VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Duomenų mokslas, 3 kursas

Vaizdų klasifikavimas naudojant konvoliucinius neuroninius tinklus

Autorius: Augustė Raišytė

Vilnius
2023

Turinys

1 Tikslas ir uždaviniai

Tikslas: apmokyti konvoliucinį neuroninį tinklą vaizdams klasifikuoti, atlikti tyrimą.

Uždaviniai:

1. Paruošti duomenis konvoliuciniam neuroniniam tinklui mokyti.
2. Sukurti programą, kurioje būtų įgyvendintas konvoliucinis neuroninis tinklas vaizdams klasifikuoti.
3. Atlikti konvoliucinio neuroninio tinklo tyrimus su validavimo ir mokymo duomenimis.
4. Nustatyti, su kokiais hiperparametrais gaunamas didžiausias klasifikavimo tiklsumas ir mažiausia paklaida.

2 Duomenys

2.1 Duomenų aprašymas

Konvoliuciniam neuroniniam tinklui apmokyti buvo naudojami iš „Kaggle“ paimti „Fashion MNIST“ duomenys. Duomenis sudaro 70000 vaizdų duomenų įrašų. Pradiniai duomenys sudarė 10000 testavimo ir 60000 mokymo aibės įrašų. Kiekvienas duomenų įrašas yra 28x28 (784 pikseliai) dydžio pilko atspalvio vaizdas, priskirtas vienai iš 10 klasių. Vienas pikselis gali įgyti reikšmę nuo 0 iki 255 (priklausomai nuo tamsumo). Galimos klasių reikšmės yra šios: 0 „T-shirt/top“, 1 „Trouser“, 2 „Pullover“, 3 „Dress“, 4 „Coat“, 5 „Sandal“, 6 „Shirt“, 7 „Sneaker“, 8 „Bag“, 9 „Ankle boot“. Klasių reikšmės nurodomos pirmame stulpelyje.

2.2 Duomenų skaidymas į validavimo ir testavimo

Pirmiausia testavimo ir mokyo aibės buvo sujungtos į vieną failą. Tada su „Python“ programvimo kalba iš apjungto failo atsitiktinai atrenkami duomenys testavimo ir validavimo aibėms, o iš pradinės aibės panaikinami. Taigi 7000 duomenų patenka į mokymo aibę ir 7000 į validavimo aibę, o likę tampa mokymo aibe.

2.3 Duomenų paruošimas

Kad būtų galima apmokyti neuroninį tinklą, visų pirma patikrinama ar nėra praleistų duomenų reikšmių. Neradus praleistų reikšmių duomenys pirmiausia normalizuojami padalijant iš 255. Taip pikselių skaičius tampa intervale nuo 0 iki 1. Taip pat prieš neuroninio tinklo apmokymą pakeičiamas duomenų įrašų formatas į 28x28 dydžio matricą.

3 Tyrimas

3.1 Skaičiavimo resursai

Tyrimams atlikti buvo naudojamasi GPU.

Kompiuterio charakteristikos:

- GPU: NVIDIA GeForce RTX 3060
- VRAM: 6 GB
- Memory Type: GDDR6
- RAM: 2x8 GB
- RAM frequency: 3200 MHz
- CPU: AMD Ryzen 5 5600H
- CPU base speed: 3,3 GHz
- Cores: 6
- Logical processors: 12
- Sequential Read: 3500 MB/s
- Sequential Write: 3000 MB/s
- Random Read: 580K IOPS
- Random Write: 500K IOPS

3.2 Pradinė tinklo architektūra

Pradinę tinklo architektūrą sudarė 1 konvoliucijos sluoksnis, 1 sujungimo sluoksnis, 1 „Flatten“ bei 2 „Dense“ sluoksniai. Konvoliuciniame sluoksnyje buvo naudojami 128 filtrai, kurių dydis buvo 2x2 dydžio matrica. Konvoliucijoje buvo naudojama „ReLU“ aktyvacijos funkcija. Pirmajame „Dense“ sluoksnyje nustatomas neuronų skaičius lygus 256. Tikimybėms skaičiuoti naudojama „Softmax“ funkcija.

3.3 Rezultatų priklausomybė nuo tinklo architektūros

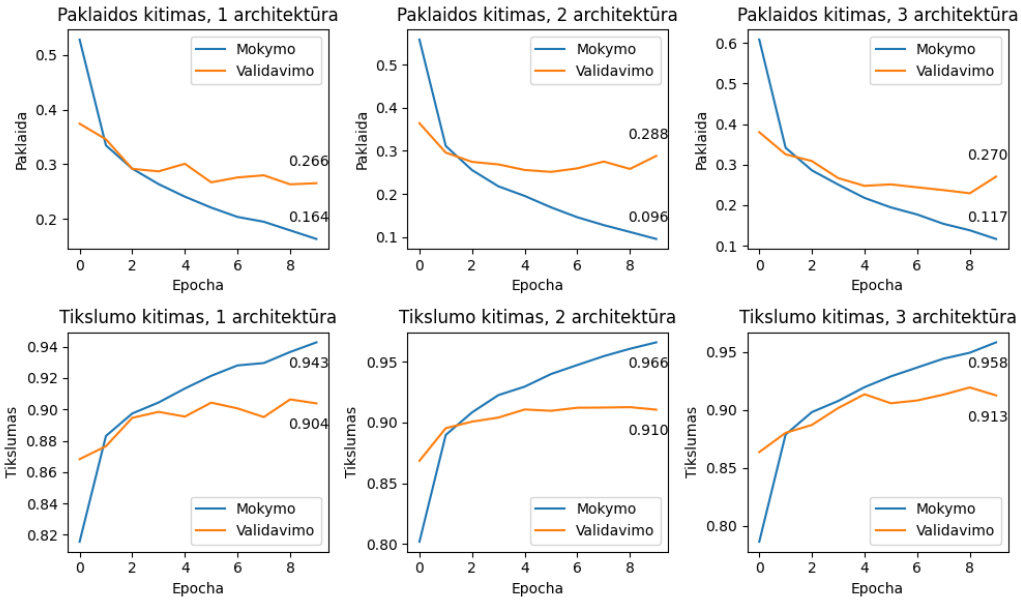
Pirmoje tyrimo dalyje buvo tiriama, kaip tikslumas ir paklaida priklauso nuo tinklo architektūros. Buvo pasirinkta naudoti 3 skirtingas architektūras. Visose architektūrose buvo naudojami 1 „Flatten“ ir 2 „Dense“ sluoksniai, „Adam“ optimizavimo algoritmas, 10 epochų, paketo dydis 512 ir sujungimo sluoksnio dydis 2x2 matrica. Keičiamas buvo tik konvoliucinių sluoksnių skaičius, filtrų skaičius ir sujungimo sluoksnio vieta.

Pirmąją architektūrą sudarė viskas taip kaip jau buvo minėta 3.2 skyrelyje.

Antrojoje architektūroje buvo pridėtas 1 konvoliucinis sluoksnis. Pirmajame konvoliuciniame sluoksnyje filtrų skaičius buvo 128, o antrajame 64. Sujungimo sluoksnis buvo po antrojo konvoliucinio sluoksnio. Buvo pasirinkta daugiau sujungimo sluoksnių nedėti, kadangi pastebėta, kad didinant jų skaičių, per stipirai pablogėja tikslumo rezultatai, taip gali nutikti dėl „Max Pooling“ metu prarandamos dalies informacijos.

Trečiojoje architektūroje buvo pridėtas dar 1 konvoliucinis sluoksnis (iš viso jų buvo 3). Pirmajame konvoliuciniame sluoksnyje filtrų skaičius buvo 128, antrajame 64, trečiajame 32. Sujungimo sluoksnis buvo naudotas po trečiojo konvoliucinio sluoksnio. Kaip ir anksčiau, daugiau sujungimo sluoksnių nebuvo pridėta.

Gauti rezultatai vaizduojami 1 paveikslėlyje. Galima pastebėti, kad geriausi rezultatai gauti su antrąją ir trečiąja architektūromis. Su antrąja architektūra geresniai rezultatai gauti mokymo aibėje, o su trečiąja validavimo. Taigi pasirenkama geriausia laikyti antrąją architektūrą.



1 pav.: tikslumo ir paklaidos rezultatai su skirtingomis architektūromis.

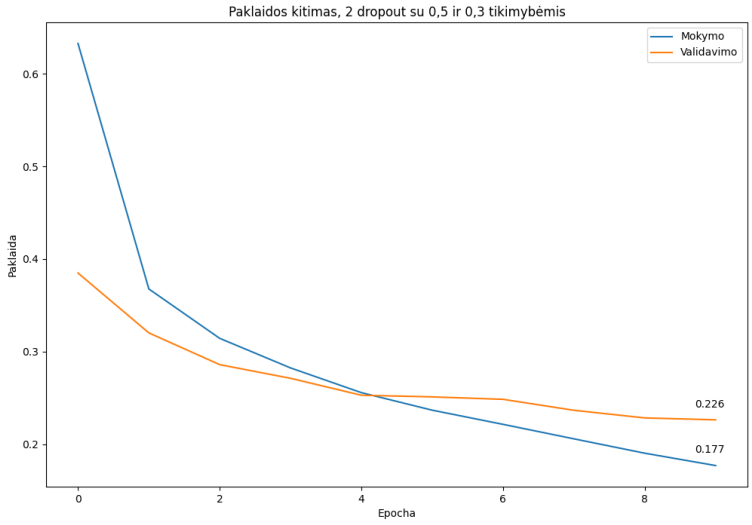
3.4 Rezultatų priklausomybė nuo išmetimo sluoksnių

Antrojoje tyrimo dalyje tiriama, kaip keičiasi klasifikavimo tikslumas ir paklaida su skirtingu išmetimo sluoksnių skaičiumi ir skirtingomis išmetimo tikimybėmis. Pirmiausia po pirmojo „Dense“ sluoksnio pridedamas 1 išmetimo sluoksnis su tikimybe 0,5. Toliau tiriama pridedant antrą išmetimo sluoksnį (po pirmojo konvoliucinio sluoksnio) su išmetimo tikimybe lygia 0,3 ir vėliau pakartojama su tikimybe lygia 0,6. Tada po 2 konvoliucinio sluoksnio (po sujungimo sluoksnio) pridedamas trečias išmetimo sluoksnis su tikimybe 0,3 ir vėlgi pakartojama su tikimybe lygia 0,6. Gauti rezultatai vaizduojami 1 lentelėje. Matoma, kad geriausi rezultatai buvo su 1 ir 2 išmetimo sluoksniais bei tikimybėmis 0,5 ir 0,5 bei 0,3 atitinkamai. Su 1 išmetimo sluoksniu gaunamas geresnis tikslumas su mokymo duomenimis, o su 2 išmetimo sluoksniais - validavimo. Taigi pasirenkama toliau atlikti tyrimus su 2 išmetimo sluoksniais ir išmetimo tikimybėmis lygiomis 0,3 ir 0,5.

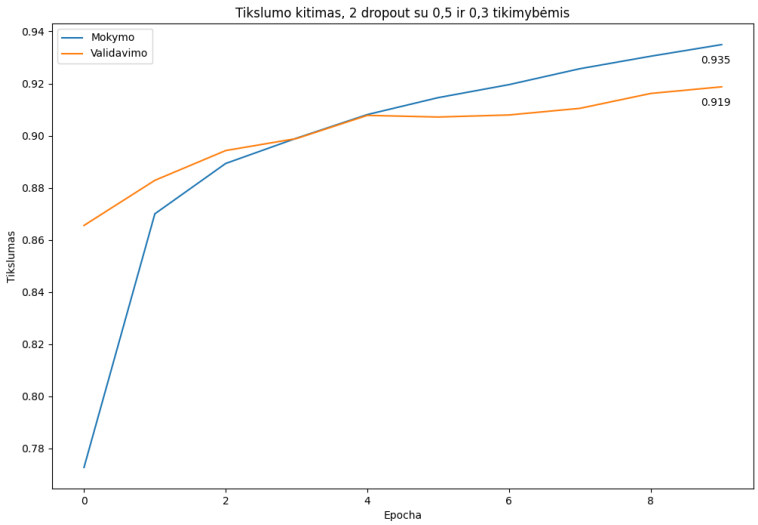
1 lentelė: rezultatai su skirtingomis tikimybėmis ir skirtingu išmetimo sluoksnių skaičiumi.

Išmetimo sluoksnių skaičius	Išmetimo tikimybė	Duomenys	Paklaida	Tikslumas
1	0,5	Mokymo	0,167	0,938
		Validavimo	0,23	0,914
2	0,3 ir 0,5	Mokymo	**0,177**	**0,935**
		Validavimo	**0,226**	**0,919**
	0,6 ir 0,5	Mokymo	0,205	0,924
		Validavimo	0,236	0,916
3	0,3, 0,3 ir 0,5	Mokymo	0,2	0,927
		Validavimo	0,229	0,919
	0,6, 0,6 ir 0,5	Mokymo	0,242	0,915
		Validavimo	0,259	0,904

Geriausių rezultatų tikslumas ir paklaida atitinkamai vaizduojami 2 ir 3 paveikslėliuose.



2 pav.: paklaida su 2 išmetimo sluoksniais bei 0,3 ir 0,5 išmetimo tikimybėmis



3 pav.: tikslumas su 2 išmetimo sluoksniais bei 0,3 ir 0,5 išmetimo tikimybėmis

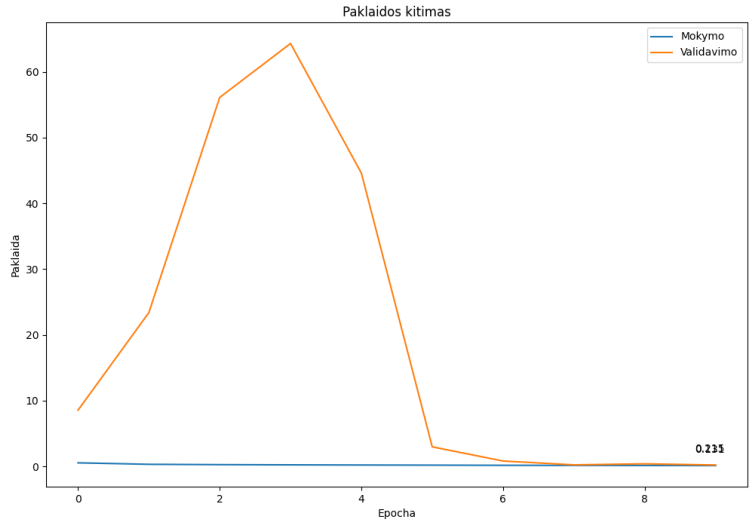
3.5 Rezultatų priklausomybė nuo paketų normalizavimo

Trečiojoje tyrimo dalyje buvo tiriama, kaip klasifikavimo tikslumui ir paklaidai įtaką daro paketų normalizavimas. Pirmiausia buvo tikrinama po pirmojo konvoliucinio sluoksnio pridedant paketų normalizavimą. Vėliau tyrimas buvo kartotas papildant paketų normalizavimą po antrojo konvoliucinio sluoksnio. Galiausiai buvo tikrinama pridedant trečiąjį paketų normalizavimo sluoksnį po pirmojo „Dense“ sluoksnio. Gauti rezultatai vaizduojami 2 lentelėje. Galima pastebėti, kad geriausi rezultatai gaunami su 3 normalizavimo sluoksniais. Lyginant rezultatus tarp be paketų normalizavimo (3.4 skyrelio rezultatai) ir su, gaunama, kad naudojant 3 normalizavimo sluoksnius rezultatai pagerėja. Be paketų normalizavimo buvo gautas mokymo duomenų tikslumas 0,935, o validavimo 0,919, tačiau su paketų normalizavimo 3 sluoksniais mokymo duomenims tikslumas lygus 0,9527, o validavimo 0,9268.

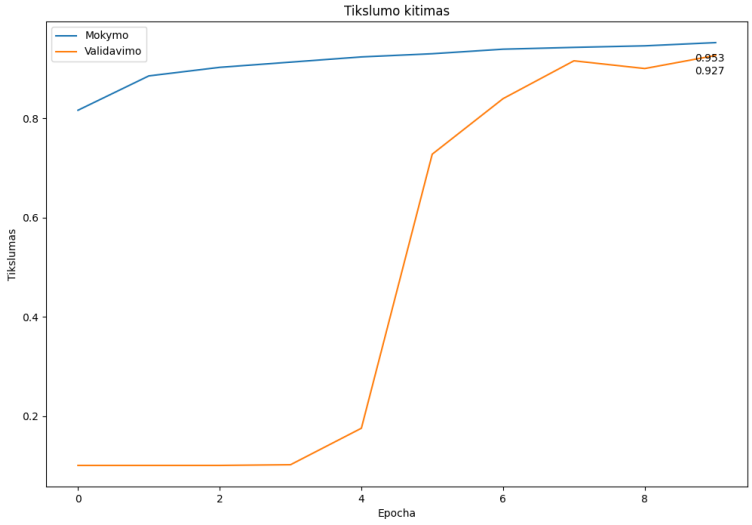
2 lentelė: tikslumo ir paklaidos priklausomybė nuo paketų normalizavimo sluoksnių skaičiaus.

Paketų normalizavimo sluoksnių skaičius	Duomenys	Paklaida	Tikslumas
1	Mokymo	0,282	0,8957
	Validavimo	0,2454	0,9116
2	Mokymo	0,2346	0,9132
	Validavimo	0,2562	0,9133
3	Mokymo	**0,1311**	**0,9527**
	Validavimo	**0,2154**	**0,9268**

5 ir 4 paveikslėliuose vaizduojamas paklaidos ir tikslumo kitimas pagal epochas su 3 normalizavimo sluoksniais.



4 pav.: paklaidos kitimas su 3 paketų normalizavimo sluoksniais.



5 pav.: tikslumo kitimas su 3 paketų normalizavimo sluoksniais.

3.6 Rezultatų priklausomybė nuo aktyvacijos funkcijos

Ketvirtojoje tyrimo dalyje buvo tiriama, kaip naudojama aktyvacijos funkcija daro įtaką paklaidai ir tikslumui mokymo ir validavimo duomenims. Palyginimui naudotos „Sigmoid“, „ReLU“, „tanh“, „Leaky ReLU“ ir „ELU“ aktyvacijos funkcijos. Gauti rezultatai vaizduojami 3 lentelėje. Iš lentelės galima pamatyti, kad prasčiausi rezultatai gaunami su „tanh“ funkcija, o geriausi rezultatai buvo gauti su „ReLU“ funkcija. Taigi tolesnėje tyrimo dalyje bus naudojama „ReLU“.

3 lentelė: tikslumo ir paklaidos priklausomybė nuo naudojamos aktyvacijos funkcijos.

Aktyvacijos funkcija	Duomenys	Paklaida	Tikslumas
Sigmoid	Mokymo	0,2365	0,9148
	Validavimo	0,3094	0,8921
ReLU	Mokymo	**0,1311**	**0,9527**
	Validavimo	**0,2154**	**0,9268**
tanh	Mokymo	0,3145	0,888
	Validavimo	0,7542	0,7968
Leaky ReLU	Mokymo	0,1634	0,9406
	Validavimo	0,2605	0,9122
ELU	Mokymo	0,1792	0,9344
	Validavimo	0,265	0,9095

3.7 Rezultatų priklausomybė nuo optimizavimo algoritmo

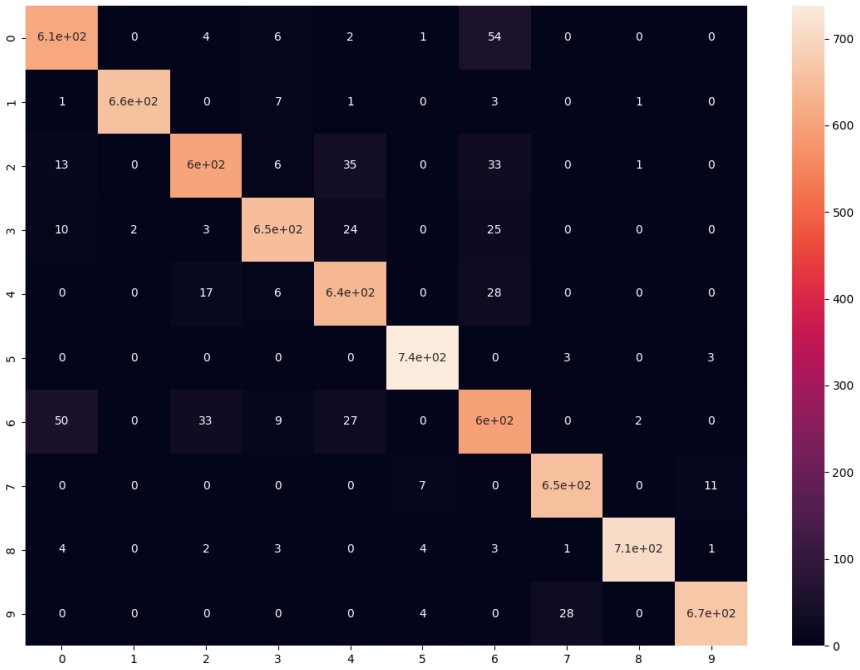
Penktojoje tyrimo dalyje buvo tirta, kaip naudojamas optimizavimo algoritmas daro įtaką tikslumui ir paklaidai mokymo ir validavimo duomenims. Palyginimui naudojami „Adam“„AdaGrad“, „RMSProp“ ir „stochastinis“ optimizavimo algoritmai. Gauti rezultatai vaizduojami 4 lentelėje. Rezultatai rodo, jog iš tirtųjų geriausi algoritmai buvo „Adam“ ir „RMSProp“, o blogiausias „AdaGrad“. Abiejų geriausiųjų algoritmų rezultatai buvo panašūs, tačiau „Adam“ algoritmas gavo geresnius rezultatus su validavimo duomenimis, o „RMSProp“ su mokymo. Taigi pagal gautus rezultatus pasirenkama toliau naudoti „Adam“ algoritmą.

4 lentelė: tikslumo ir paklaidos priklausomybė nuo optimizavimo algoritmo.

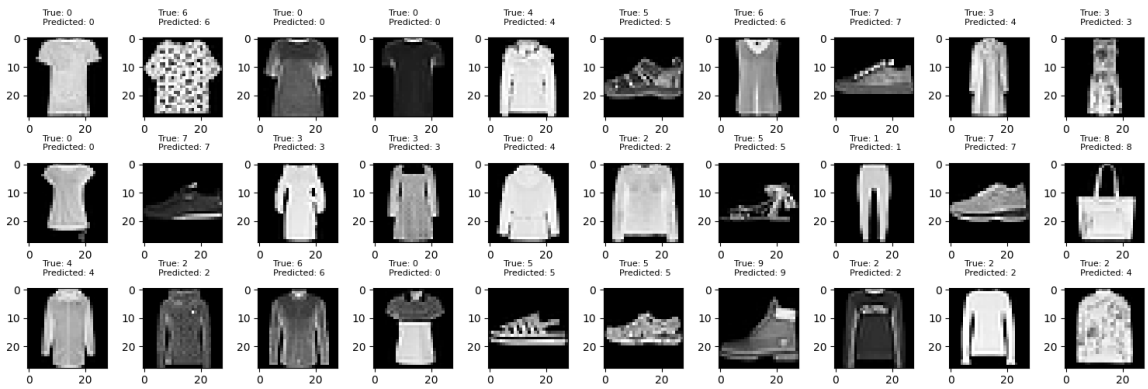
Optimizavimo algoritmas	Duomenys	Paklaida	Tikslumas
Adam	Mokymo	**0,1311**	**0,9527**
	Validavimo	**0,2154**	**0,9268**
RMSProp	Mokymo	0,1076	0,9613
	Validavimo	0,2322	0,9181
AdaGrad	Mokymo	0,439	0,8472
	Validavimo	0,3824	0,8673
Stochastinis	Mokymo	0,3366	0,8803
	Validavimo	0,336	0,8784

3.8 Testavimo duomenų rezultatai

Atlikus tyrimą buvo pastebėta, kad geriausi tikslumo ir paklaidos rezultatai gaunami su 3.3 skyrelyje aprašyta 2 neuroninio tinklo architektūra, 2 išmetimo sluoksniais (su tikimybėmis 0,3 ir 0,5), 3 paketų normalizavimo sluoksniais, „ReLU“ aktyvacijos funkcija bei „Adam“ optimizavimo algoritmu. Su šiais parametrais patikrinus testavimo duomenis buvo gauti tokie rezultatai: tikslumas 0,932 ir paklaida 0,195. Gauta klasifikavimo matrica vaizduojama 6 paveikslėlyje. Pirmi 30 testavimo įrašų su priskirtomis ir tikrosiomis klasėmis matomos 7 paveikslėlyje. Iš paimtų pirmų 30 duomenų įrašų matoma, kad priskiriant klases buvo suklysta 3 kartus.



6 pav.: klasifikavimo matrica testavimo duomenims.



7 pav.: pirmi 30 testavimo duomenų įrašų vaizdų.

4 Išvados

Taigi šio darbo metu buvo išsiaiškinta, su kokia architektūra ir kokiais hiperparametrais gaunamas geriausias tikslumas „Fashion MNIST“ duomenims. Tyrimas parodė, kad su didesniu konvoliucinių sluoksnių skaičiumi gaunamas geresnis tikslumas (šiuo atveju tyrimo rezultatai su 2 ar 3 konvoliucijomis skyrėsi nedaug). Tyrime buvo matoma, kaip geriausi rezultatai gauti su 1 arba 2 išmetimo sluoksniais. Su 1 išmetimo sluoksniu geresniai rezultatai buvo gaunami mokymo aibėje, o su 2 sluoksniais tiksliau buvo gauta su validavimo duomenimis. Taip pat tyrimo metu buvo matoma, kaip su didesnėmis išmetimo tikimybėmis tikslumas suprastėja ir mokymo, ir validavimo duomenims. Be to, atlikto darbo metu pastebėta, jog kuo daugiau normalizavimo sluoksnių buvo dedama, tuo tikslumas buvo gaunamas didesnis. Ištyrus 5 aktyvacijos funkcijas buvo gauta, kad geriausi rezultatai gauti su „ReLU“, o blogiausi su „tanh“. Ištyrus rezultatų priklausomybę nuo optimizavimo algoritmo su 4 optimizavimo algoritmais, buvo gauta, jog su „Adam“ ir „RMSProp“ buvo gautas geriausias tikslumas. „Adam“ atveju geresniai rezultatai gaunami validavimo aibei, o „RMSProp“ atveju mokymo. Naudojant geriausius hiperparametrus ir architektūrą testavimo duomenims buvo gauti pakankamai geri rezultatai (tikslumas lygus 0,932 ir paklaida 0,195).

5 Priedai

Programos kodas:

```
import tensorflow as tf
import pandas as pd
import numpy as np
from tensorflow import keras
from matplotlib import pyplot
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn import metrics
import matplotlib.pyplot as plt
from scipy.io import arff
from io import StringIO
import pandas as pd
import random

#Nuskaitomi duomenys (pradiniai testavimo ir mokymo duomenys buvo sujungti į vieną)
data =  pd.read_csv('combined_file.csv', sep=",")
duom_dydis = len(data)
#Atsitiktinai atrenkami duomenys testavimui, iš viso bus atrenkama 10% bendrų duomenų
selected_indices1 = random.sample(range(duom_dydis), round(duom_dydis * 0.1))
#Atrinkti duomenys priskiriami testavimo aibei
test_data = data.iloc[selected_indices1]
#Panaikinami testavimo duomenys iš pradinės duomenų aibės
data.drop(selected_indices1, inplace=True)
data.reset_index(drop=True, inplace=True)

#Atsitiktinai atrenkami duomenys validavimui, iš viso bus atrenkama 10% bendrų duomenų
selected_indices2 = random.sample(range(len(data)), round(duom_dydis * 0.1))

#Atrinkti duomenys priskiriami validavimo aibei
valid_data = data.iloc[selected_indices2]
valid_data.reset_index(drop=True, inplace=True)
test_data.reset_index(drop=True, inplace=True)
#Panaikinami validavimo duomenys iš pradinės aibės, pradinė aibė bus mokymo duomenys
data.drop(selected_indices2, inplace=True)
data.reset_index(drop=True, inplace=True)

train_file = 'train_data1.csv'
test_file = 'test_data1.csv'
```

```
valid_file = 'validation_data1.csv'

#Gautos aibės įrašomos į failus
data.to_csv(train_file)
test_data.to_csv(test_file)
valid_data.to_csv(valid_file)

#Nuskaitomi duomenys
train_x = pd.read_csv("train_data1.csv")
test_x = pd.read_csv("test_data1.csv")
valid_x = pd.read_csv("validation_data1.csv")

# Paimamos duomenų klasių reikšmės
train_y1 = train_x.iloc[:, 1]
test_y1 = test_x.iloc[:, 1]
valid_y1 = valid_x.iloc[:, 1]

# Panaikinami nereikalingi stulpeliai
train_x = train_x.drop(train_x.columns[1], axis=1)
test_x = test_x.drop(test_x.columns[1], axis=1)
valid_x = valid_x.drop(valid_x.columns[1], axis=1)

train_x = train_x.drop(train_x.columns[0], axis=1)
test_x = test_x.drop(test_x.columns[0], axis=1)
valid_x = valid_x.drop(valid_x.columns[0], axis=1)

# Patikrinama ar nėra tuščių reikšmių
print(train_x.isnull().any().sum())
print(test_x.isnull().any().sum())
print(valid_x.isnull().any().sum())

# Normalizavimas
train_x = train_x / 255.0
test_x = test_x / 255.0
valid_x = valid_x / 255.0


# Duomenys paruošiami neuroniniam tinklui mokyti tinkamu formatu (28x28 matrica)
train_x = train_x.to_numpy().reshape((train_x.shape[0], 28, 28, 1))
test_x = test_x.to_numpy().reshape((test_x.shape[0], 28, 28, 1))
valid_x = valid_x.to_numpy().reshape((valid_x.shape[0], 28, 28, 1))
```

```
# Tikrosios klasių reikšmės paruošiamos mokymui
train_y = keras.utils.to_categorical(train_y1)
test_y = keras.utils.to_categorical(test_y1)
valid_y = keras.utils.to_categorical(valid_y1)

# Neuroninio tinklo modelis
model2 = tf.keras.Sequential([
    # Konvoliucijos sluoksnis
    tf.keras.layers.Conv2D(filters=128, kernel_size=2, activation="relu",
    input_shape=(28, 28, 1)),
    # Normalizavimas
    tf.keras.layers.BatchNormalization(),
    # Išmetimo sluoksnis. Mokymo metu kiekvienoje iteracijoje laikinai išmetami dalis neuronų.
    tf.keras.layers.Dropout(0.3),

    tf.keras.layers.Conv2D(filters=64, kernel_size=2, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    # Atliekamas pooling'as
    tf.keras.layers.MaxPooling2D(pool_size=2),

    # Šiame sluoksnyje dvimačiai masyvai paverčiami į vienmatį masyvą
    tf.keras.layers.Flatten(),
    #Pilnai sujungtas sluoksnis
    tf.keras.layers.Dense(256, activation="relu"),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    # Gaunamos kiekvienos klasės tikimybės
    tf.keras.layers.Dense(10, activation='softmax')
])
# Sukompiliuojamas neuroninio tinklo modelis
model2.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history2= model2.fit(train_x, train_y, batch_size=512, epochs=10, verbose=1,
validation_data=(valid_x, valid_y))

# Įvertinami testavimo aibės rezultatai
evaluation2 = model2.evaluate(test_x, test_y)
print(f'Test accuracy : {evaluation2[1]:.3f}')
print(f'Test loss : {evaluation2[0]:.3f}')

#Tikrosios klasės
```

```
test_y_arg = np.argmax(test_y, axis = 1)
#Nuspėtos klasės
predicted_classes = np.argmax(model2.predict(test_x), axis = 1)

# Klasifikavimo matrica
cmatrix = confusion_matrix(test_y_arg, predicted_classes)
plt.figure(figsize = (14,10))
sns.heatmap(cmatrix, annot=True)
plt.savefig('confusion_matrix.png')

# Vaizduojamos pirmos 30 testavimo duomenų reikšmių
fig = pyplot.figure(figsize=(15, 5))
for i in range(30):
    ax = pyplot.subplot(3, 10, i + 1)
    ax.text(5, -5, f'True: {test_y_arg[i]}\nPredicted: {predicted_classes[i]}',
            color='black', fontsize=8)
    ax.imshow(test_x[i], cmap=pyplot.get_cmap('gray'))

pyplot.tight_layout()
pyplot.savefig("fashion_data.png")

# Tikslumo ir paklaidos grafikai
plt.figure(figsize=(12, 8))
plt.plot(history2.history['loss'], label='Loss')
plt.plot(history2.history['val_loss'], label='val_Loss')
plt.xlabel('Epocha')
plt.ylabel('Paklaida')
plt.legend(['Mokymo', 'Validavimo'])
plt.title('Paklaidos kitimas')

# Paskutinės paklaidos vaizdavimas
last_epoch2 = len(history2.history['loss']) - 1
plt.annotate(
    f'{history2.history["loss"][-1]:.3f}',
    xy=(last_epoch2, history2.history['loss'][-1]),
    xytext=(-20, 20),
    textcoords='offset points',
    ha='left',
    va='top'
)
plt.annotate(
```

```
    f'{history2.history["val_loss"][-1]:.3f}',
    xy=(last_epoch2, history2.history['val_loss'][-1]),
    xytext=(-20, 20),
    textcoords='offset points',
    ha='left',
    va='top'
)
plt.savefig('paklaida.png')

plt.figure(figsize=(12, 8))
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title("Tikslumo kitimas")
plt.xlabel('Epocha')
plt.ylabel('Tikslumas')
plt.legend(['Mokymo', 'Validavimo'])

# Paskutinio tikslumo vaizdavimas
plt.annotate(
    f'{history2.history["accuracy"][-1]:.3f}',
    xy=(last_epoch2, history2.history['accuracy'][-1]),
    xytext=(-20, -10),
    textcoords='offset points',
    ha='left',
    va='top'
)

plt.annotate(
    f'{history2.history["val_accuracy"][-1]:.3f}',
    xy=(last_epoch2, history2.history['val_accuracy'][-1]),
    xytext=(-20, -10),
    textcoords='offset points',
    ha='left',
    va='top'
)
plt.savefig('tikslumas.png')
```

6 Šaltiniai

https://github.com/VictorOwinoKe/CNN-Image-classification-Using-Fashion-MNIST-dataset/blob/master/CNN%20Image%20classifocationn%20python%20%20Notebook.ipynb