juos klasifikuoti.

	Correct	Label	Prediction
0	True	Frog	Frog
1	False	Cat	Deer
2	True	Ship	Ship
3	True	Plane	Plane
4	False	Dog	Bird
5	False	Cat	Ship
13	False	Cat	Dog
14	True	Bird	Bird
15	False	Plane	Bird
16	True	Car	Car
17	False	Car	Truck
18	True	Car	Car
19	True	Frog	Frog
20	True	Plane	Plane
21	False	Ship	Plane
22	True	Cat	Cat
23	False	Bird	Deer
24	True	Frog	Frog
25	True	Car	Car
26	True	Dog	Dog
27	False	Deer	Bird
28	True	Ship	Ship
29	False	Ship	Truck

Pav. 29 30 įrašų klasifikavimo pavyzdys

Išvados

Klasifikavimui buvo naudojamas nuotraukų rinkinys CIFAR10 sudarytas iš 60000 nuotraukų, 10 klasių. Buvo sukurtas modelis pasiteliant TensorFlow biblioteką. Buvo sudarytos hiperparametrų kombinacijos iš šių hiperparametrų: aktyvacija – relu, sigmoid, paketo dydis – 32; 64; 128, optimizavimo algoritmas – sgd, adam. Nustatyta, kad tiksliausia kombinacija yra su hiperparametrais relu, 64, adam, kurios tikslumas buvo 0.6988. Naudojant sigmoid ir sgd hiperparametrus pastebėta, kad tikslumas yra labiausiai nepastovus ir šokinėjantis, todėl šių hiperparametrų kombinacija yra netinkama norint pasiekti gerą tikslumą. Pastebėta, kad tikslumą būtų galima didinti didinant epochų skaičių. Iš klasifikavimo matricos matyti, kad daugiausiai yra klaidinami lėktuvai su laivais. Sėkmingiausiai atpažįstami automobiliai, varlės ir laivai.

Programos kodas:

```
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.model_selection import ParameterGrid
from keras.datasets import cifar10
from keras import models, layers
from pathlib import Path
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
def displayDataset():
    for i in range(16):
        plt.subplot(4, 4, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(train_images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[train_labels[i][0]])

    plt.show()


def createModel(activationFun):
    model = models.Sequential()
    model.add(layers.Input(shape=(32, 32, 3)))
    model.add(layers.Rescaling(scale=1.0 / 255))
    model.add(layers.Conv2D(
        32, (3, 3), activation=activationFun, input_shape=(32, 32, 3)))
    model.add((layers.MaxPooling2D(2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation=activationFun))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation=activationFun))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation=activationFun))
    model.add(layers.Dense(10, activation='softmax'))

    return model


def trainModel(train_images, train_labels,
               epochs=10,
               activation='relu',
               batch_size=32,
               optimizer='adam',
               setNum=1):
    model = createModel(activation)
    global debounce
    if not debounce:
        model.summary()
        debounce = True

    model.compile(optimizer=optimizer,
                  loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
```

```
    history = model.fit(train_images, train_labels, epochs=epochs,
                        validation_split=0.2, batch_size=batch_size)

    model.save(f'models/modelWSet{setNum}')

    return model, history


def evaluateModel(model, history, test_images, test_labels, **kwargs):
    if not history == None:
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title(f'model accuracy\n{kwargs.__str__()}')
        plt.ylabel('accuracy')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()

        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title(f'model loss\n{kwargs.__str__()}')
        plt.ylabel('loss')
        plt.xlabel('epoch')
        plt.legend(['train', 'test'], loc='upper left')
        plt.show()

    loss, accuracy = model.evaluate(test_images, test_labels)

    print(f'Loss: {loss}')
    print(f'Accuracy: {accuracy}')

    return accuracy


if __name__ == '__main__':
    debounce = False
    # using cifar from https://www.cs.toronto.edu/~kriz/cifar.html
    (train_images, train_labels), (test_images, test_labels) =
cifar10.load_data()

    class_names = ['Plane', 'Car', 'Bird', 'Cat',
                   'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

    displayDataset()
```

```
    grid = ParameterGrid({'activation': ['relu', 'sigmoid'], 'batch_size':
[
                          32, 64, 128], 'optimizer': ['sgd', 'adam']})

    results = pd.DataFrame(list(grid))

    allAccuracies = []
    allModels = []
    allHistories = []

    for i in range(len(results)):
        if not Path(f'models/modelWSet{i}').exists():
            model, history = trainModel(
                train_images, train_labels, **grid[i], setNum=i)
            accuracy = evaluateModel(
                model, history, test_images, test_labels, **grid[i])
            allAccuracies.append(accuracy)
            allModels.append(model)
            allHistories.append(history)
            print(f'iteration {i}')

    for i in range(len(results)):
        model = models.load_model(f'models/modelWSet{i}')
        accuracy = evaluateModel(model, None, test_images, test_labels)
        allAccuracies.append(accuracy)
        allModels.append(models)

    bestIndex = allAccuracies.index(max(allAccuracies))

    print(f'most accurate set: modelWSet{bestIndex}')
    print('model results:')
    results['test_accuracies'] = allAccuracies
    print(results)

    bestModel = models.load_model(f'models/modelWSet{bestIndex}')

    yPred = bestModel.predict(test_images)
    yPred = [np.argmax(i) for i in yPred]
    confusion_matrix = confusion_matrix(test_labels, yPred)
    print(confusion_matrix)
    testAccuracy = accuracy_score(test_labels, yPred)
    print(f'Best model accuracy: {testAccuracy}')

    resultsdf = pd.DataFrame(classification_report(
```

```
        test_labels, yPred, output_dict=True))
    resultsdf.columns = np.concatenate(
        (class_names,  resultsdf.columns[10:].values))
    resultsdf

    print('prediction fragment:')

    label = []
    prediction = []
    correct = []

    indices = np.random.choice(
        np.where(np.array(yPred) == np.array(yPred))[0], 30)
    for i in indices:
        correct.append(yPred[i] == test_labels[i][0])
        prediction.append(class_names[yPred[i]])
        label.append(class_names[test_labels[i][0]])

    df = pd.DataFrame(
        {'Correct': correct, 'Label': label, 'Prediction': prediction})
    print(df)
```