VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Praktinė užduotis Nr. 4

Vaizdų klasifikavimas naudojant konvoliucinius neuroninius tinklus

Atliko: Programų sistemų 4 k. 1 gr. stud. Rokas Petrauskas

VILNIUS, 2023

Turinys

Tikslas

Tikslas yra apmokyti konvoliucinį neuroninį tinklą vaizdams klasifikuoti, atlikti tyrimą ir nustatyti kaip skirtingi parametrai lemia modelio rezultatus.

Naudojami duomenys

Dirbtinio neuroninio tinklo mokymui ir testavimui naudota Fashion MNIST duomenų aibė: https://www.kaggle.com/datasets/zalando-research/fashionmnist

Fashion-MNIST yra Zalando puslapio produktų nuotraukų rinkinys. Jį sudaro 60 000 mokymo ir 10 000 testavimo įrašų, kuris kiekvienas yra 28x28 pikselių dydžio juodai balta nuotrauka. Nuotraukos yra saugomos CSV formatu, kur pirmoji vertė yra klasės numeris, o kiekviena likusi kableliu atskirta vertė yra konkretaus pikselio tamsumo vertė nuo 1 iki 255. Kiekviena klasė atitinka skirtingo tipo drabužį.

Duomenų paruošimas

Visi (mokymo ir testavimo) įrašai sudedami į vieną failą ir padalinami į atskirus mokymo, validavimo, testavimo failus proporcija 80:10:10 tai reiškia, kad mokymui naudojau 56 000, o validacijai ir testavimui po 7 000 įrašų. Šie failai yra paduodami programai, kur pikselių tamsumo vertės yra normalizuojamos į intervalą (0;1).

Programos kodas

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.metrics import confusion_matrix
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense

train_data = pd.read_csv('data/train.csv')
validate_data = pd.read_csv('data/validate.csv')
test_data = pd.read_csv('data/test.csv')

X_train = train_data.iloc[:, 1:].values.reshape(-1, 28, 28, 1) / 255
y_train = pd.get_dummies(train_data.iloc[:, 0]).values
X_val = validate_data.iloc[:, 1:].values.reshape(-1, 28, 28, 1) / 255
y_val = pd.get_dummies(validate_data.iloc[:, 0]).values
X_test = test_data.iloc[:, 1:].values.reshape(-1, 28, 28, 1) / 255
y_test = pd.get_dummies(test_data.iloc[:, 0]).values

model = tf.keras.models.Sequential()

model.add(Conv2D(32, kernel_size=(5, 5), activation='relu', input_shape=(28, 28,
1))) #num of kernels #kernel size
model.add(MaxPooling2D(pool_size=(2, 2))) #filter size
model.add(Dropout(0.25)) #dropout probability
model.add(Flatten())
model.add(Dense(10, activation='softmax')) #activation function
```

```
optimizer = tf.keras.optimizers.Adadelta(learning_rate=0.01)  # learning rate
#optimizer

model.compile(loss=tf.keras.losses.categorical_crossentropy, # loss function
              optimizer=optimizer,
              metrics=['accuracy'])

model.fit(X_train, y_train,
          batch_size=128, #batch size
          epochs=10, #num of epochs
          verbose=1,
          validation_data=(X_val, y_val))

score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])

y_pred = model.predict(X_test)
confusion_matrix = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
print('Confusion Matrix:')
print(confusion_matrix)
```

Tiek čia aprašyta, tiek vėliau nagrinėjamos architektūros sugeneruotos naudojant chatGPT pagalbą.

Hiperparametrai

Modelyje naudojama 2 aktyvacijos funkcijos, pirmoji konvoliuciniame sluoksnyje naudojama „ReLu“, o galutinė aktyvacijos funkcija „softmax“. Modelis apmokomas per 10 epochų, mokymosi greitis 0,01, paketo dydis 128. Konvoliuciniame sluoksnyje naudojami 32 branduoliai, jų dydis 5x5. Filtro dydis sujungimo sluoksnyje 2x2. Išmetimo sluoksnyje naudojama tikimybe 0,25. Optimizavimo funkcija Adedelta, klaidos funkcija „categorical cross entropy“.

Naudojami skaičiavimo resursai

Programa leidžiama asmeniniame „Microsoft Windows 10 Home“ operacinę sistemą naudojančiame kompiuteryje. Skaičiavimai atlikti pasitelkiant CPU „Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz 4 Cores, 8 Logical Processors“ su 8 GB RAM.

Rezultatai

Bandytos 3 skirtingos architektūros:

Modelis 1:

```
model = tf.keras.models.Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), activation='relu', input_shape=(28, 28,
1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(10, activation='softmax')) #activation function
```

Klaida mokymo duomenims paskutinėje epochoje: 0,6564
Tikslumas mokymo duomenims paskutinėje epochoje: 0,765
Klaida validavimo duomenims paskutinėje epochoje: 0,6310
Tikslumas validavimo duomenims paskutinėje epochoje: 0,7791
Klaida testavimo duomenims: 0,6255
Tikslumas testavimo duomenims: 0,7831

Modelis 2:

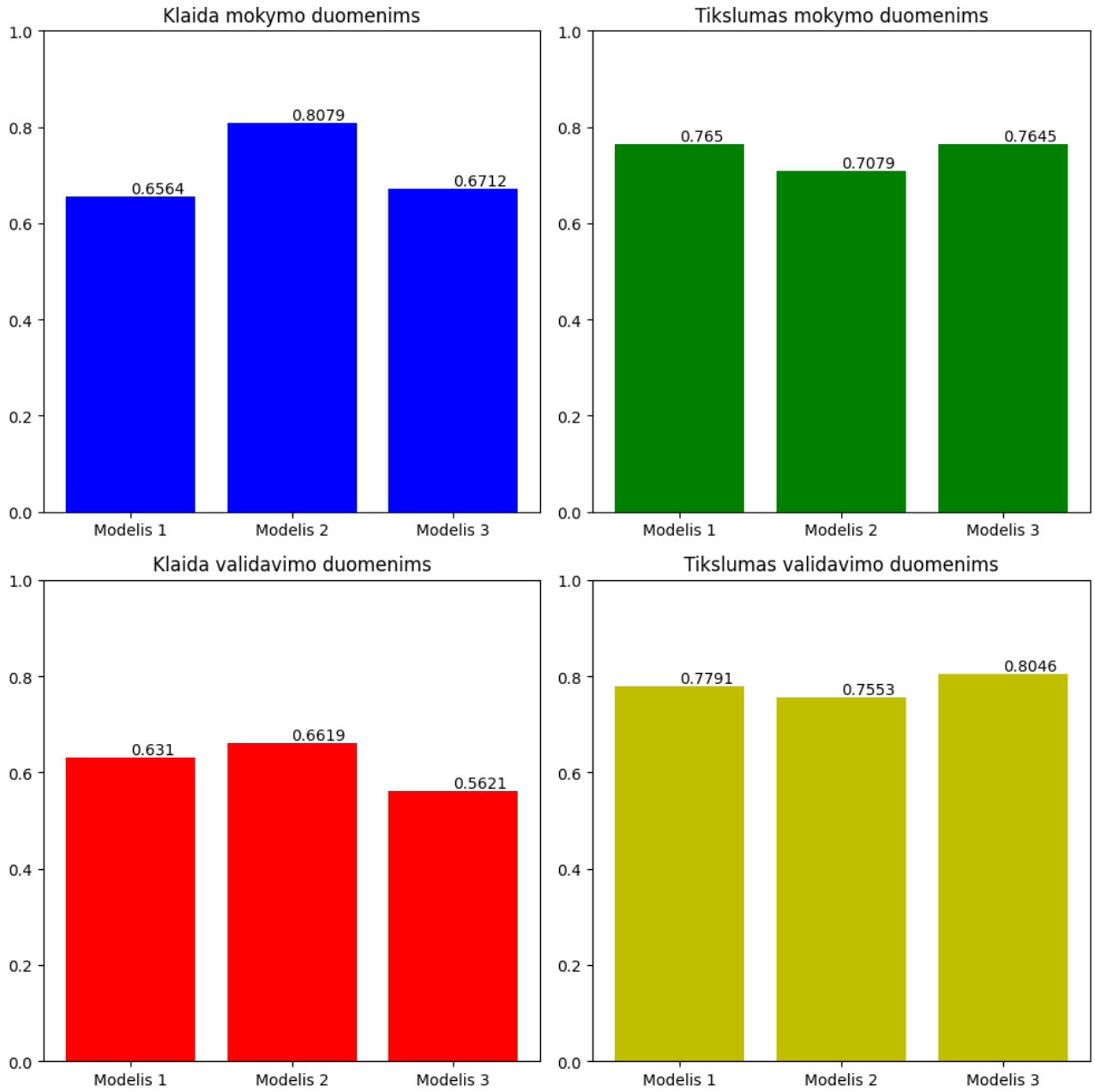
```
model = tf.keras.models.Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Klaida mokymo duomenims paskutinėje epochoje: 0,8079
Tikslumas mokymo duomenims paskutinėje epochoje: 0,7079
Klaida validavimo duomenims paskutinėje epochoje: 0,6619
Tikslumas validavimo duomenims paskutinėje epochoje: 0,7553
Klaida testavimo duomenims: 0,6668
Tikslumas testavimo duomenims: 0,7571

Modelis 3:

```
model = tf.keras.models.Sequential()
model.add(Conv2D(64, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Klaida mokymo duomenims paskutinėje epochoje: 0,6712
Tikslumas mokymo duomenims paskutinėje epochoje: 0,7645
Klaida validavimo duomenims paskutinėje epochoje: 0,5621
Tikslumas validavimo duomenims paskutinėje epochoje: 0,8046
Klaida testavimo duomenims: 0,5648
Tikslumas testavimo duomenims: 0,8051

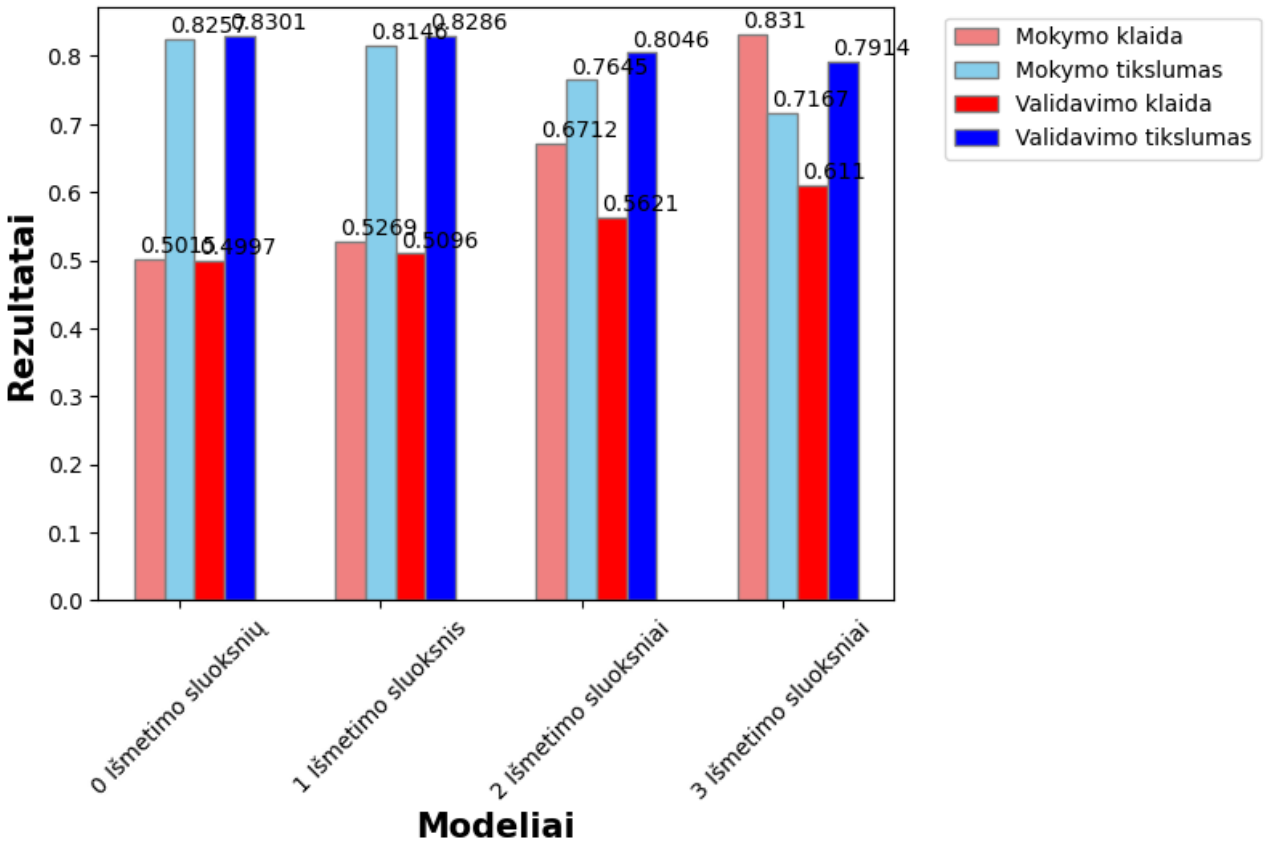


1 pav. Skirtingų architektūrų modelių tikslumai ir klaida.

Pirmame paveikslėlyje matomi skirtingų architektūrų modelių tikslumai ir klaida mokymo bei validavimo duomenims. Kadangi rezultatai geriausi naudojant trečio modelio architektūrą, jo hiperparametrai ir sluoksniai keičiami tolimesnei tyrimo eigai vykdyti.

Išmetimo sluoksnių svarba rezultatams

Antrame paveikslėlyje matomi rezultatai mokymo ir validavimo duomenims su skirtingu kiekiu išmetimo sluoksnių. Modelis 3 naudoja 2 išmetimo sluoksnius su 0,25 ir 0,5 išmetimo tikimybėmis, jis pavadintas „2 Išmetimo sluoksniai“, pašalinus 0,5 išmetimo sluoksnį gauti rezultatai pažymėti pavadinimu „1 Išmetimo sluoksnis“, pašalinus 0,25 išmetimo sluoksnį gauti rezultatai pažymėti pavadinimu „0 Išmetimo sluoksnių“. Taip pat 3 modeliui pridėjau papildomą 0,5 tikimybės išmetimo sluoksnį, jo rezultatai pažymėti pavadinimu „3 Išmetimo sluoksniai“. Analizuojant rezultatus matyti, kad didinant išmetimo sluoksnių kiekį, klaida tolsta nuo 0 tiek mokymo, tiek validavimo duomenims, o tikslumas mažėja.

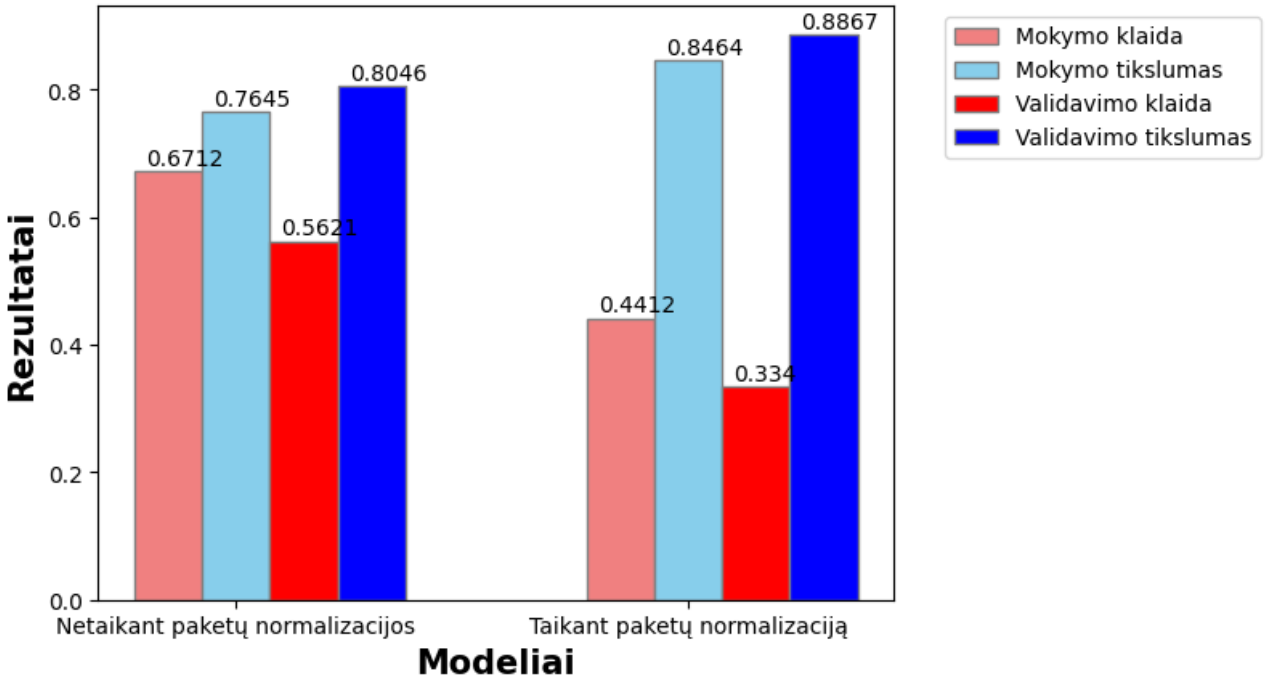


2 pav. Skirtingų išmetimo sluoksnių kiekio modelių rezultatai.

Paketų normalizavimo svarba rezultatams

Rezultatai gauti naudojant 3 modelį (2 išmetimo sluoksniai) pridėjus paketų normalizavimo sluoksnį:
Klaida mokymo duomenims paskutinėje epochoje: 0,4412
Tikslumas mokymo duomenims paskutinėje epochoje: 0,8464
Klaida validavimo duomenims paskutinėje epochoje: 0,3340
Tikslumas validavimo duomenims paskutinėje epochoje: 0,8867

3 Paveikslėlyje palyginti 3 modelio rezultatai taikant paketų normalizaciją ir jos netaikant. 3 Modelis netaiko paketų normalizacijos, todėl pridėjus šį sluoksnį po konvoliucinio sluoksnio gaunamas alternatyvus modelis. Matoma, kad taikant paketų normalizaciją gaunami geresni rezultatai.



3 pav. Paketų normalizavimo svarba rezultatams.

Aktyvacijos funkcijos svarba rezultatams

Rezultatai gauti naudojant 3 modelį (2 išmetimo sluoksniai) pakeitus paskutinę aktyvacijos funkciją iš „softmax“ į „sigmoid“:

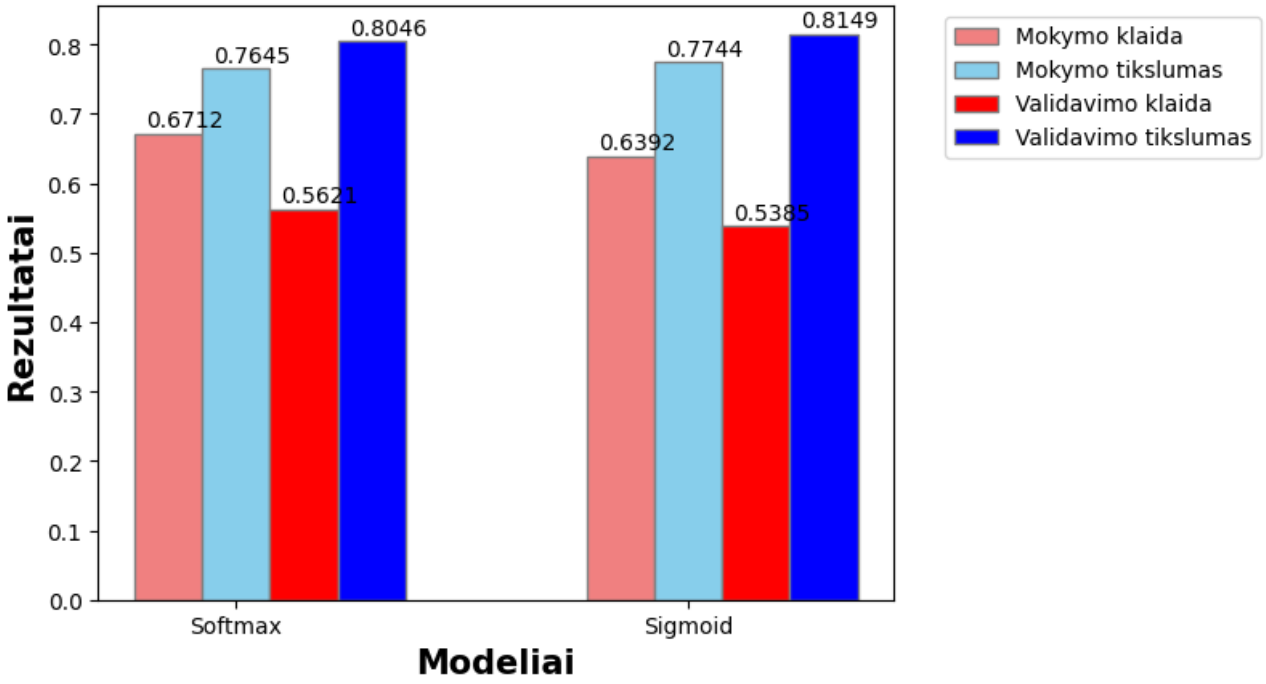
Klaida mokymo duomenims paskutinėje epochoje: 0,6392

Tikslumas mokymo duomenims paskutinėje epochoje: 0,7744

Klaida validavimo duomenims paskutinėje epochoje: 0,5385

Tikslumas validavimo duomenims paskutinėje epochoje: 0,8149

4 Paveikslėlyje palyginti 3 modelio rezultatai naudojant paskutinę aktyvacijos funkciją kaip „softmax“ ir „sigmoid“. Matomi nežymūs skirtumai.



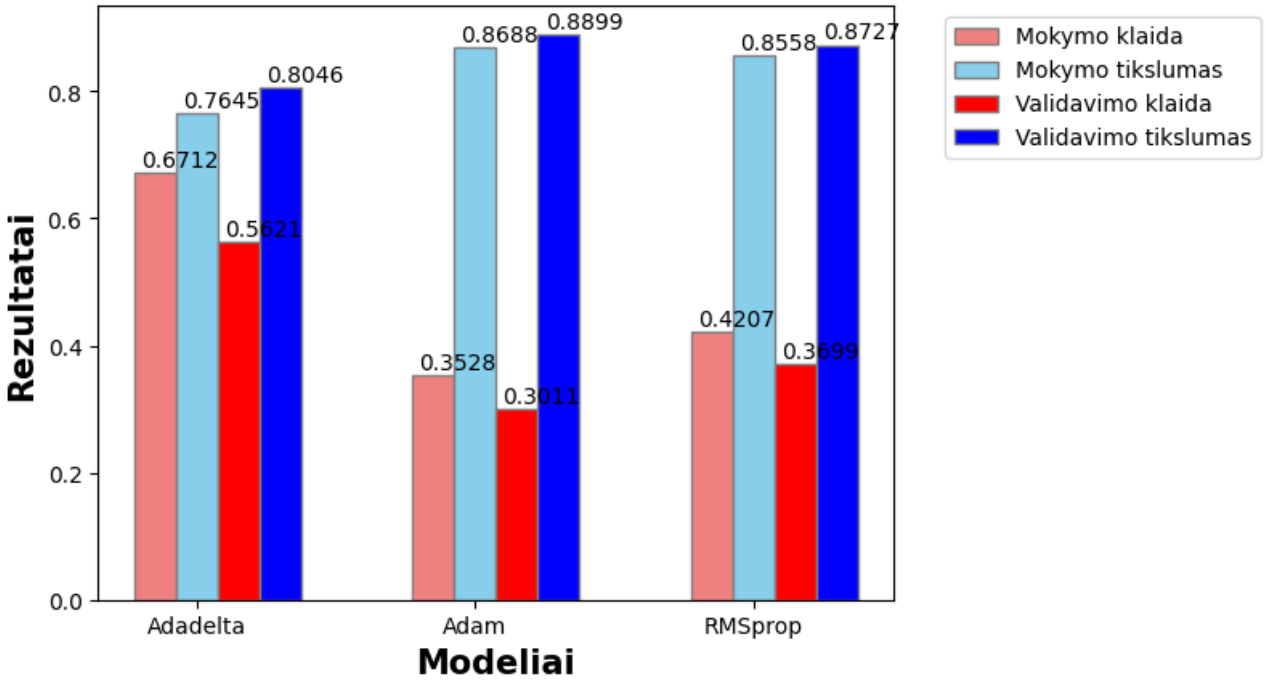
4 pav. Skirtingos aktyvacijos funkcijos.

Optimizavimo algoritmo svarba rezultatams

Rezultatai gauti naudojant 3 modelį (2 išmetimo sluoksniai) naudojant „Adam“ optimizavimo algoritmą:
Klaida mokymo duomenims paskutinėje epochoje: 0,3528
Tikslumas mokymo duomenims paskutinėje epochoje: 0,8688
Klaida validavimo duomenims paskutinėje epochoje: 0,3011
Tikslumas validavimo duomenims paskutinėje epochoje: 0,8899

Rezultatai gauti naudojant 3 modelį (2 išmetimo sluoksniai) naudojant „RMSprop“ optimizavimo algoritmą:
Klaida mokymo duomenims paskutinėje epochoje: 0,4207
Tikslumas mokymo duomenims paskutinėje epochoje: 0,8558
Klaida validavimo duomenims paskutinėje epochoje: 0,3699
Tikslumas validavimo duomenims paskutinėje epochoje: 0,8727

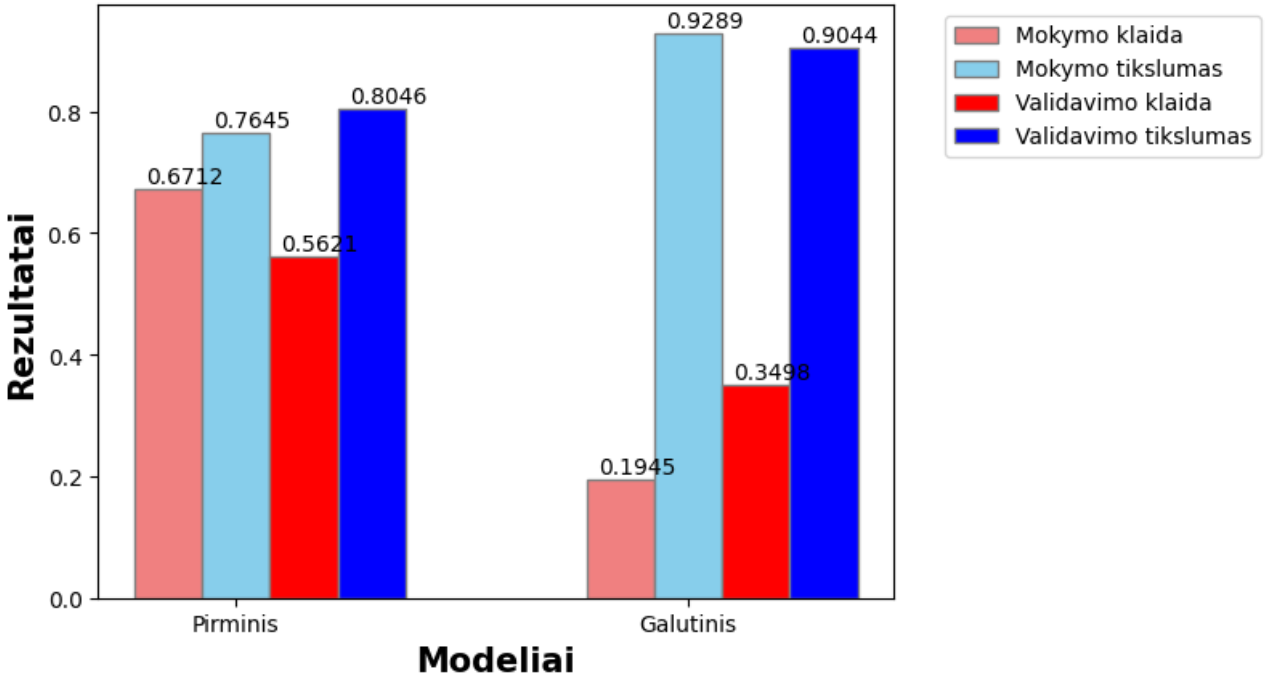
5 paveikslėlyje palyginti 3 modelio rezultatai naudojant „Adadelta“, „Adam“, „RMSprop“ optimizavimo algoritmus. Matoma, kad naudojant „Adam“ optimizavimo algoritmą gaunami geriausi rezultatai.



5 pav. Skirtingi optimizavimo algoritmai.

Galutinis modelis

Pasitelkiant anksčiau gautus rezultatus buvo pakoreguotas programos kodas. Imtas 3 modelis, pašalinti išmetimo sluoksniai, pridėtas paketų normalizavimo sluoksnis bei naudojamas „Adam“ optimizavimo algoritmas. 6 Paveikslėlyje palyginami pirminio ir galutinio modelio rezultatai mokymo ir validavimo duomenims. 1 Lentelėje pateikta galutinio modelio klasifikavimo matrica.



6 pav. Pirminio ir galutinio modelio rezultatai.

Galutinio modelio rezultatai:
Klaida testavimo duomenims: 0,3619002103805542
Tikslumas testavimo duomenims: 0,899142861366272

Lentelė 1. Galutinio modelio klasifikavimo matrica.

		Prognozuojamos klasės									
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Tikrosios klasės	**0**	544	7	8	83	5	0	44	1	10	0
	1	0	646	9	24	7	0	8	0	1	0
	2	4	2	425	1	160	0	104	0	9	0
	3	20	12	2	630	14	0	35	0	4	0
	4	0	3	113	38	479	1	56	0	4	0
	5	0	0	0	2	0	594	0	59	0	25
	6	143	3	130	51	116	1	235	0	17	0
	7	0	0	0	0	0	35	0	602	3	44
	8	2	0	11	20	3	5	21	2	626	3
	9	0	0	0	0	1	5	0	47	1	680

30 pavyzdinių įrašų klasių ir prognozių:
Tikroji klasė: 0, Prognozuojama klasė: 6
Tikroji klasė: 2, Prognozuojama klasė: 2
Tikroji klasė: 1, Prognozuojama klasė: 1
Tikroji klasė: 3, Prognozuojama klasė: 3
Tikroji klasė: 4, Prognozuojama klasė: 4
Tikroji klasė: 5, Prognozuojama klasė: 5
Tikroji klasė: 6, Prognozuojama klasė: 6
Tikroji klasė: 7, Prognozuojama klasė: 7
Tikroji klasė: 8, Prognozuojama klasė: 8
Tikroji klasė: 9, Prognozuojama klasė: 9
Tikroji klasė: 6, Prognozuojama klasė: 6
Tikroji klasė: 9, Prognozuojama klasė: 9
Tikroji klasė: 7, Prognozuojama klasė: 7
Tikroji klasė: 8, Prognozuojama klasė: 8
Tikroji klasė: 5, Prognozuojama klasė: 5
Tikroji klasė: 0, Prognozuojama klasė: 0
Tikroji klasė: 1, Prognozuojama klasė: 1
Tikroji klasė: 4, Prognozuojama klasė: 4
Tikroji klasė: 3, Prognozuojama klasė: 3
Tikroji klasė: 2, Prognozuojama klasė: 2
Tikroji klasė: 5, Prognozuojama klasė: 5
Tikroji klasė: 4, Prognozuojama klasė: 4
Tikroji klasė: 6, Prognozuojama klasė: 6
Tikroji klasė: 3, Prognozuojama klasė: 3
Tikroji klasė: 7, Prognozuojama klasė: 7
Tikroji klasė: 2, Prognozuojama klasė: 2
Tikroji klasė: 8, Prognozuojama klasė: 8
Tikroji klasė: 1, Prognozuojama klasė: 1
Tikroji klasė: 9, Prognozuojama klasė: 9
Tikroji klasė: 0, Prognozuojama klasė: 0

Išvados

Tyrimo metu bandytos 3 skirtingos architektūros, išrinkus šiam uždaviniui geriausius rezultatus gaunančia architektūrą, prie jos ir apsistota. Eigoje buvo keičiami hiperparametrai bei pridedami ar šalinami tinklo sluoksniai siekiant gauti geriausius rezultatus. Pastebėta, kad visos architektūros pasiekia gana panašius rezultatus kai epochų skaičius lygus 10. Parinktas geriausias modelis, jis naudojo 2 šalinimo sluoksnius, todėl pabandžiau jo variacijas su 0, 1 ir 3 šalinimo sluoksniais. Pastebėta, kad rezultatai geriausi visai nenaudojant išmetimo sluoksnių. Nors išmetimo sluoksniai padeda išvengti per didelio pritaikymo kai modelis nesugeba atpažinti naujų, mokyme nenaudotų įrašų, tačiau šiam konkrečiam modeliui problema nėra aktuali, modelyje be išmetimo sluoksnių validacijos rezultatai yra geri lyginant su mokymo rezultatais. Taip pat pastebėta, kad paketų normalizavimo sluoksnis pagerino tikslumą 8% bei gerokai priartino klaidos vertę prie 0. Bandytos skirtingos aktyvacijos funkcijos, tačiau didelio skirtumo nepastebėjau, nors didelė tikimybė, kad naudojant kitas aktyvacijos funkcijas skirtumas gali būti didesnis. Optimizavimo algoritmų pakeitimas lėmė didžiausią rezultatų pagerėjimą, kur „Adam“ algoritmas tikslumą pagerino 10%, o klaidos vertę sumažino beveik dvigubai. Galiausiai pritaikius pastebėjimus naujo modelio kūrime matyti, kad hiperparametrai bei skirtingi sluoksniai gali labai stipriai veikti modelio rezultatus, mokymo laiką, skaičiavimo resursų apkrovą.