STU FIIT

Fakulta informatiky a informačných technológii

Ilkovičova 2, 842 16 Bratislava 4

Klasifikácia datasetov scikit-learn

(Zadanie č.3d)

Mária Matušisková **AIS ID: 116248**
UI (Umelá inteligencia)
Cvičiaci: Ing. Adam Novocký (Št 14:00) **ZS 2023/2024**

Mária Matušisková
AID ID: 116248

Cvičiaci: Ing. Adam Novocký (Št 14:00)
UI (Umelá inteligencia)

Zadanie úlohy:

Zadanie č. 3d

Vašou úlohou je vytvoriť sofistikovanú neurónovú sieť, ktorá bude schopná klasifikovať dáta zo známeho datasetu dostupného v knižnici scikit-learn. Tento dataset obsahuje informácie o rôznych triedach, do ktorých má byť klasifikované a špecifikuje aj vlastnosti prvkov, ktoré budú slúžiť na klasifikáciu.

Dataset z knižnice scikit-learn obsahuje dôležité informácie, ktoré sa týkajú tried alebo kategórií, do ktorých je potrebné priradiť dáta. Tieto údaje o triedach sú kľúčovými pre cieľovú klasifikáciu a umožnia neurónovej sieti naučiť sa rozpoznať vzory a vzťahy medzi rôznymi triedami.

Okrem toho dataset obsahuje údaje o rôznych vlastnostiach alebo atribútoch, ktoré charakterizujú jednotlivé prvky datasetu. Tieto atribúty poskytujú podstatné informácie, ktoré sú vstupom pre neurónovú sieť a umožňujú jej učiť sa, ako tieto vlastnosti ovplyvňujú priradenie do rôznych tried.

Vaša úloha zahŕňa nasledujúce kroky:

1. Zvoľte si a načítajte dataset zo scikit-learn a preskúmajte jeho štruktúru a obsah.
2. Definujte architektúru neurónovej siete, ktorá bude vhodná pre klasifikáciu na základe zvoleného datasetu.
3. Vhodne rozdeľte dataset na trénovaciu a testovaciu množinu.
4. Trénujte zodpovedajúcu neurónovú sieť na trénovacej množine dát a sledujte jej schopnosť naučiť sa vzory a vzťahy medzi triedami a atribútmi.
5. Vyhodnoťte výkonnosť neurónovej siete pomocou relevantných metrík (aspoň tri) a grafických vizualizácií.
6. Vyskúšajte viacero typov neurónových sietí (tiež aspoň tri) a diskutujte o ich plusoch a mínusoch.

Ako výstup sa vyžaduje funkčná neurónová sieť, ktorej keď sa zadá akákoľvek vstupná vzorka, bude ju schopná zaradiť túto vzorku do správnej triedy.

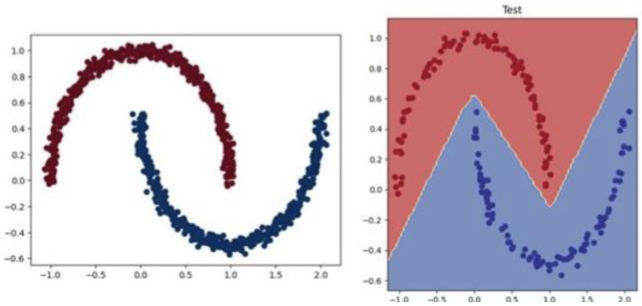
Je dôležité, aby ste experimentovali s jednotlivými vrstvami siete a vyhodnotili, ako zmena parametrov ovplyvňuje klasifikáciu siete. (Nielen počty neurónov ale aj architektúra – voľba lineárnych či nelineárnych aktivačných funkcii, ich kombinácia a podobne.) Presnosť by mala určite presiahnuť 90%. Zadanie realizujte v ľubovoľnom frameworku (odporúčané Tensorflow alebo Pytorch).

Dokumentácia by mala vychádzať z krokov, ktoré boli spomenuté v zadaní. Mala by obsahovať dôvod voľby daného datasetu, bližší popis datasetu a jeho spracovanie. Taktiež by mala obsahovať odôvodnenie voľby zvoleného pomeru na testovacie a trenovacie dáta. Ďalej by mala obsahovať experimentovanie s architektúrou siete – k daným sieťam aj výsledky z metrík. Vyhodnotenie plusov a mínusov použitých architektúr sietí. Ak sa dá výsledok vizualizovať tak to spraviť. Ak sa nedá vizualizovať, treba vložiť do siete aspoň 5 rôznych vzoriek a tie klasifikovať.

Dostupné datasety v sklearn:

Iris	California Housing
Diabetes	MNIST
Digits	Fashion-MNIST
Linnerud	make_classification
Wine	make_regression
Breast Cancer Wisconsin	make_blobs
Boston Housing	make_moons and make_circles
Olivetti Faces	Make_sparse_coded_signal

Príklad vizualizácie: (trénovacia množina a testovacia s rozhodovacou hranicou)



Mária Matušisková
AID ID: 116248

Cvičiaci: Ing. Adam Novocký (Št 14:00)
UI (Umelá inteligencia)

Table of Contents

Mária Matušisková
AID ID: 116248

Cvičiaci: Ing. Adam Novocký (Št 14:00)
UI (Umelá inteligencia)

1 Realizácia zadania, výber datasetu

Projekt bol realizovaný vo vývojárskom prostredí PyCharm v programovacom jazyku Python na počítači MacBook Pro (13-inch, M1, 2020). V zadaní sa použil iba jeden súbor *main.py.*

Pri výbere vhodného datasetu pre projekt sa zohľadnilo niekoľko kritérií. Ako je relevantnosť pre umelú inteligenciu a neurónky. Dataset Iris je široko využívaný, dobre dostupný a ľahký na pochopenie základov ako taká neurónová sieť funguje. Aké princípy využíva na rôzne matematické rovnice a funkcie. Využíva sa najme v oblasti štatistiky a aj strojového učenia. Je tiež veľmi dobre zdokumentovaný a dostupný v mnohých knižniciach.

Boli použité knižnice:

- **keras** – rozhranie (API) pre neurónové siete, trénuje ich a testuje
- **matplotlib** – vizualizácia dát, tvorba grafov
- **sklearn** – implementácia datasetu Iris (môže aj iné), vrátane tried a metód
- **tensorflow** – tvorba a trénovanie neurónových sieti, učí aj rôzne modely

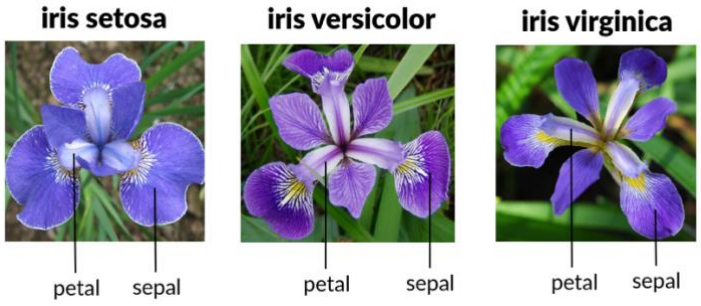
```
from keras import Sequential
from keras.src.layers import Dense
from keras.src.optimizers import Adam
from matplotlib import pyplot as plt
from sklearn import datasets
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics
import tensorflow as tf
```

Obrázok 1, import knižníc

2 Dataset Iris

Tento dataset poskytuje komplexné informácie o troch rôznych druhoch kvetov (Setosa, Versicolor, Virginica). Tieto dáta obsahujú záznamy o dĺžke a šírke v meraní kališných a okvetných lístkov. Jeho história siaha už od roku 1936, vo vydaní jedného článku od Ronalda Aylmera Fishera s názvom Používanie viacerých meraní v taxonomických problémoch.

Údaje poskytujú morfologické vlastnosti kvetov, meraných v centimetroch, vrátane dĺžky a šírky lístia a tiež okvetných lístkov.



Obrázok 2, zo stránky: Machine Learning HD

V kóde sú tieto atribúty rozdelené na SepalLengthCm, SepalWidthCm, PetalLengthCM, PetalWidthCm a príslušné triedy (Labels).

```
1 usage   mariamatusiskova
def iris():
    # load the iris dataset
    iris_database = datasets.load_iris()

    # show data
    print('\nDataset (SepalLengthCm, SepalWidthCm, PetalLengthCM, PetalWidthCm): ')
    print(iris_database.data[:5])
    print('\nLabels: ')
    print(iris_database.target[:5])
```

Obrázok 3, inicializácia datasetu Iris

```
Dataset (SepalLengthCm, SepalWidthCm, PetalLengthCM, PetalWidthCm):
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]

Labels:
[0 0 0 0 0]
```

Obrázok 4, výstup databázy 5-tich meraní

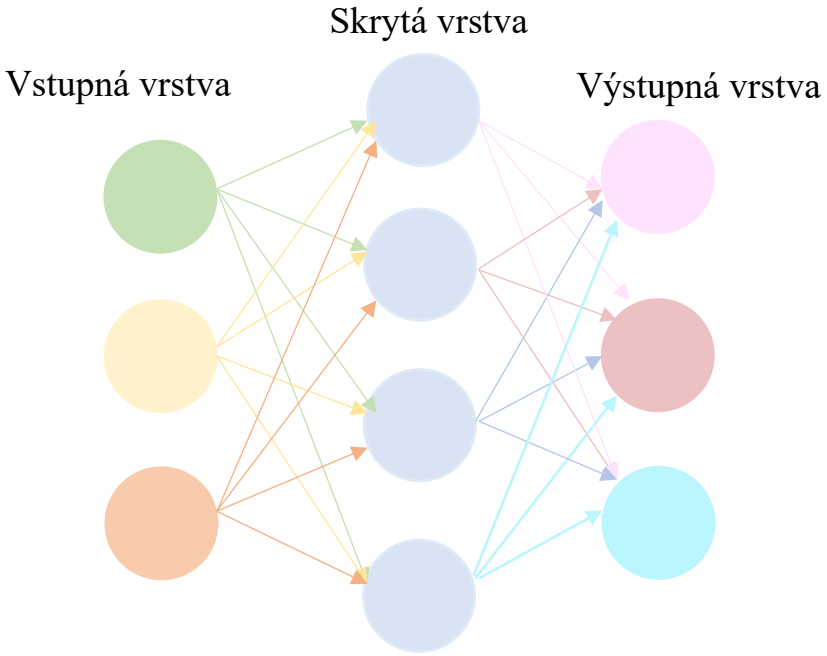
Obrázok 5, zobrazenie celej databázy Iris

3 Architektúra neurónovej siete

Vybrali sa štyri rôzne architektúry neurónových sietí. Každá je reprezentovaná samostatným modelom.

- model_one
- model_two
- model_three
- model_four

Skladá sa z niekoľko vrstiev. Základná je vstupná vrstva a výstupná vrstva. Medzi nimi sa nachádzajú skryté vrstvy.



Každý model začína vstupnou vrstvou s 16 neurónmi, ktorá zodpovedá štyrom vstupným atribútom datasetu Iris. Aktivačná funkcia tejto vrstvy je nastavená na relu (Recified Linear Unit).

Skryté vrstvy obsahuj rôzny počet neurónov s aktivačnými funkciami relu.

V prípade vybraného datasetu výstupnou vrstvou budú vždy 3 neuróny. Pretože zodpovedajú počtu tried (setosa, versicolor, virginica). Aktivačná funkcia je nastavená na softmax, čo je vhodné pre viacnásobnú klasifikáciu.

Kompiláciu modelu využívajú modely stratovej funkciou kategoriálnej krížovej entropie (categorical_crossentropy). Tiež používajú optimalizátor Adam s rýchlosťou učenia sa 0,001. Metrikou pri trénovaní je presnosť (accuracy).

Trénuje sa po 50 epochách, pretože dataset má dokopy 150 dát. Z toho 50 je pre jeden druh. Vývoj chyby a presnosti je sledovaný na trénovanej a validačnej sade.

Pre každú jednu sieť sú vytvorené grafy, ktoré vizualizujú vývoj chyby a presnosti počas jednotlivých epoch trénovania. Slúžia pre lepšie pochopenie danej problematiky.

4 Trénovacia a testovacia časť

V tejto fáze implementácie sú dáta rozdelené pomocou funkcie *train_test_split* na:

- **testovací set –** 10% dát, vyhodnotí konečný výkon modelu po jeho úplnom natrénovaní, podľa neho sa potom počítajú aj metriky a porovnávanie predikcii s reálnymi hodnotami
- **trénovací set –** 80% dát, slúži na adaptáciu váh modelu počas učenia
- **validačný set –** 10% dát, poskytuje možnosť monitorovať výkon modelu na dátach, ktoré neboli použité počas trénovania, čím zabráni pretrénovaniu

```
# test, train and validation data split
# train_test_split - utility for splitting a dataset into random train and test sets
# 10% of data for testing, 90% for training
x_training, x_test, y_training, y_test = train_test_split( *arrays: x, y, test_size=0.10)
x_training, x_val, y_training, y_val = train_test_split( *arrays: x_training, y_training, test_size=0.1)
# print(f'x_training: \n{x_training}\n x_test: \n{x_test}\n x_val: \n{x_val}\n y_training: \n{y_training}\n y_test: \n{y_test}\n y_val: \n{y_val}\n')
# after two splits, the program has three sets of data
# - training set 80%
# - validation set 10% (the validation helps ensure that model is learning well in some new situations)
# - test set 10%
```

Obrázok 6, rozdelenie dát na sady

5 Experimentovanie s architektúrou siete

Pre riešenie úlohy klasifikácie na datasete sa realizovala séria experimentácii s architektúrou neurónových sieti. Každý model od model_one po model_four bol vytvorený s rôznymi konfiguráciami vrstiev a počtom neurónov a cieľom preskúmať ich vplyv na výkonnosť modelu.

Experimentovalo sa aj s rôznym počtom skrytých vrstiev. Tieto sledovania sú určené na optimalizáciu a schopnosť sa adaptovať na komplexné vzory v dátach (kvetov Iris).

5.1 Popis architektúr modelov

model_one – základná architektúra s jednou skrytou vrstvou

```
### build models ###

# model one ------------------------------------------------------------------------------------------------------------
model_one = Sequential()

# input layer
model_one.add(Dense( units: 16, input_shape=(4,), activation='relu', name='input_layer'))
# hidden layers
model_one.add(Dense( units: 32, activation='relu', name='hidden_layer1'))
model_one.add(Dense( units: 16, activation='relu', name='hidden_layer2'))
# output layer
model_one.add(Dense( units: 3, activation='softmax', name='output'))

model_compile_summary_fit(model_one, x_val, y_val, x_training, y_training, x_test, y_test, f'{model_one=}'.split('=')[0])
```

Obrázok 7, vytvorenie neurónovej siete model_one

model_two – rozšírená architektúra s viacerými skrytými vrstvami a zvýšeným počtom neurónov

```
# model Two ------------------------------------------------------------------------------------------------------------
model_two = Sequential()

# input layer
model_two.add(Dense( units: 16, input_shape=(4,), activation='relu', name='input_layer'))
# hidden layers
# increased number of neurons in hidden_layer2
model_two.add(Dense( units: 64, activation='relu', name='hidden_layer1'))
model_two.add(Dense( units: 32, activation='relu', name='hidden_layer2'))
# added an additional hidden layer
model_two.add(Dense( units: 16, activation='relu', name='hidden_layer3'))
# output layer
model_two.add(Dense( units: 3, activation='softmax', name='output'))

model_compile_summary_fit(model_two, x_val, y_val, x_training, y_training, x_test, y_test, f'{model_two=}'.split('=')[0])
```

Obrázok 8, vytvorenie neurónovej siete model_two

model_three – architektúra s vyšším počtom neurónov v prvých dvoch vrstvách

```
# model three (an example with different architecture) ----------------------------------------------------------
model_three = Sequential()

# input layer
# increased the number of neurons in input_layer
model_three.add(Dense( units: 16, input_shape=(4,), activation='relu', name='input_layer'))
# hidden layers
model_three.add(Dense( units: 32, activation='relu', name='hidden_layer1'))
model_three.add(Dense( units: 16, activation='relu', name='hidden_layer2'))
model_three.add(Dense( units: 8, activation='relu', name='hidden_layer3'))
# output layer
model_three.add(Dense( units: 3, activation='softmax', name='output'))

model_compile_summary_fit(model_three, x_val, y_val, x_training, y_training, x_test, y_test, f'{model_three=}'.split('=')[0])
```

Obrázok 9, vytvorenie neurónovej siete model_three

model_four – architektúra s menším počtom neurónov v tretej skrytej vrstve

```
# Model Four ----------------------------------------------------------
model_four = Sequential()

# input layer
model_four.add(Dense( units: 16, input_shape=(4,), activation='relu', name='input_layer'))
# hidden layers
model_four.add(Dense( units: 32, activation='relu', name='hidden_layer1'))
model_four.add(Dense( units: 16, activation='relu', name='hidden_layer2'))
# reduced the number of neurons in hidden_layer3
model_four.add(Dense( units: 8, activation='relu', name='hidden_layer3'))
# output layer
model_four.add(Dense( units: 3, activation='softmax', name='output'))

model_compile_summary_fit(model_four, x_val, y_val, x_training, y_training, x_test, y_test, f'{model_four=}'.split('=')[0])
```

Obrázok 10, vytvorenie neurónovej siete model_four

5.2 Vyhodnotenie a porovnanie

Každý model bol skompilovaný s optimalizačným algoritmom Adam a stratovou funkciou categorical_crossentropy. Po trénovaní sa analyzovali vývoje presnosti a chyby v priebehu epoch pre trénovanie a validačné dáta.

Modely boli testované na testovacej vzorke a následne sa vyhodnotili metriky. Výsledky boli vizualizované v grafoch.

```
4 usages  mariamatusiskova
def model_compile_summary_fit(model, x_val, y_val, x_training, y_training, x_test, y_test, model_name):
    optimizer = Adam(learning_rate=0.001)

    model.compile(
        loss="categorical_crossentropy",
        optimizer=optimizer,
        metrics=["accuracy"]
    )

    model.summary()

    # train model (the neural network)
    history_results = model.fit(
        x_training,
        y_training,
        verbose=2,
        epochs=50,
        validation_data=(x_val, y_val)
    )

    create_accuracy_graph(model_name, history_results)
    create_loss_graph(model_name, history_results)

    # test
    test(model, x_test, y_test, x_training, y_training)
```

Obrázok 11, funkcia na skompilovanie, vypísanie a testovanie dát

```
1 usage  mariamatusiskova
def test(model, x_test, y_test, x_training, y_training):
    test_loss, test_accuracy = model.evaluate(x_test, y_test)
    print(f'--------- Test Loss: {round(test_loss * 100, 4)}%\n--------- Test Accuracy: {round(test_accuracy * 100, 4)}%')

    print("--------- Training predictions:")
    test_training = model.predict(x_training)
    training_positives = tf.argmax(test_training, axis=1)

    true_positives = tf.argmax(y_training, axis=1)
    print(metrics.classification_report(true_positives, training_positives, digits=3))
    print(metrics.confusion_matrix(true_positives, training_positives))

    print("--------- Test predictions:")
    test_prediction = model.predict(x_test)
    predicted_positives = tf.argmax(test_prediction, axis=1)

    true_positives = tf.argmax(y_test, axis=1)
    print(metrics.classification_report(true_positives, predicted_positives, digits=3))
    print(metrics.confusion_matrix(true_positives, predicted_positives))
```

Obrázok 12, testovacia funkcia

```
1 usage   mariamatusiskova
def create_loss_graph(model_name, history_results):
    # validation accuracy  is used to assess how well the model generalizes to new, unseen data
    plt.plot( *args: range(50), history_results.history['loss'], color='b', label='Training loss')
    plt.plot( *args: range(50), history_results.history['val_loss'], color='r', label='Validation loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.title(f'Loss over Epochs of the {model_name}')
    plt.legend()
    plt.show()


1 usage   mariamatusiskova
def create_accuracy_graph(model_name, history_results):
    # validation loss  is used to assess how well the model generalizes to new, unseen data
    plt.plot( *args: range(50), history_results.history['accuracy'], color='g', label='Training accuracy')
    plt.plot( *args: range(50), history_results.history['val_accuracy'], color='orange', label='Validation accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.title(f'Accuracy over Epochs of the {model_name}')
    plt.legend()
    plt.show()
```

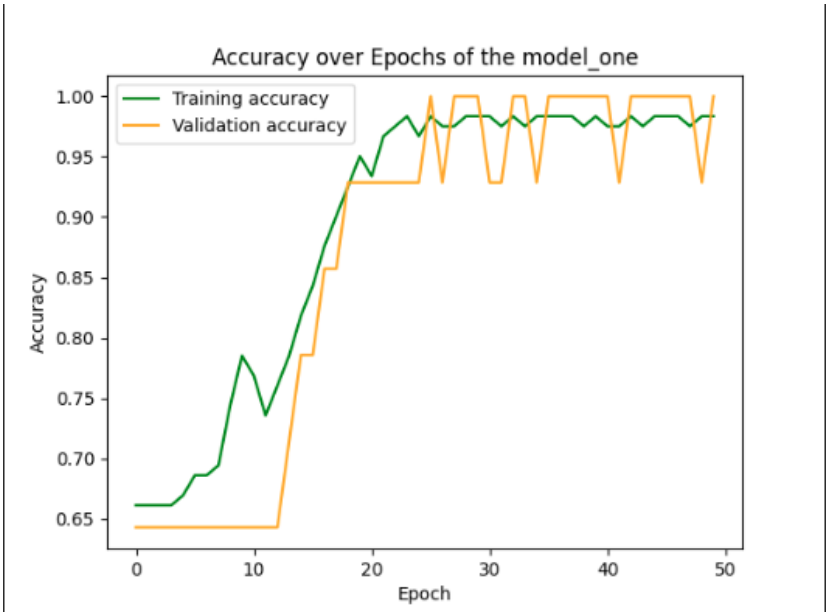
Obrázok 13, funkcie na vytvorenie výsledkov testov v grafe

6 Výsledky metrík

Vizualizácia grafov jednotlivých modelov.

6.1 Graf trénovacej a validačnej presnosti

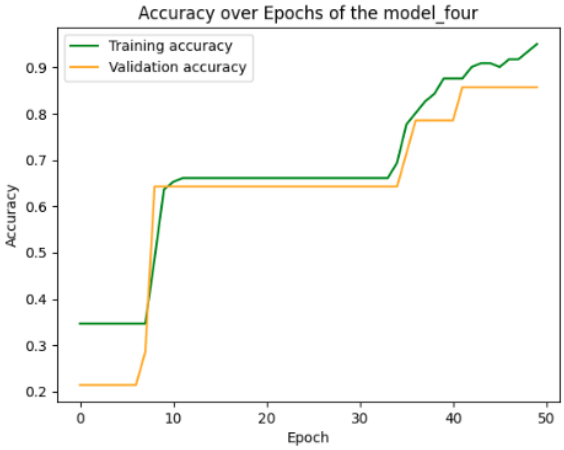
Tento graf zobrazuje presnosť modelu počas trénovania a overovania v priebehu epoch. Presnosť indikuje percento správne klasifikovaných príkladov. Čím sú krivky bližšie, tým lepšie model generalizuje na nové dáta.



Obrázok 14, model_one, test presnosti

Obrázok 15, model_two, test presnosti

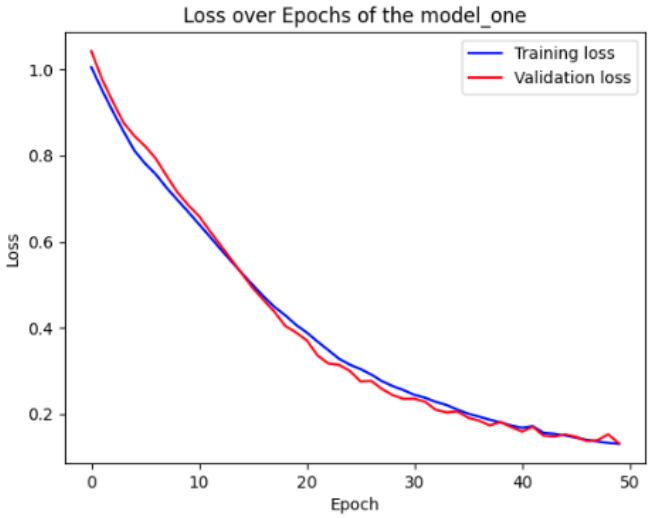
Obrázok 16, model_three, test presnosti



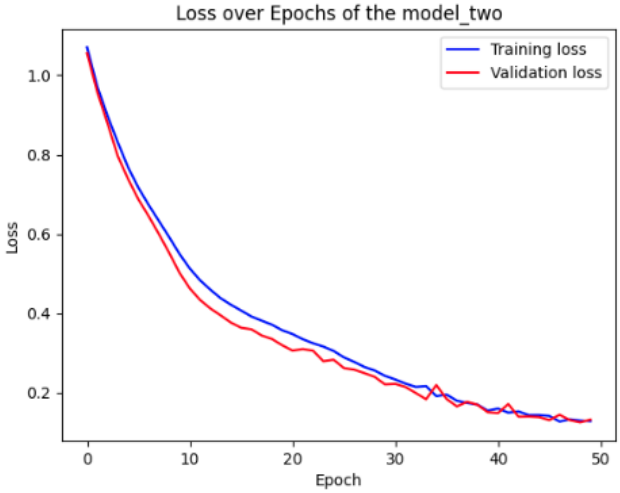
Obrázok 17, model_four, test presnosti

6.2 Graf trénovacej a validačnej chyby

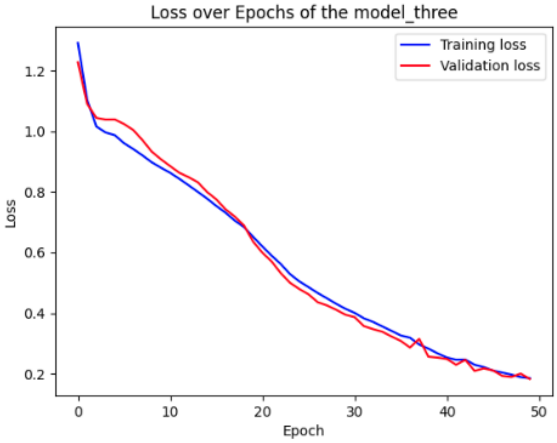
Grafické spracovanie chybovosti entropie (loss) modelov počas trénovania a overovania v priebehu epoch. Trénovacia časť môže identifikovať, ako rýchlo model konverguje, zatiaľ čo časť overovania pomáha odhaliť prípadné preučenie alebo nedoučenie.



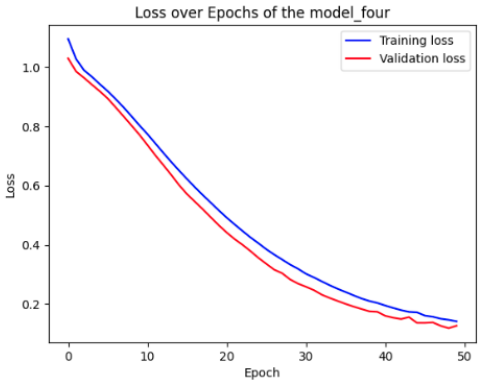
Obrázok 18, model_one, test chybovosti



Obrázok 19, model_two, test chybovosti

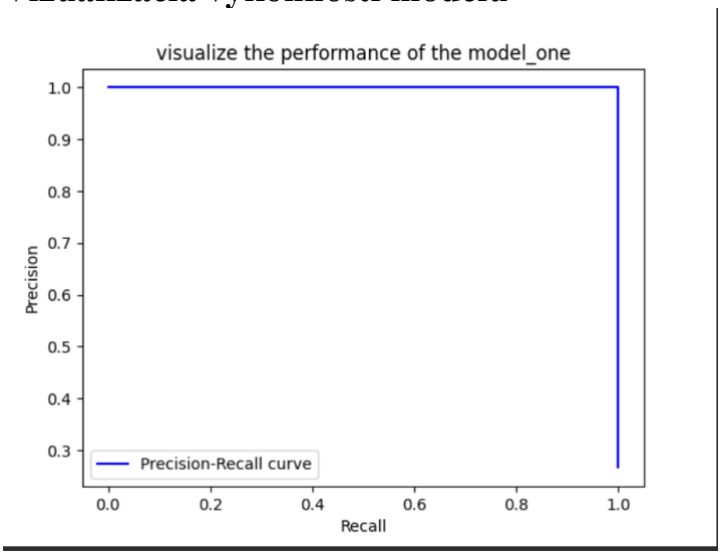


Obrázok 20, model_three, test chybovosti

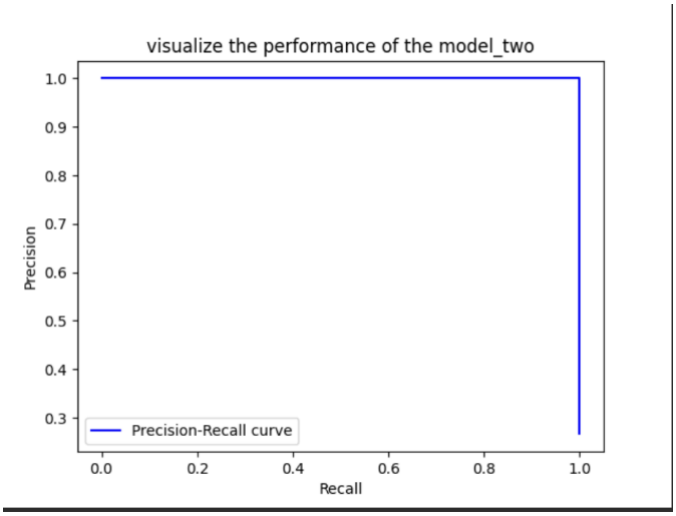


Obrázok 21, model_four, test chybovosti

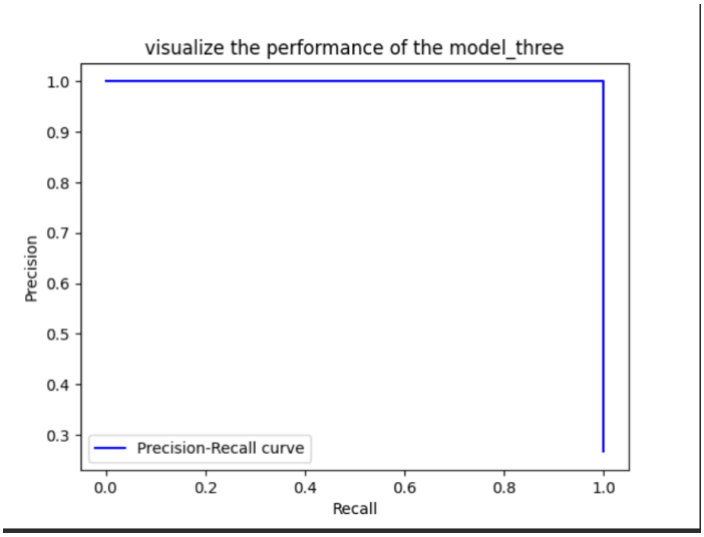
6.3 Vizualizácia výkonnosti modelu



Obrázok 22, model_one presision-recall



Obrázok 23, model_two precision-recall



Obrázok 24, model_three precision-recall

Obrázok 25, model_four precision-recall

6.4 Vyhodnotenie na testovacích dátach

Po trénovaní sa modely otestovali na testovacích dátach a získala sa presnosť a strata. Taktiež sa vykonala analýza výsledkov pomocou klasifikačnej správy a matice. Táto analýza ukáže presnejšie štatistiky.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (Dense)         (None, 16)                80

 hidden_layer1 (Dense)       (None, 32)                544

 hidden_layer2 (Dense)       (None, 16)                528

 output (Dense)              (None, 3)                 51

=================================================================
Total params: 1203 (4.70 KB)
Trainable params: 1203 (4.70 KB)
Non-trainable params: 0 (0.00 Byte)
```

Obrázok 26, ukázanie vrstiev vo výpise modelu one

```
Epoch 49/50
4/4 - 0s - loss: 0.1890 - accuracy: 0.9669 - val_loss: 0.1485 - val_accuracy: 0.9286 - 11ms/epoch - 3ms/step
Epoch 50/50
4/4 - 0s - loss: 0.1816 - accuracy: 0.9835 - val_loss: 0.1406 - val_accuracy: 1.0000 - 11ms/epoch - 3ms/step
1/1 [==============================] - 0s 13ms/step - loss: 0.2262 - accuracy: 1.0000
--------- Test Loss: 22.6198%
--------- Test Accuracy: 100.0%
--------- Training predictions:
4/4 [==============================] - 0s 472us/step
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        42
           1      1.000     0.950     0.974        40
           2      0.951     1.000     0.975        39

    accuracy                          0.983       121
   macro avg      0.984     0.983     0.983       121
weighted avg      0.984     0.983     0.983       121

[[42  0  0]
 [ 0 38  2]
 [ 0  0 39]]
--------- Test predictions:
1/1 [==============================] - 0s 7ms/step
              precision    recall  f1-score   support

           0      1.000     1.000     1.000         4
           1      1.000     1.000     1.000         5
           2      1.000     1.000     1.000         6

    accuracy                          1.000        15
   macro avg      1.000     1.000     1.000        15
weighted avg      1.000     1.000     1.000        15

[[4 0 0]
 [0 5 0]
 [0 0 6]]
Model: "sequential_1"
```

Obrázok 27, vypísanie výsledkov testovania modelu one

Mária Matušisková
AID ID: 116248

Cvičiaci: Ing. Adam Novocký (Št 14:00)
UI (Umelá inteligencia)

7 Výhody a nevýhody použitých architektúr

Každá z navrhnutých architektúr neurónových sietí má svoje výhody a nevýhody, ktoré môžu byť zohľadnené pri rozhodovaný o ich použití.

- model_one
  - výhody – jednoduchá architektúra s nízkou náročnosťou na zdroje
  - nevýhody – obmedzená schopnosť zložitejšie vzory v dátach
- model_two
  - výhody – zvýšený počet neurónov a vrstiev pre hlbšiu reprezentáciu dát
  - nevýhody – zvýšená náročnosť na zdroje, môže vykazovať znaky preučenia
- model_three
  - výhody – zmena počtu neurónov v rôznych vrstvách pre flexibilitu
  - nevýhody – zvýšená komplexita, ktorá môže viesť ku nadmernému trénovaniu
- model_four
  - výhody – redukcia počtu neurónov pre efektívnejšie učenie
  - nevýhody – zvýšená komplexita, ktorá môže viesť k nadmernému trénovaniu

Záver

Vo všeobecnosti platí, že čím väčšia a hlbšia architektúra je, tak môže mať lepšiu schopnosť naučiť sa komplexnejšie vzory. Avšak môže dochádzať ku nadmernému trénovaniu.

Naopak, jednoduchšie architektúry môžu byť rýchlejšie a menej náročné na zdroje. Ale nemusia zvládať náročnejšie úlohy.

Pri výbere architektúry je dôležité zvážiť kompromis medzi výkonom a náročnosťou na zdroje vzhľadom na konkrétnu úlohu a dataset.