VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMOS

Skaitmeninis intelektas ir sprendimų priėmimas

Vaizdų klasifikavimas naudojant konvoliucinius neuroninius tinklus

Darbą atliko:
Pijus Zlatkus

Vilnius
2024

Turinys

1. Užduoties tikslas 3
2. Naudojami klasifikavimo duomenys 3
3. Duomenų paruošimas apmokymui 3
4. Konvoliucinio neuroninio tinklo ir hiperparametrų realizacija 5
4.1. Naudojamo įrenginio nustatymas (GPU ir TPU) 5
4.2. Konvoliucinio neuroninio tinklo modeliai 7
4.3. Hiperparametrai 9
5. Atlikti tyrimai 10
6. Išvados 16

1. Užduoties tikslas

Užduoties tikslas – apmokyti konvoliucinį neuroninį tinklą vaizdams klasifikuoti ir atlikti tyrimą su skirtingais hiperparametrais..

2. Naudojami klasifikavimo duomenys

Tyrimui atlikti buvo panaudotas „Fashion MNIST“ duomenų rinkinys su 28x28 pikselių dydžio nespalvotomis nuotraukomis. Šį duomenų rinkinį sudaro 10 klasių įvairių drabužių nuotraukų: **0** - marškinėliai/palaidinė, **1** – Kelnės, **2** – Megztinis, **3** – Suknelė, **4** – Paltas, **5** – Sandalas, **6** – Marškiniai, **7** – Sportbatis, **8** – Krepšys, **9** - Batai iki kulkšnies.

Visą duomenų rinkinį sudaro 60000 mokymui skirtų pavyzdžių ir 10000 testavimui skirtų pavyzdžių. Kad būtų paprasčiau, šiam duomenų rinkiniui atsisiųsti buvo panaudota TensorFlow biblioteka, kuri turi reikalingas operacija atsisiųsti ir užkrauti kai kuriuos duomenų rinkinius, įskaitant ir Fashion MNIST.

3. Duomenų paruošimas apmokymui

Kokybiškam modelio apmokymui įprastai naudojamos trys duomenų rinkinių rūšys: mokymo, validacijos ir testavimo. Kadangi šis duomenų rinkinys turi tik dvi – mokymo ir testavimo, tai jį reikia pertvarkyti. Pirmiausiai Fashion MNIST mokymo ir testavimo rinkiniai yra sujungiami ir iš sujungto duomenų rinkinio yra suskaidoma į mokymo, validacijos ir testavimo duomenų rinkinius santykiu 80:10:10.

```python
# Fashion MNIST duomenų rinkinio užkrovimas
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion_mnist.load_data()

# Sujungiami mokymo ir testavimo duomenų rinkiniai
combined_images = np.concatenate((train_images, test_images))
combined_labels = np.concatenate((train_labels, test_labels))

# Nustatomas klasių numeravimas nuo 0 iki 9
combined_labels = to_categorical(combined_labels, num_classes=10)

# Normalizuojami paveikslėliai, kad jie turėtų reikšmes tarp 0 ir 1
combined_images = combined_images.astype('float32') / 255

# Išskaidomas sujungtas duomenų rinkinys į naujus mokymo, validavimo ir testavimo rinkinius
train_images, temp_images, train_labels, temp_labels = train_test_split(
    combined_images, combined_labels, test_size=0.2, random_state=42)

val_images, test_images, val_labels, test_labels = train_test_split(
    temp_images, temp_labels, test_size=0.5, random_state=42)

# Mokymo, validavimo ir testavimo rinkinių paveikslėlių formos ir dydžiai
print("Training set images shape:", train_images.shape)
print("Validation set images shape:", val_images.shape)
print("Test set images shape:", test_images.shape)
print("Training set labels shape:", train_labels.shape)
print("Validation set labels shape:", val_labels.shape)
print("Test set labels shape:", test_labels.shape)
```

✓ 3.4s

```
Training set images shape: (56000, 28, 28)
Validation set images shape: (7000, 28, 28)
Test set images shape: (7000, 28, 28)
Training set labels shape: (56000, 10)
Validation set labels shape: (7000, 10)
Test set labels shape: (7000, 10)
```

1 pav. *Duomenų rinkinio Fashion MNIST užkrovimas ir suskaidymas naudojant TensorFlow ir Keras*

Pagal **1 pav.** Duomenų rinkinio Fashion MNIST užkrovimas ir suskaidymas naudojant TensorFlow ir Keras esantį programos kodą galima pastebėti, kad visas duomenų rinkinys yra ištraukiamas su viena komanda. Po to eina visi anksčiau išvardinti veiksmai ir gale yra atspausdinamos duomenų rinkinių ir jų klasių žymų formos. Prie paveikslėlių forma (56000, 28, 28) reiškia, kad yra 28x28 pikselių dydžio 56000 paveikslėliai. Prie klasių žymų, (56000, 10) reiškia, kad yra iš viso sužymėta 56000 paveikslėliai su 10 klasių. Čia būtina pabrėžti, kad klasės nėra žymimos nuo 0 iki 9 kaip išvardinta anksčiau, bet yra sužymėtos naudojant „one-hot“ koduotę, kai tikroji klasė yra priskiriama 1, o kitos likusios – 0. Todėl prie kiekvieno paveikslėlio žymų yra 10 skaičių.

```python
batch_size = 32

# Pakeičiame paveikslėlių formą į (32, 28, 28, 1). 32 - batch dydis, 28x28 - paveikslėlio dydis, 1 - spalvų kanalų skaičius
train_images = train_images.reshape((-1, 28, 28, 1))
val_images = val_images.reshape((-1, 28, 28, 1))
test_images = test_images.reshape((-1, 28, 28, 1))

train_dataset = tf.data.Dataset.from_tensor_slices((train_images, train_labels))
train_dataset = train_dataset.shuffle(train_labels.size).batch(batch_size).prefetch(tf.data.AUTOTUNE)

val_dataset = tf.data.Dataset.from_tensor_slices((val_images, val_labels))
val_dataset = val_dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)

test_dataset = tf.data.Dataset.from_tensor_slices((test_images, test_labels))
test_dataset = test_dataset.batch(batch_size).prefetch(tf.data.AUTOTUNE)

print("\nDatasets created successfully.")
for images, labels in train_dataset.take(1):
  print(f'Train batch images shape: {images.shape}')
  print(f'Train batch labels shape: {labels.shape}')

for images, labels in val_dataset.take(1):
  print(f'Validation batch images shape: {images.shape}')
  print(f'Validation batch labels shape: {labels.shape}')

for images, labels in test_dataset.take(1):
  print(f'Test batch images shape: {images.shape}')
  print(f'Test batch labels shape: {labels.shape}')
```

✓ 3.0s

```
Datasets created successfully.
Train batch images shape: (32, 28, 28, 1)
Train batch labels shape: (32, 10)
Validation batch images shape: (32, 28, 28, 1)
Validation batch labels shape: (32, 10)
Test batch images shape: (32, 28, 28, 1)
Test batch labels shape: (32, 10)
```

2 pav. *Duomenų rinkinių paruošimas modelio apmokymui*

Atlikus šiuos veiksmus, duomenys dar nėra galutinai paruošti modelio apmokymui. Kad būtų efektyviai apmokytas modelis, tam reikia visus duomenų rinkinius suskirstyti į mažesnius rinkinius (angl. batches) kaip pavaizduota **2 pav.** Duomenų rinkinių paruošimas modelio apmokymui. Vieno rinkinio dydis buvo pasirinktas 32. Pasiekti aukštesniam tikslumui, šie duomenys yra išmaišomi tarpusavyje ir greitesniam mokymui yra iš anksto užkraunami.

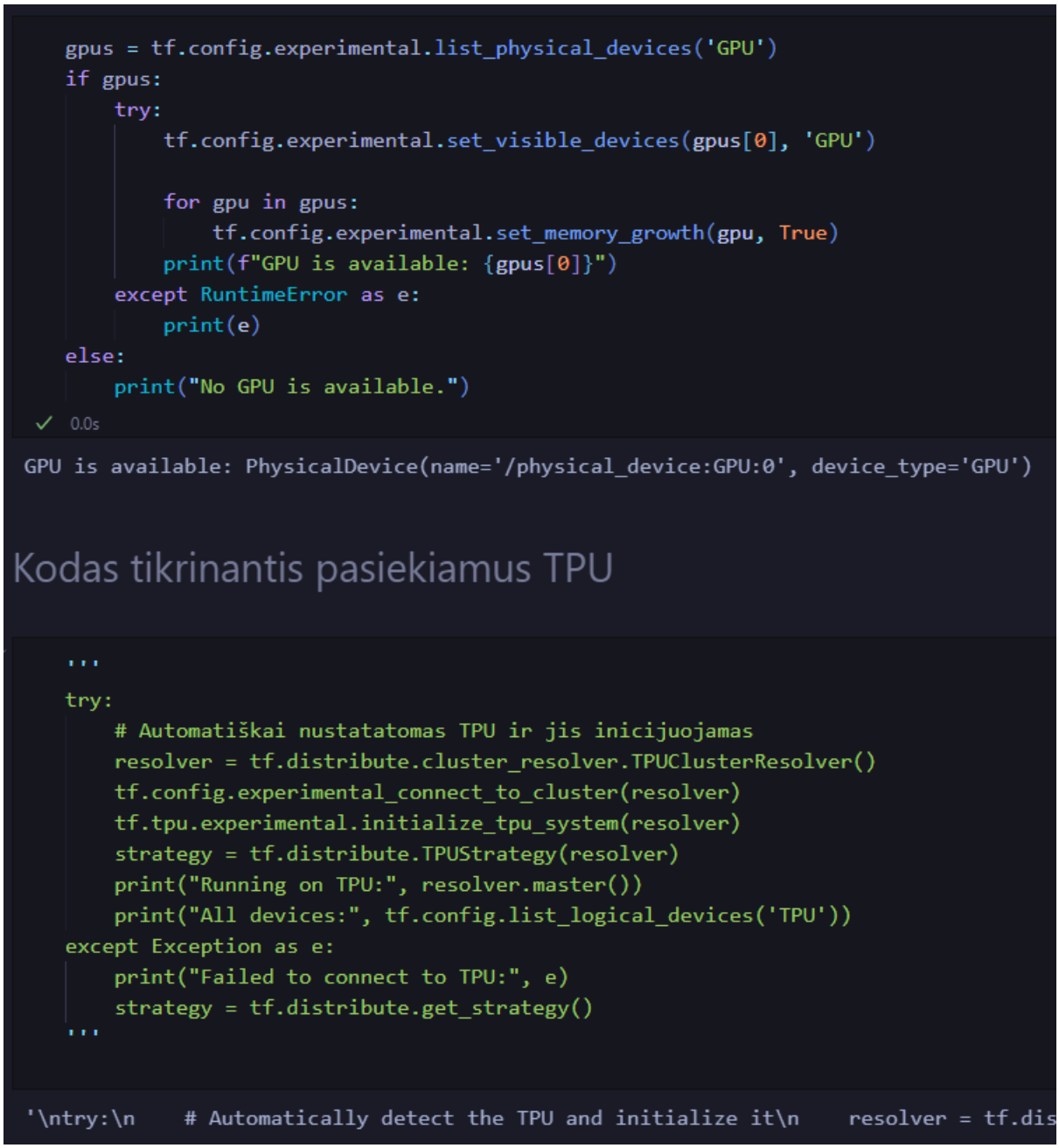
Dabar galima pastebėti, kad duomenų rinkinių formos pasikeitė. Pirmiausiai visur atsirado papildomas skaičius – 1, kuris reiškia, kad nuotraukos yra vieno kanalo (nespalvotos). Taip pat prie formos nurodoma, kad paveikslėliai iškart yra paimami kaip mažas rinkinys, kurio dydis 32. Dabar šie duomenys yra pilnai paruošti efektyviam modelio apmokymui.

4. Konvoliucinio neuroninio tinklo ir hiperparametrų realizacija

4.1. Naudojamo įrenginio nustatymas (GPU ir TPU)

Konvoliucinio neuroninio tinklo modeliui mokinti reikia nemažai skaičiavimo resursų, todėl naudoti kompiuterio procesorių šiam tikslui nėra efektyvu ir tai gali užtrukti daug laiko. Geriausia tai atlikti pasitelkiant kompiuterio grafinį procesorių (GPU) ar tenzorinį procesorių

(TPU). Dažniausiai GPU galima rasti asmeniniame kompiuteryje arba naudoti debesijose esančius procesorius, kaip Google Colab. TPU dažniausiai randami tik naudojantis debesijas. Šiai užduočiai atlikti buvo paruoštas kodas naudotis GPU ir TPU, kadangi dalis užduoties buvo atlikta naudojantis Google Colab ir dalis atlikta lokaliai asmeniniame kompiuteryje.



3 pav. *Naudojamo įrenginio nustatymas (GPU arba TPU)*

3 pav. Naudojamo įrenginio nustatymas (GPU arba TPU) esantis kodas būtent ir nustato, kurį įrenginį naudosime atlikti užduotį. Verta pastebėti, kad reikia vykdyti kode vieną iš esančių variantų, kad nenaudotume abiejų įrenginių ir nebūtų klaidų.

4.2. Konvoliucinio neuroninio tinklo modeliai

Atlikti tyrimus buvo realizuoti trys skirtingi konvoliucinio neuroninio tinklo modeliai. Jie buvo pavadinti „paprastu“, „tarpiniu“ ir „sudėtingu“ pavadinimais ir buvo pasiūlyti dirbtinio intelekto įrankio ChatGPT.

```
if architecture_type == 'basic':
  model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))

  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))

  model.add(Conv2D(128, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))

  model.add(Flatten())
  model.add(Dense(128, activation='relu'))
  model.add(Dense(10, activation='softmax'))
```

4 pav. *Pirmasis "paprastas" konvoliucinio neuroninio tinklo modelis*

```
elif architecture_type == 'intermediate':
  model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))

  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))

  model.add(Conv2D(128, (3, 3), activation='relu'))
  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'))
```

5 pav. *Antrasis "tarpinis" konvoliucinio neuroninio tinklo modelis*

```
elif architecture_type == 'advanced':
  model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
  model.add(Conv2D(32, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))

  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(Conv2D(64, (3, 3), activation='relu'))
  model.add(MaxPooling2D((2, 2)))
  model.add(Dropout(0.25))

  model.add(Conv2D(128, (3, 3), activation='relu'))
  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'))
```

6 pav. *Trečiasis "sudėtingas" konvoliucinio neuroninio tinklo modelis*

4 pav. Pirmasis "paprastas" konvoliucinio neuroninio tinklo modelis, **5 pav.** Antrasis "tarpinis" konvoliucinio neuroninio tinklo modelis ir **6 pav.** Trečiasis "sudėtingas" konvoliucinio neuroninio tinklo modelis paveikslėliuose parodytos visos konvoliucinio neuroninio tinklo modelių architektūros. Kiekviena eilutė reiškia neuroninio tinklo sluoksnį. Kadangi šiam tyrimui buvo naudojami konvoliuciniai neuroniniai tinklai, tai galima pastebėti, kad naudojami Conv2D sluoksniai, kurie atlieka konvoliucijos operacijas su nurodytu parametrų skaičiumi. Taip pat kartu naudojamos ir MaxPooling2D operacijos, kurios kartu su konvoliucijos operacijomis išryškina ir padeda įsiminti paveikslėlių klasių požymius. Antrame ir trečiame modeliuose taip pat yra naudojama ir Dropout funkcija, kuri išmeta nurodytą dalį parametrų reikšmių ir gali sumažinti nereikalingų požymių skaičių ir sumažinti modelio persimokymą. Tačiau šiuos sluoksnius ir atitinkamai jų parametrus reikia keisti atsargiai, nes jie gali stipriai pakeisti norimus gauti rezultatus.

Aktyvacijos funkcija buvo pasirinkta softmax funkcija, kuri gražina vieną iš klasių, kurios yra didžiausia prognozės reikšmė.

4.3. Hiperparametrai

Papildomai be modelio architektūros svarbu parinkti ir optimizavimo, bei nuostolių funkcijomis. Šioms trims architektūroms parinkau **kategorinę kryžminės entropijos funkciją** (angl. Categorical Cross Entropy). Ši funkciją pasirinkta dėl kelių priežaščių:

- Duomenų rinkinys yra užkoduotas „one-hot“ koduote su kuria ne visas nuostolių funkcijas galima naudoti.
- Architektūroje naudojama „softmax“ aktyvacijos funkcija.
- Šie modeliai skirti spręsti kelių klasių klasifikavimo uždavinį
- Ši nuostolių funkcija padeda optimizuoti svorius taip, kad sumažintų atotrūkį tarp prognozuotų ir tikrųjų klasių. Tai užtikrina, kad modelis mokosi tiksliai klasifikuoti kiekvieną klasę.

Optimizavimo funkcijomis buvo parinkti 3 variantai: Adam, SGD ir RMSprop. Atlikti tyrimą su skirtingomis mokymosi žingsnių reikšmėmis buvo naudota Adam funkcija. Tik atrinkus geriausią modelį ir mokymosi žingsnio reikšmę yra testuojamos kitos optimizavimo funkcijos.

```
if optimizer == 'adam':
  opt = Adam(learning_rate=learning_rate)
elif optimizer == 'sgd':
  opt = SGD(learning_rate=learning_rate)
elif optimizer == 'rmsprop':
  opt = RMSprop(learning_rate=learning_rate)

model.compile(
  optimizer=opt,
  loss=CategoricalCrossentropy(),
  metrics=['accuracy'],
)
return model
```

7 pav. *Modelio optimizavimo ir nuostolių funkcijos nustatymas*

Be **7 pav.** Modelio optimizavimo ir nuostolių funkcijos nustatymas pavaizduotų hiperparametrų buvo naudotos pagalbinės mokymo taisyklės. Viena iš jų yra ankstyvo sustabdymo taisyklė, kuri sustabdo mokimosi ciklą jam nepasiekus nustatyto epochų skaičiaus, kuris šiuo atveju yra lygus 100. Jeigu modelis nebesugeba toliau mokytis iš duomenų, jis yra

sustabdomas. Kita taisyklė yra mokymosi žingsnio mažinimas mokymosi metu. Jeigu modelis pradeda lėtai mokytis arba neišeina toliau mokytis iš tų pačių duomenų, prieš sustabdant ciklą bandoma mažinti mokymosi žingsnį, taip pasiekiant dar didesnį tikslumą.

```
early_stopping = EarlyStopping(monitor='val_loss', patience=6, verbose=1, mode='min', restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, verbose=1, mode='min', min_lr=1e-8)
```

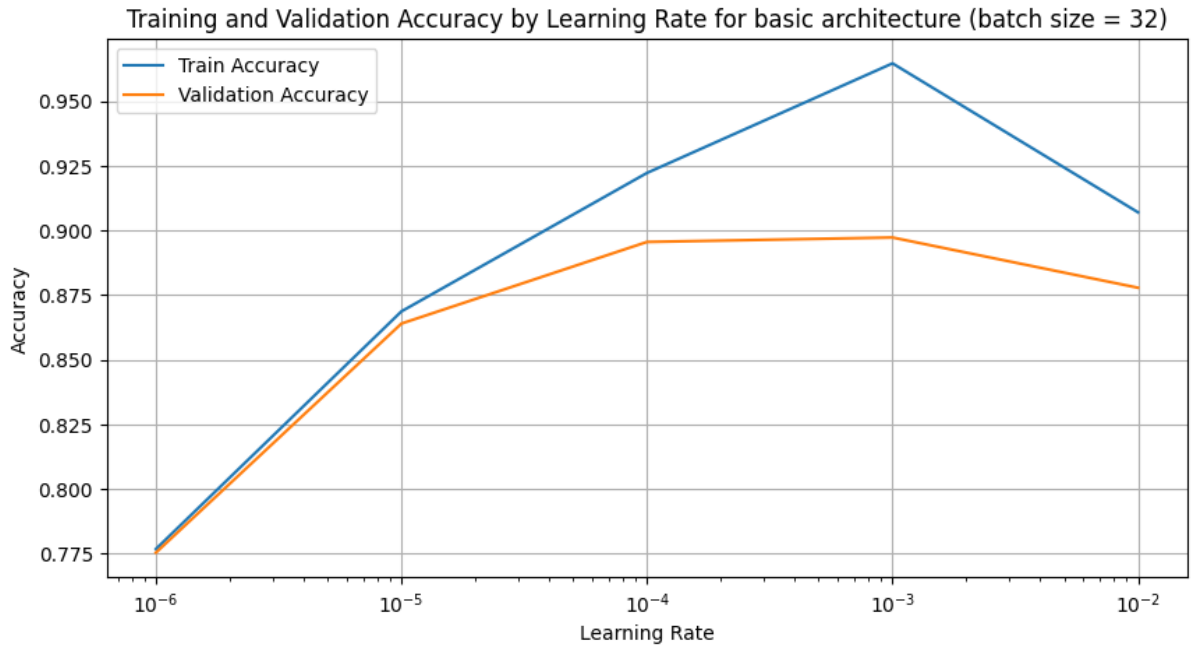
8 pav. *Ankstyvo sustabdymo ir mokymo žingsnio dydžio mažinimo panaudojimas*

Šios dvi taisyklės yra atvaizduotos **8 pav.** Ankstyvo sustabdymo ir mokymo žingsnio dydžio mažinimo panaudojimas, kurios yra jau realizuotos TensorFlow bibliotekoje.

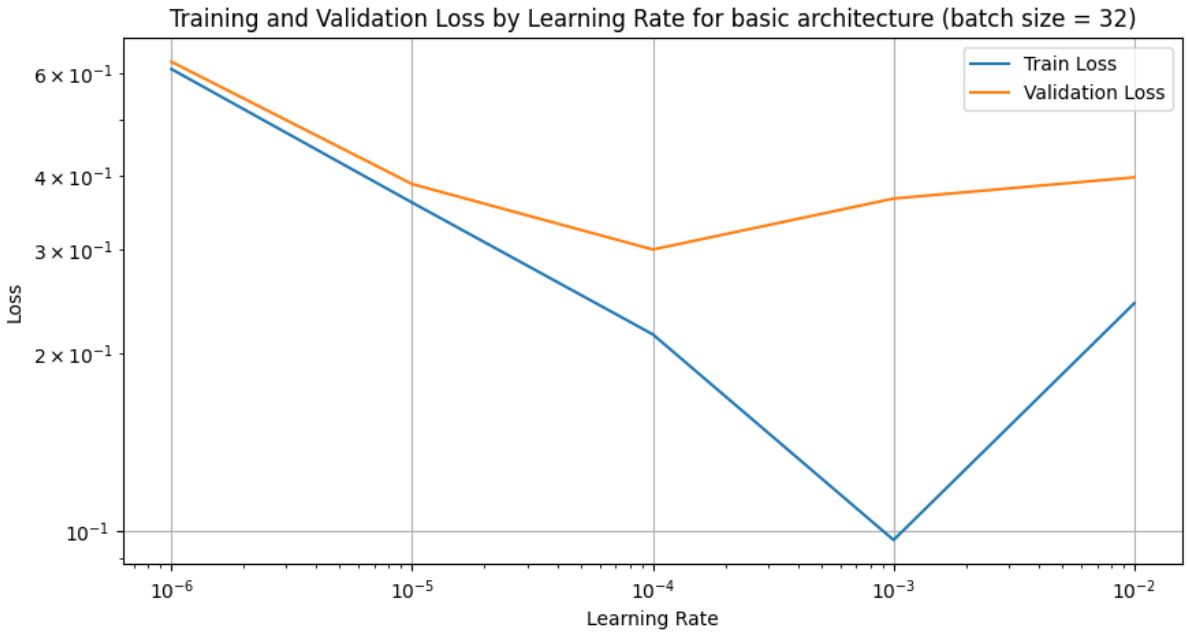
5. Atlikti tyrimai

Atrasti geriausiam modeliui buvo atliekami tyrimo 3 etapai. Pirmasis etapas su mokymosi ir validacijos duomenų rinkiniais buvo atrasti geriausią architektūrą iš pateiktų trijų naudojant Adam optimizavimo funkciją ir testuojant kiekvieną modelį su pradiniais 5 mokymosi žingsniais: 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3} ir 10^{-2}.

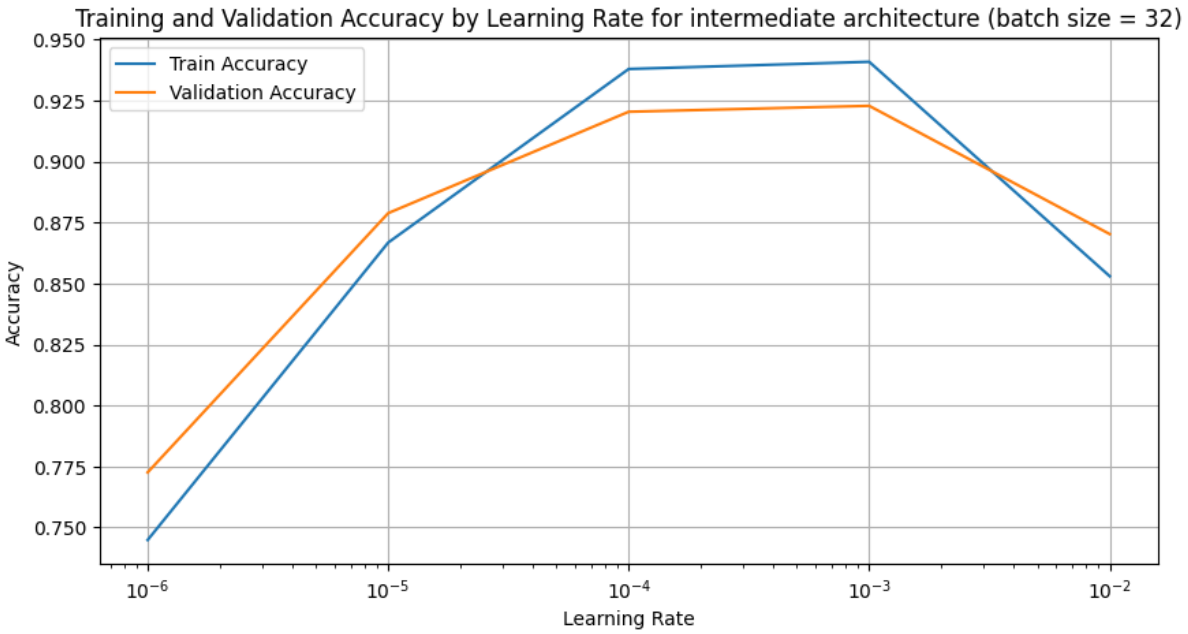
Iš viso buvo gautos 6 duomenų diagramos:



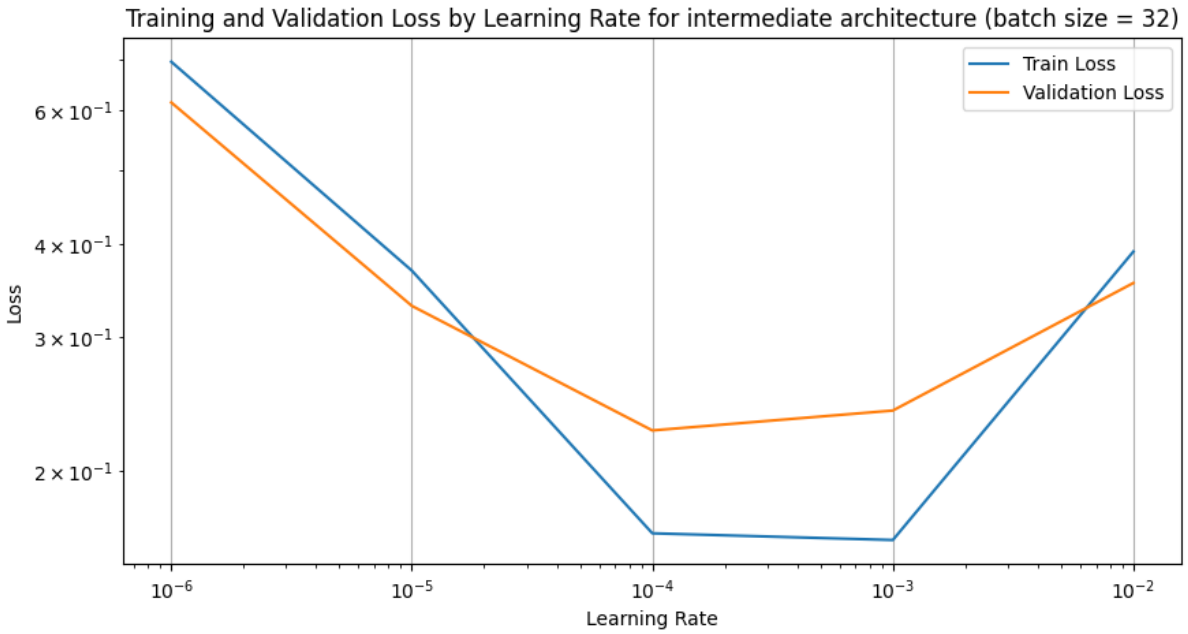
9 pav. *Mokymosi ir validacijos duomenų tikslumas nuo pradinio mokymosi žingsnio naudojant "paprastą" architektūrą*



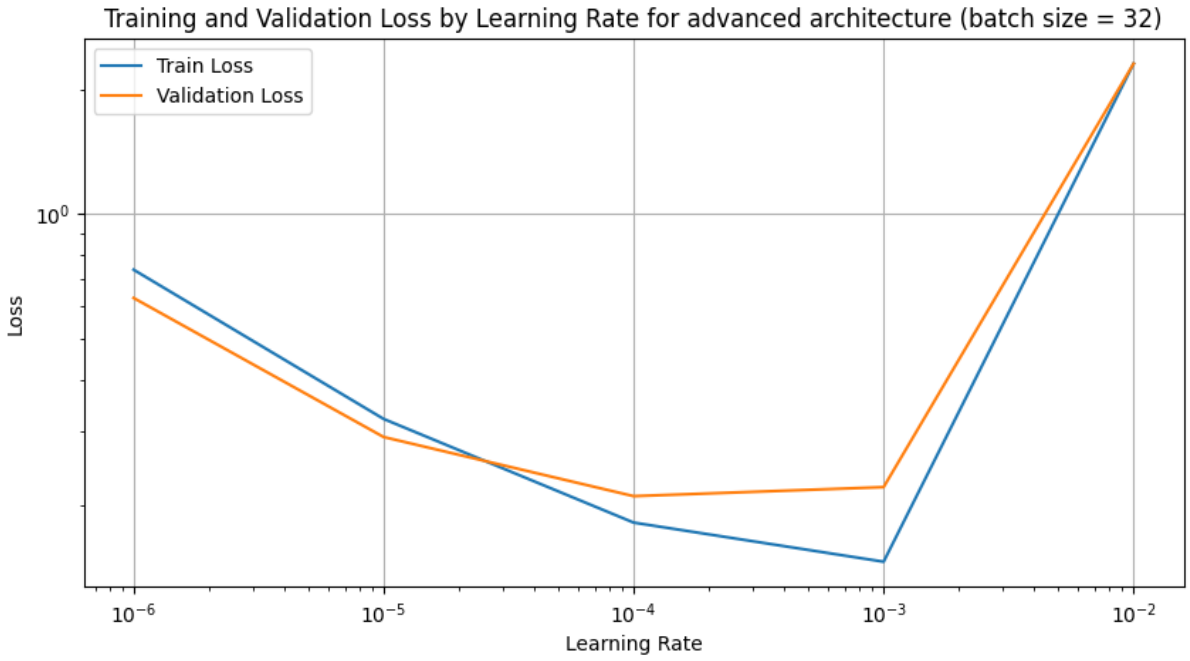
10 pav. *Mokymosi ir validacijos duomenų paklaida nuo pradinio mokymosi žingsnio naudojant "paprastą" architektūrą*



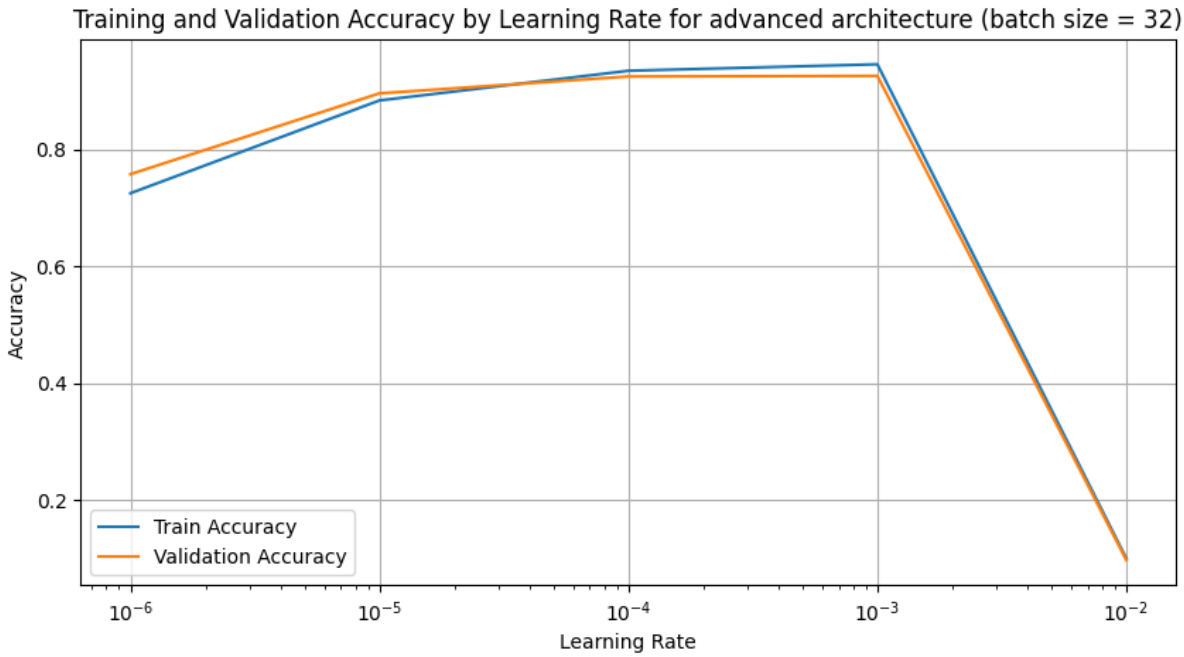
11 pav. *Mokymosi ir validacijos duomenų tikslumas nuo pradinio mokymosi žingsnio naudojant "tarpinę" architektūrą*



12 pav. *Mokymosi ir validacijos duomenų paklaida nuo pradinio mokymosi žingsnio naudojant "tarpinę" architektūrą*



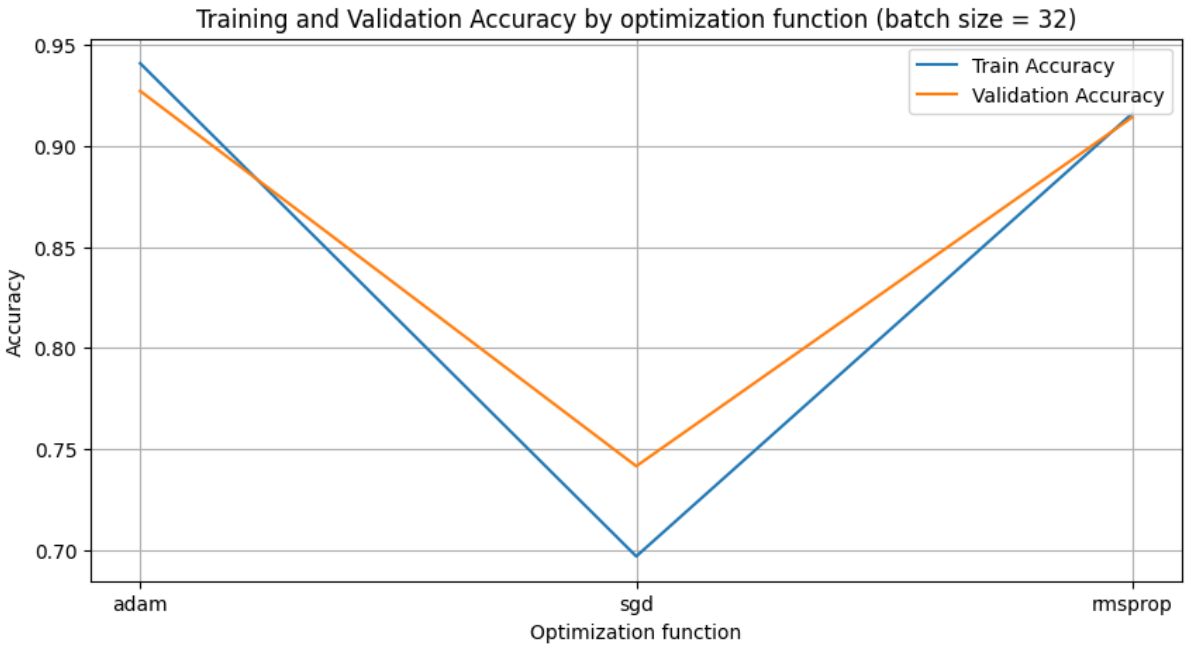
13 pav. *Mokymosi ir validacijos duomenų tikslumas nuo pradinio mokymosi žingsnio naudojant "sudėtingą" architektūrą*



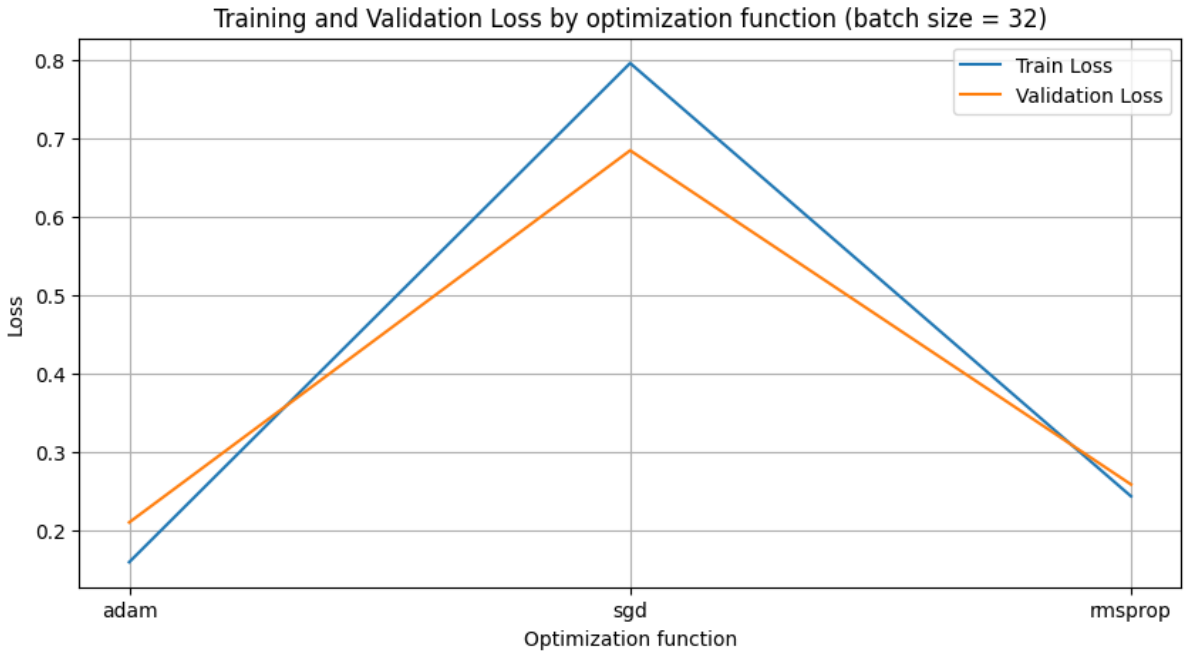
14 pav. *Mokymosi ir validacijos duomenų paklaida nuo pradinio mokymosi žingsnio naudojant "sudėtingą" architektūrą*

Pagal viršuje pateiktas diagramas galima pastebėti, kad geriausi rezultatai buvo nustačius 10^{-4} pradinę mokymosi žingsnio reikšmę. Tarp visų architektūrų, geriausi rezultatai buvo su „sudėtinga“ architektūra. Tačiau reikia turėti omenyje, kad nebuvo ištestuota dar didesni modeliai ir didesnis skaičius sluoksnių nebūtinai reiškia, kad modelis visada bus geresnis.

Antras tyrimo etapas buvo su rastu geriausiu modeliu ir pradine mokymosi žingsnio reikšme. Šį kartą buvo ta patį architektūra apmokyta ant 3 skirtingų optimizavimo funkcijų: Adam, SGD ir RMSprop.



15 pav. *Mokymosi ir validacijos duomenų tikslumas nuo pasirinktos optimizavimo funkcijos*



16 pav. *Mokymosi ir validacijos duomenų paklaida nuo pasirinktos optimizavimo funkcijos*

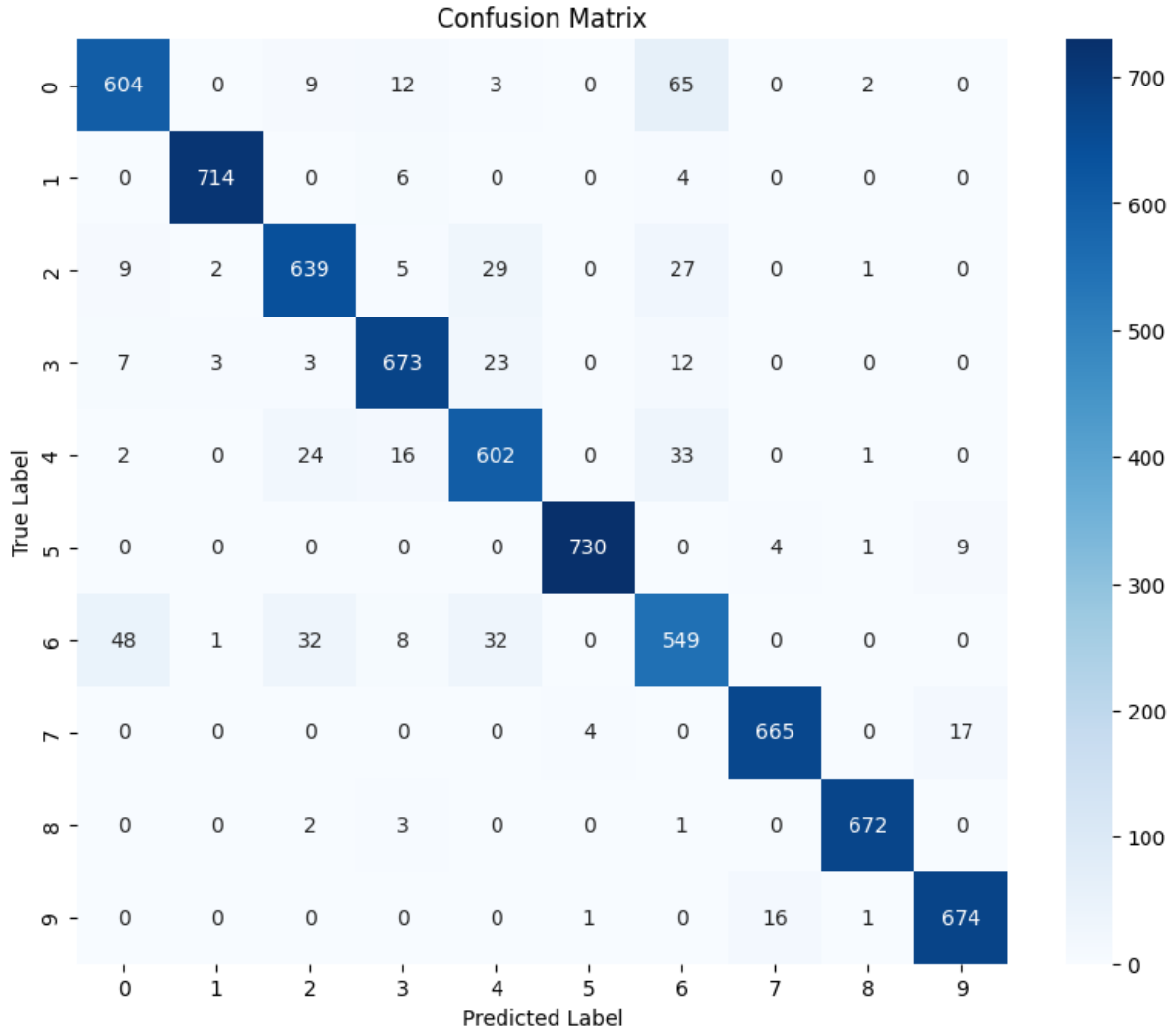
Pagal gautus rezultatus galima nustatyti, kuris modelis ir optimizavimo funkcija yra geriausi. Pagal naujus duomenis, gautus tyrimų antrame etape, galima nustatyti, kad Adam optimizavimo funkcija šiuo atveju yra nedaug geresnė už RMSprop funkciją.

Dabar rastą geriausią modelį, galima patestuoti su testavimo duomenų rinkinyje esančiais duomenimis.

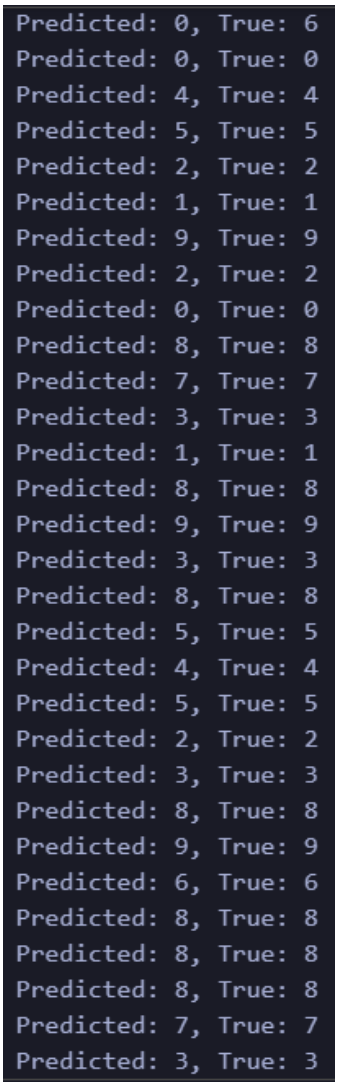
```
Test Loss: 0.18114188313484192
Test Accuracy: 0.9317142963409424
```

17 pav. *Testavimo duomenų klasifikavimo tikslumas ir paklaida*

Pagal **17 pav.** Testavimo duomenų klasifikavimo tikslumas ir paklaida pateiktus rezultatus galima teigti, kad tyrimo metus pasiektas geras klasifikavimo tikslumas ir paklaida. Tačiau vien remiantis šiais duomenimis, negalima pasakyti kaip tiksliai visos klasės yra klasifikuojamos. Tam reikia klasifikavimo matricos su testavimo duomenų aibės prognozės su atsitiktinai parinktais duomenimis .



18 pav. *Klasifikavimo matrica testavimo duomenims*



19 pav. *Testavimo duomenų aibėje parinktų 30 įrašų prognozės*

Pagal **18 pav.** Klasifikavimo matrica testavimo duomenims atvaizduotą klasifikavimo matricą galima tiksliau matyti, kurios klasės atpažįstamos netiksliausiai. Tai padeda nuspręsti, kurių klasių duomenis galima naudoti papildomam mokymuisi. Taip pat iš **19 pav.** Testavimo duomenų aibėje parinktų 30 įrašų prognozės gautų rezultatų galima matyti kaip modelis atpažįsta paveikslėlius su realiais pavyzdžiais.

6. Išvados

Atlikus tyrimą su konvoliuciniais neuroniniais tinklais vaizdų klasifikavimo užduočiai, buvo gauti šie rezultatai:

1. Iš trijų skirtingų modelių („paprastas“, „tarpinis“ ir „sudėtingas“) geriausią tikslumą demonstravo „sudėtingas“ modelis, naudojant 10^{-4} pradinę mokymosi žingsnio reikšmę.

2. Modelio optimizavimo funkcijų palyginimas parodė, kad Adam optimizavimo funkcija pasiekė geresnius rezultatus nei RMSprop ir SGD, nors skirtumas tarp Adam ir RMSprop buvo nedidelis.

3. Testavimo duomenų rinkinys patvirtino, kad modelis su Adam optimizavimo funkcija turi gerą klasifikavimo tikslumą ir mažą paklaidą.

4. Klasifikavimo matrica parodė, kurios klasės buvo atpažįstamos tiksliausiai, ir kurios klasės sukėlė didžiausias problemas, leidžiant suprasti modelio trūkumus ir galimus tobulinimo būdus.

5. Modelio apmokymo metu naudotos ankstyvo sustabdymo ir mokymosi žingsnio mažinimo taisyklės pasirodė efektyvios, padedančios išvengti modelio persimokymo ir padidinančios apmokymo efektyvumą.

6. Tyrimo metu buvo patvirtinta, kad tinkamas duomenų paruošimas ir hiperparametrų parinkimas yra esminiai veiksniai, siekiant aukšto modelio tikslumo.

7. Nors „sudėtingas“ modelis pasiekė geriausius rezultatus, tyrimas parodė, kad modelio sudėtingumas turi būti subalansuotas, nes didesnis sluoksnių skaičius ne visada garantuoja geresnius rezultatus.

8. Tolimesniems tyrimams rekomenduojama išbandyti dar sudėtingesnes architektūras ir papildomai optimizuoti hiperparametrus, siekiant dar didesnio tikslumo.

Šie rezultatai rodo, kad konvoliuciniai neuroniniai tinklai yra veiksmingi vaizdų klasifikavimo užduotims, tačiau norint gauti geriausią modelį reikia nuolat ieškoti ir testuoti jų architektūras ir optimizavimo metodus, siekiant dar geresnių rezultatų.