I – SUNS:

ZADANIE 01

Meno: Róbert Šumšala
ID: 111 464

ÚVOD

Náplň zadania:

Náplňou zadania bolo implementovať program, v ľubovoľnom jazyku, ktorý bude kategorizovať piesne z platformy Spotify do kategórií podľa nálady.

Programovací jazyk a knižnice:

Ako programovací jazyk sme vybrali Python, a využili knižnice pandas, numpy, tensorflow, matplotlib, sklearn a keras.

1. časť: Jednoduchá sieť

Načítanie dát:

Ako prvé sme načítali dáta obsahujúce údaje o jednotlivých skladbách, ktoré poskytuje Spotify. Dáta obsahovali 12 kategórií, vrátane emócie, ktorá danú skladbu vystihuje najviac. Náš dataset obsahoval 11 929 vzoriek.

Predspracovanie dát:

Po načítaní sme si prezreli informácie, hovoriace o tom, ktorá kategória je ako vyjadrená, resp. aké hodnoty, z akého intervalu môže nadobúdať. Tieto dodatočné informácie boli nájdené tiež na Spotify. Ďalej sme si dali v programe vypísať minimá a maximá jednotlivých kategórií a porovnali so získanými poznatkami o intervaloch pre jednotlivé kategórie.

---------- Min ----------	
danceability	0.0
energy	0.000197
loudness	-47.046
speechiness	0.0
acousticness	0.0
instrumentalness	0.0
liveness	0.00967
valence	0.0
tempo	0.0
duration_ms	-427346.0
popularity	0.0
number_of_artists	1.0
explicit	False

Min hodnoty pred odstránením outlierov

---------- Max ----------	
danceability	8.375
energy	1.0
loudness	1.519
speechiness	0.965
acousticness	0.996
instrumentalness	0.994
liveness	0.997
valence	0.995
tempo	241.423
duration_ms	1930821300.0
popularity	82.0
number_of_artists	19.0
explicit	True

Max hodnoty pred odstránením outlierov

Na základe týchto poznatkov sme určili outliery (neobvyklé hodnoty). Stĺpec, ktorého minimum sa vymykali určeným intervalom bol stĺpec „duration“. Táto kategória síce nemala určený interval, ale usúdili sme, že dĺžka skladby udávaná v milisekundách nemôže byť záporná.

Stĺpce, ktorých maximá nepatrili do pridelených intervalov, boli „danceability“ a „loudness“. Pričom kategória „danceability“ nemala nadobúdať hodnoty väčšie ako jedna, ale maximum v príslušnom stĺpci bolo 8,375. Zatiaľ čo hodnoty „loudness“ mali byť z intervalu <-60, 0>, medzi dátami boli aj hodnoty väčšie ako nula.

Tieto outliery sme odstránili nasledovne:

```
df = df[df['duration_ms'] >= 0]
df = df[df['danceability'] <= 1]
df = df[df['loudness'] <= 0]
```

pričom df bol náš dataframe.

Následne sme si vypísali počet chýbajúcich hodnôt pre každý stĺpec. Podľa výpisu sme zistili, že chýbajúce hodnoty sa vyskytli len v troch stĺpcoch („popularity“, „number_of_artists“, „top_genre“).

Množstvo chýbajúcich hodnôt pre stĺpec nepresiahol 169, čo sme považovali za zanedbateľné pri skoro 12 000 vzorkách. Preto by sme zvažovali odstránenie jednotlivých null hodnôt, namiesto celých stĺpcov. Keďže sme ale tieto kategórie považovali za nepotrebné pre trénovanie našej siete, odstránili sme celé tieto stĺpce.

Okrem týchto troch spomínaných stĺpcov sme za nepotrebný určili aj stĺpec „url“. Preto sme tieto štyri stĺpce z dataframu odstránili.

```
Length of the dataset: 11929
danceability           0
energy                 0
loudness               0
speechiness            0
acousticness           0
instrumentalness       0
liveness               0
valence                0
tempo                  0
duration_ms            0
popularity           116
number_of_artists    120
explicit               0
name                   0
url                    0
genres                 0
filtered_genres        0
top_genre            169
emotion                0
```

Počet chýbajúcich hodnôt pred odstránením stĺpcov

```
Lenght of dataset: 11929
danceability        0
energy              0
loudness            0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_ms         0
explicit            0
name                0
genres              0
filtered_genres     0
emotion             0
```

Počet chýbajúcich hodnôt po odstránení stĺpcov

V ďalšej fáze sme si dali vypísať typy dát jednotlivých stĺpcov aby sme zistili, ktoré nie sú reprezentované číselnou hodnotou, s ktorou dokáže trénovací algoritmus pracovať. Zistili sme, že ide o dáta kategórií „name“, „genres“, „filtered_genres“ a „emotion“. Preto sme tieto stĺpce zakódovali použitím LabelEncoderu. Čo znamená, že každej nečíselnej hodnote bola pridelená unikátna číselná hodnota (Integer).

Trénovanie jednoduchej siete (*použitý Sklearn*):

Ako prvé sme rozdelili dáta na vstupné (X) a výstupné (y) množiny. Za výstupné dáta sme zvolili kategóriu „emotion“, a ako vstupné dáta sme určili zvyšné stĺpce z pripraveného dataframu.

V ďalšom kroku sme ešte dáta rozdelili na trénovaciu, validačnú a testovaciu množinu. Pomer sme zvolili nasledovne:

- trénovacia – 50%
- validačná – 12,5%
- testovacia – 12,5%

```
# Splitting dataset into train, valid and test
X_train, X_valid_test, y_train, y_valid_test = train_test_split(X, y,
shuffle=True, test_size=0.25, random_state=42)

X_valid, X_test, y_valid, y_test = train_test_split(X_valid_test,
y_valid_test, shuffle=True, test_size=0.5, random_state=42)
```

Následne sme naše rozdelené dáta škálovali. Keďže väčšina z dát už bola v intervale od 0 po 1, rozhodli sme sa použiť *MinMaxScaler()*.

```
# Scale data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid_test)
X_test = scaler.transform(X_test)
```

Pre skontrolovanie si výsledkov škálovacieho procesu sme si dali vypísať minimá a maximá jednotlivých kategórií nášho dataframu.

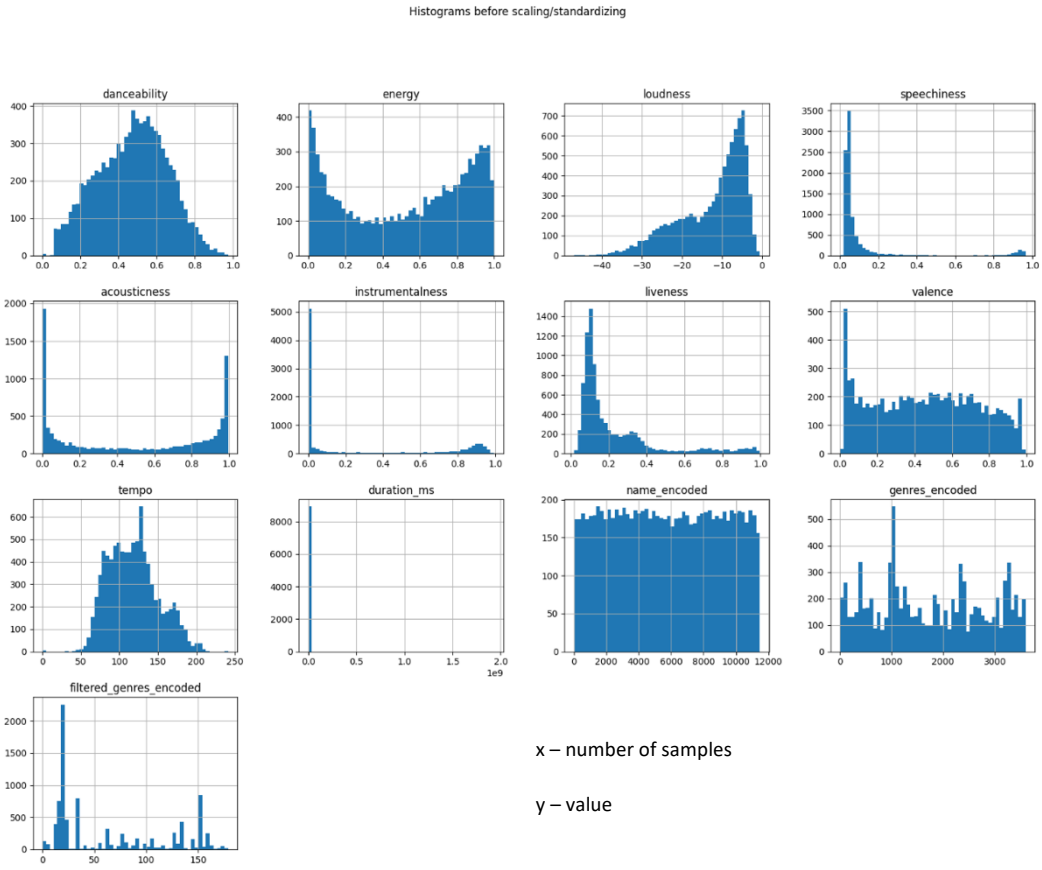
```
---------- Min ----------
danceability              0.0
energy                    0.0
loudness                  0.0
speechiness               0.0
acousticness              0.0
instrumentalness          0.0
liveness                  0.0
valence                   0.0
tempo                     0.0
duration_ms               0.0
explicit                  0.0
name_encoded              0.0
genres_encoded            0.0
filtered_genres_encoded   0.0
```

Minimá po škálovaní

```
---------- Max ----------
danceability              1.0
energy                    1.0
loudness                  1.0
speechiness               1.0
acousticness              1.0
instrumentalness          1.0
liveness                  1.0
valence                   1.0
tempo                     1.0
duration_ms               1.0
explicit                  1.0
name_encoded              1.0
genres_encoded            1.0
filtered_genres_encoded   1.0
```

Maximá po škálovaní

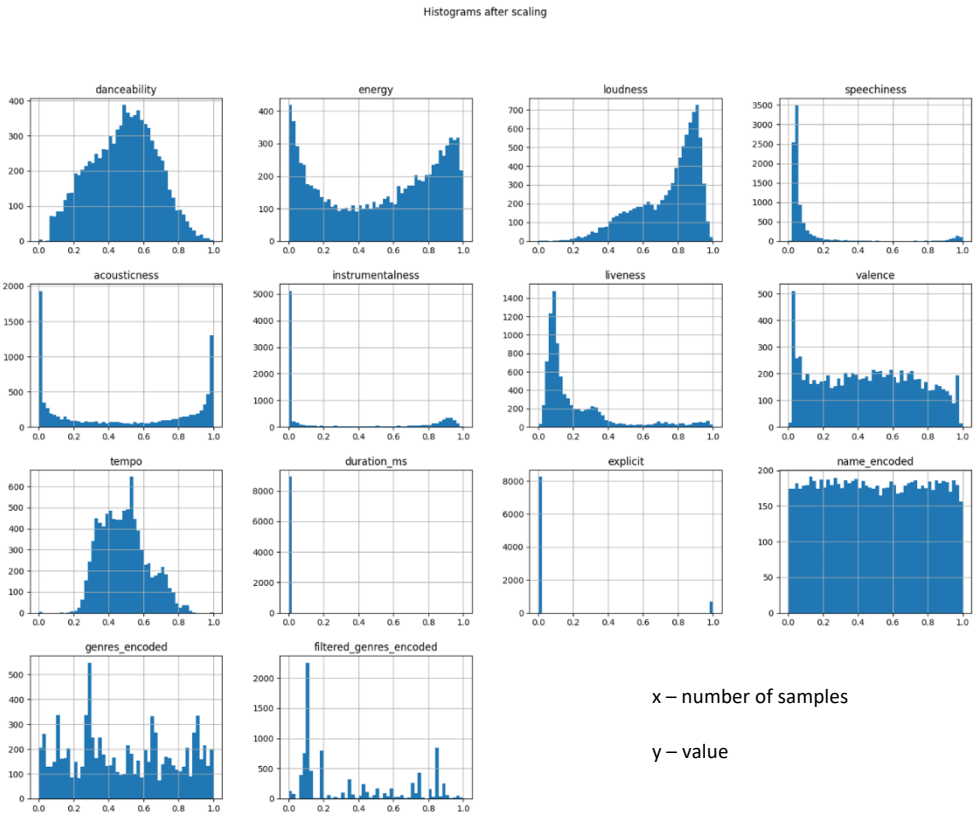
Pre ešte lepšie zobrazenie sme si dali pre každú kategóriu vypísať histogram pred aj po procese škálovania. Každý graf ukazuje koľko dát, z danej kategórie, nadobúdalo aké hodnoty. Na základe faktu, že grafy sa vizuálne nezmenili a zároveň hodnoty na xovej osi, ktoré pred škálovaním boli rôzne, sa pohybovali len v intervale od 0 po 1, sme usúdili, že škálovanie bolo úspešné. (Medzi grafmi zobrazujúcimi dáta pred škálovaním sa nenachádza kategória „explicit“ a po škálovaní už áno. Je to spôsobené tým, že táto kategória nebola zakódovaná, keďže to nebolo potrebné, čiže stále obsahovala hodnoty true, false. Avšak po škálovaní tieto hodnoty boli už zmenené na 0 a 1 a graf sa vypísal. Tento graf by sa pred a po aj tak nelíšil, keďže v podstate, sa jeho minimálna a maximálna hodnota nezmenili.)



x – number of samples

y – value

Histogramy zobrazujúce rozdelenie dát v jednotlivých kategóriach pred škálovaním



x – number of samples

y – value

Histogramy zobrazujúce rozdelenie dát v jednotlivých kategóriach po škálovaním

Na záver sme zostavili trénovací model a natrénovali jednoduchú sieť. Použili sme knižnicu Sklearn. Po viacerých experimentoch sme zvolili za vhodné nasledovné nastavenie siete.

- 3 skryté vrstvy s počtami neurónov: 50, 25, 10
- Maximálny počet iterácií: 250
- Miera učenia: adaptívna
- Počiatočná miera učenia: 0,001
- EarlyStopping nebol nutný

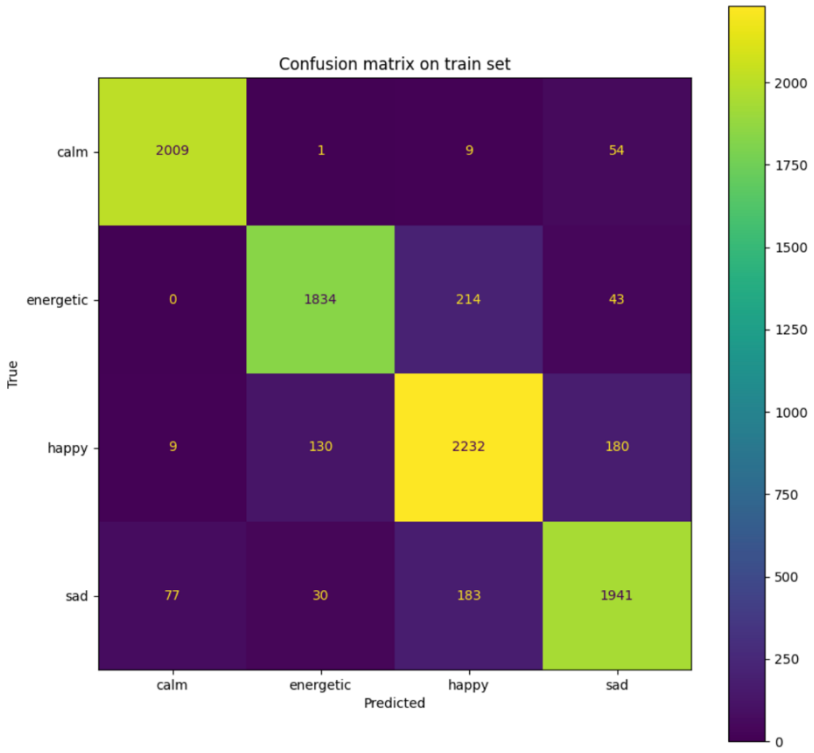
```
clf = MLPClassifier(
    hidden_layer_sizes=(50, 25, 10),
    random_state=1,
    max_iter=250,
    validation_fraction=0.2,
    early_stopping=False,
    learning_rate='adaptive',
    learning_rate_init=0.001,
).fit(X_train, y_train)
```

Vyhodnotenie jednoduchej siete (*použitý Sklearn*)

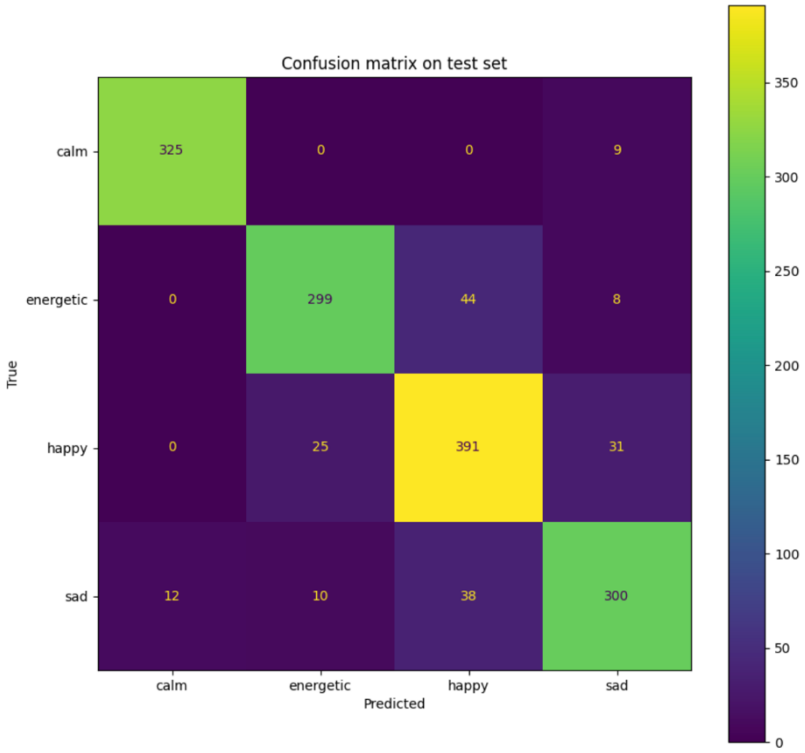
Naša natrénovaná sieť dosiahla presnosť na trénovacom sete 89,6% a na testovacom 88,1%.

```
MLP accuracy on train set:  0.8960
MLP accuracy on test set:  0.88136
```

Vyhodnotenie siete sme spravili na základe informácií, ktoré nám poskytli konfúzne matice pre testovacie aj trénovacie dáta. Po ich prezretí sme dospeli k záveru, že natrénovanie našej jednoduchej siete bolo úspešné.



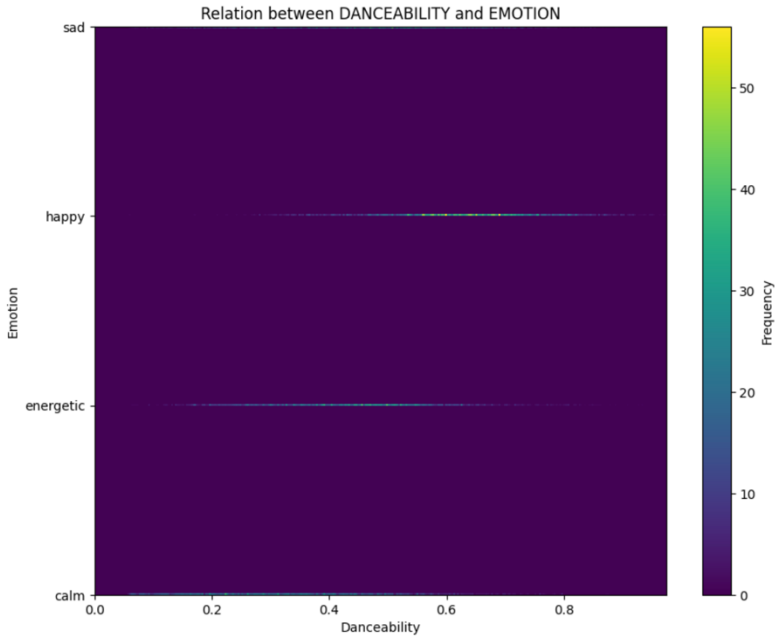
Konfúzna matica pre trénovacie dáta jednoduchej siete



Konfúzna matica pre testovacie dáta jednoduchej siete

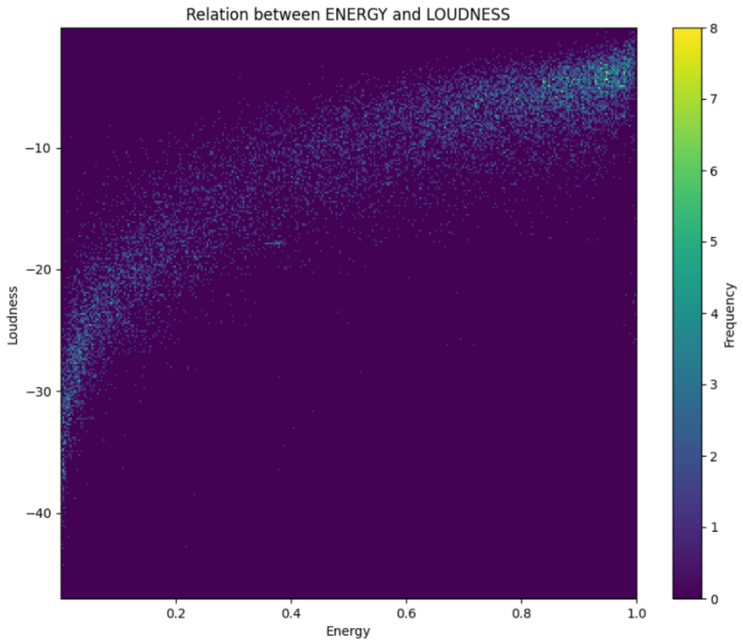
2. časť: EDA

Analyzovaním datasetu cez EDA (Exploratory Data Analysis) sme našli rôzne závislosti medzi rôznymi kategóriami datasetu. Pracovali sme s dátami po odstránení outlierov. Závislosti sme zobrazili v podobe heatmap grafov knižnice matplotlib. Toto je päť zaujímavých vzťahov, ktoré sme objavili.



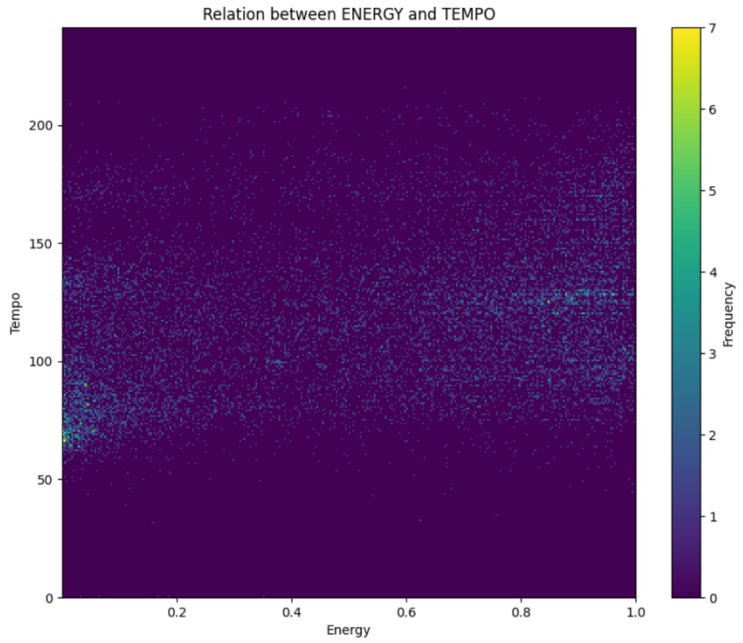
Graf zobrazujúci vzťah medzi kategóriami „danceability“ a „emotion“

Z vyššie uvedeného grafu vyplýva, že kategória „danceability“ ovplyvňuje kategóriu „emotion“. Ak je skladba výrazne tanečná, zaraďuje sa medzi veselé, ak priemerne, spadá do kategórie energických skladieb a ak je iba mierne tanečná, skladba je považovaná za pokojnú. Taktiež sa dá z grafu vyčítať, že pokiaľ je skladba tanečná („danceability“ sa nerovná nule), nie je považovaná za smutnú.



Graf zobrazujúci vzťah medzi kategóriami „energy“ a „loudness“

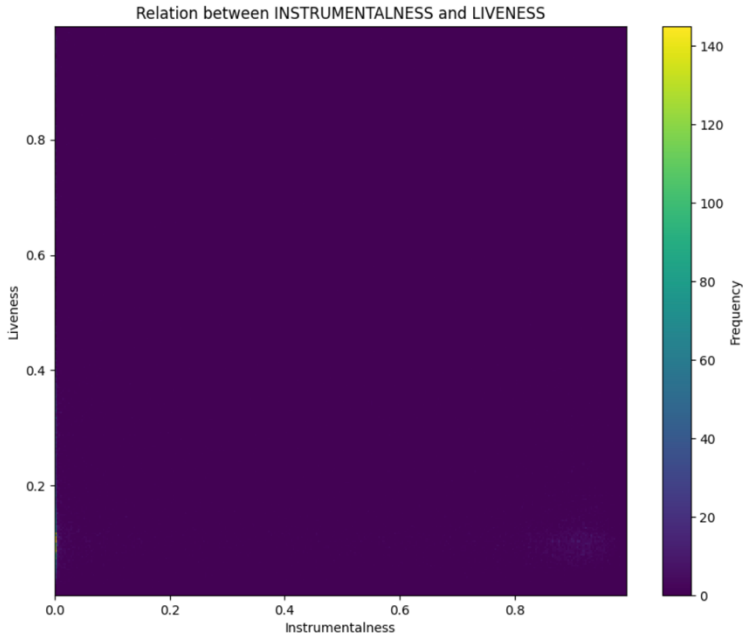
Tento graf ukazuje závislosť energie od hlasitosti a vidíme, že čím je skladba hlasnejšia, tým viac je energická. Tak ako by sme aj predpokladali.



Graf zobrazujúci vzťah medzi kategóriami „energy“ a „tempo“

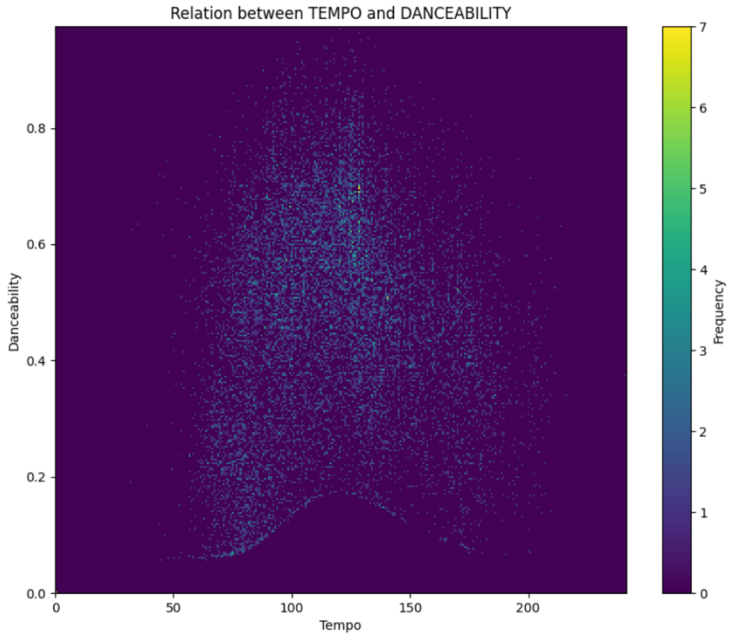
Na tomto grafe sledujeme opäť energiu, tentoraz ale v závislosti s tempom skladby. Prekvapujúco je tento vzťah menej výrazný ako by sme očakávali. Napriek tomu sa dá

povedať, že vo veľa prípadoch platí, čím rýchlejšia skladba, tým energickejšia. Nie je to však pravidlo.



Graf zobrazujúci vzťah medzi kategóriami „instrumentalness“ a „livness“

Vďaka tomuto zobrazenie, popisujúcemu vzťah inštrumentálnosti a živosti skladby, sme zistili, že medzi týmito dvomi kategóriami nie je žiadna korelácia. Fakt, ktorý sa dal očakávať, keďže prítomnosť obecenstva nebýva podmienená prevahou nástroja, či spevu.



Graf zobrazujúci vzťah medzi kategóriami „tempo“ a „danceability“

Na tomto grafe vidíme, že medzi tempom a tanečnosťou skladby existuje závislosť. Aj keď neplatí pravidlo, čím rýchlejšia skladba, tým tanečnejšia, vieme zo zobrazenia vyčítať, že vo väčšine prípadov je skladba s miernym tempom považovaná za najtanečnejšiu. S rastúcim a aj klesajúcim tempom, voči miernemu tempu, tanečnosť skladby klesá.

3. časť: Finálna neurónová sieť

Predspracovanie dát:

Dataframe sme použili rovnaký ako pri trénovaní jednoduchšej siete v prvej časti. To znamená, odstránené outliery a aj nepotrebné stĺpce („popularity“, „number_of_artists“, „top_genre“, „url“). Tentoraz sme navyše odstránili aj kategórie „genres“, „filtered_genres“ a „name“. Ktoré sa ukázali ako nie veľmi dôležité pre trénovanie siete.

Trénovanie finálnej siete (*použitý Keras*) – *Prototyp*:

Tak ako v prvej časti zadania, sme najprv rozdelili dáta na vstupné (X) a výstupné (y) množiny. Za výstupné dáta sme zvolili kategóriu „emotion“, a ako vstupné dáta sme určili zvyšné stĺpce z pripraveného dataframu.

Následne sme kategóriu „emotion“, t.j. množinu y, zakódovali one-hot kódovaním.

```
y_enc = pd.get_dummies(y)
```

Princíp tohoto kódovania spočíva v nahradení pôvodnej kategórie viacerými stĺpcami/ka-tegóriami. Každý stĺpec reprezentuje jednotlivý prvok z pôdnej kategórie. Dôsledkom je, že každá vzorka dát už neobsahuje jednu hodnotu, ale vektor núl s jednotkou na pozícií, ktorá zodpovedá poradiu novo pridaného stĺpca, ktorý reprezentuje príslušnú kategóriu, do ktorej vzorka spadá.

	calm	energetic	happy	sad
0	False	False	True	False
1	False	False	True	False
2	False	True	False	False
3	False	False	True	False
4	False	False	True	False
5	False	True	False	False
6	False	False	True	False

Tvar množiny y po one-hot zakódovaní (prvých 6 vzoriek)

V ďalšom kroku sme rozdelili dáta na trénovaciu, validačnú a testovaciu množinu. Rovnakým spôsobom ako v prvej časti. Pomer sme taktiež zachovali:

- trénovacia – 50%
- validačná – 12,5%
- testovacia – 12,5%

```
# Splitting dataset into train, valid and test
X_train, X_valid_test, y_train, y_valid_test = train_test_split(X, y_enc,
shuffle=True, test_size=0.25, random_state=42)
X_valid, X_test, y_valid, y_test = train_test_split(X_valid_test,
y_valid_test, shuffle=True, test_size=0.5, random_state=42)
```

Dáta sme potom škálovali. Opäť sme použili *MinMaxScaler()*.

```
# Apply Min-Max scaling
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_valid = scaler.transform(X_valid)
X_test = scaler.transform(X_test)
```

Na záver sme vytvorili nový zložitejší model pre našu neurónovú sieť. Tentoraz sme využili knižnicu keras. Náš prvý pokus mal nasledovné parametre:

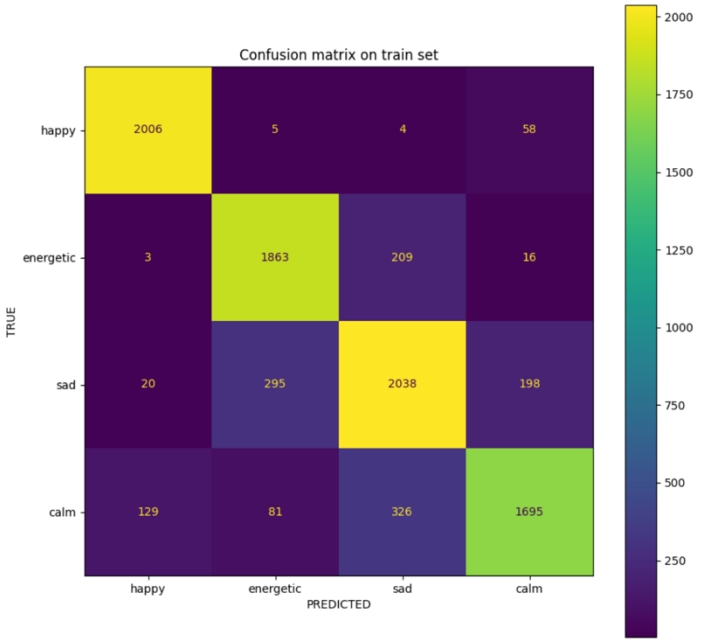
- 2 skryté vrstvy s počtami neurónov: 24, 15
- Výstupná vrstva so 4 neurónmi, jeden pre každú kategóriu výstupnej množiny
- Aktivačná funkcia na skrytých vrstvách: relu

- Aktivačná funkcia na výstupnej vrstve: softmax
- Použili sme „categorical_crossentropy“
- Miera učenia: 0,0002
- Počet epoch: 20
- Veľkosť dávky: 32
- EarlyStopping nebol nutný

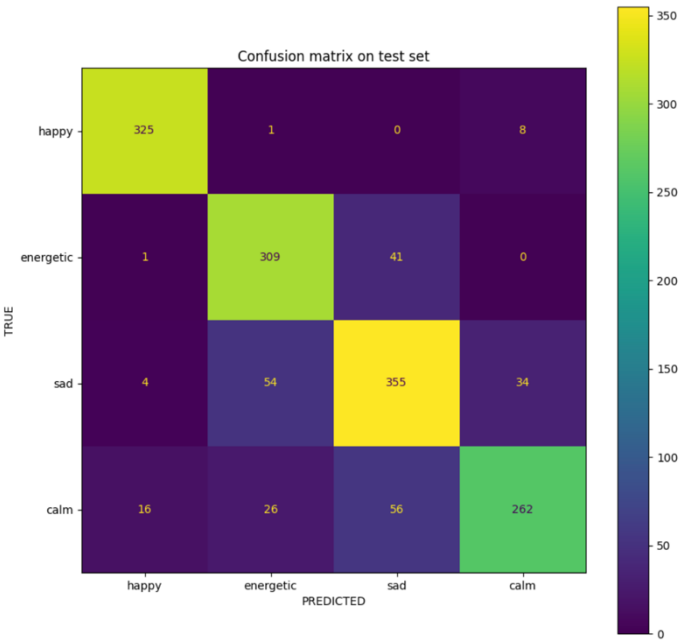
```
# Train MLP model in Keras (overtrain with early stopping)
model = Sequential()
model.add(Dense(24, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(15, activation='relu'))
model.add(Dense(4, activation='softmax'))

model.compile(loss=tf.keras.losses.categorical_crossentropy,
optimizer=Adam(learning_rate=0.0002), metrics=['accuracy'])

history = model.fit(x=X_train, y=y_train, validation_data=(X_valid,
y_valid), epochs=20, batch_size=32, callbacks=[early_stopping])
```

Vyhodnotenie finálnej siete (*použitý Keras*) – *Prototyp*:

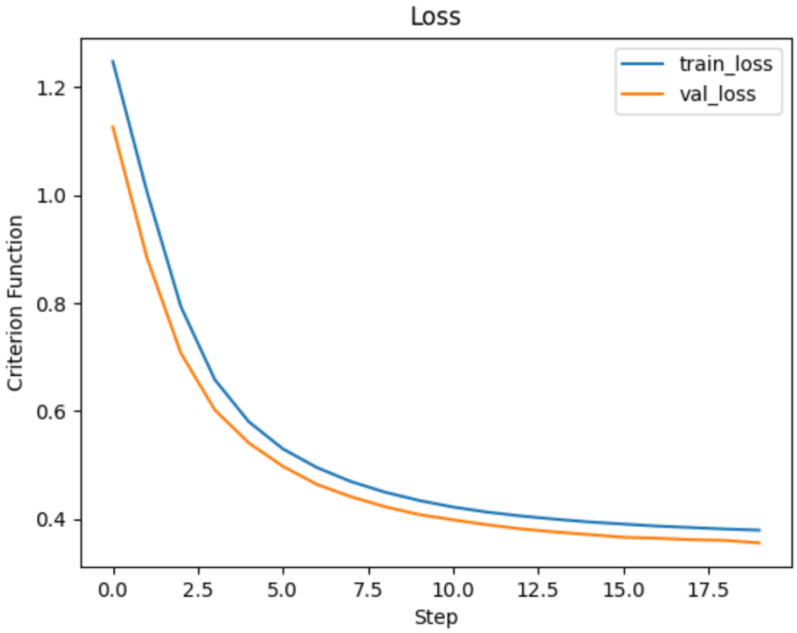
Sieť sme vyhodnotili na základe konfúznych matíc a grafov zobrazujúcich zmeny v presnosti a v stratách počas procesu. Na základe týchto dát sme dospeli k záveru, že náš prototyp môže byť považovaný za funkčne natrénovanú neurónovú sieť. Prototyp dosiahol presnosť na trénovacom sete 84,93% a na testovacom 83,85%



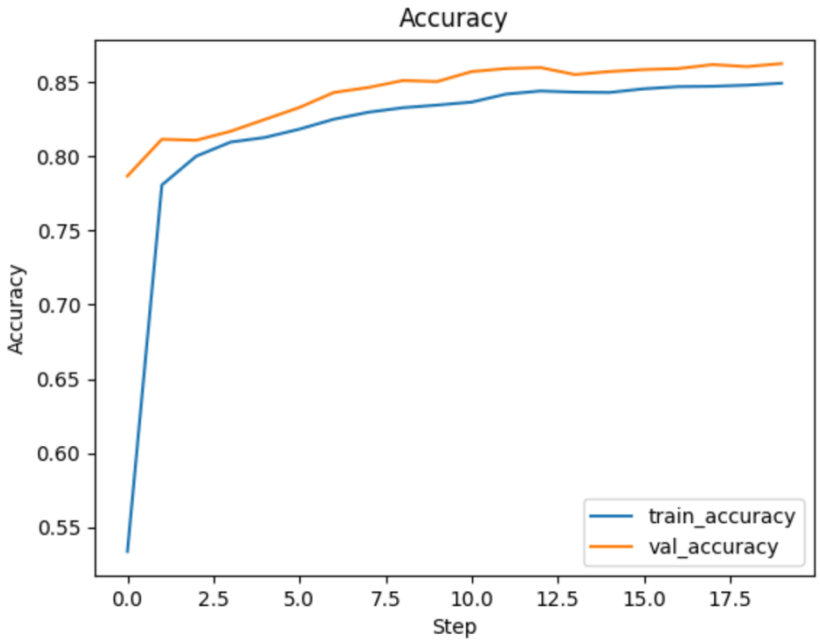
Konfúzna matica prototypu pre trénovacie dáta jednoduchej siete



Konfúzna matica prototypu pre testovacie dáta jednoduchej siete



Prototyp - Priebeh zmien straty počas procesu



Prototyp - Priebeh zmien presnosti počas procesu

Trénovanie finálnej siete (*použitý Keras*) – *Pretrénovanie*:

Po úspešnom dokončení sme vytvorili model, ktorý zabezpečil pretrénovanie siete. Aby sme to dosiahli, zvolili sme parametre nasledovne:

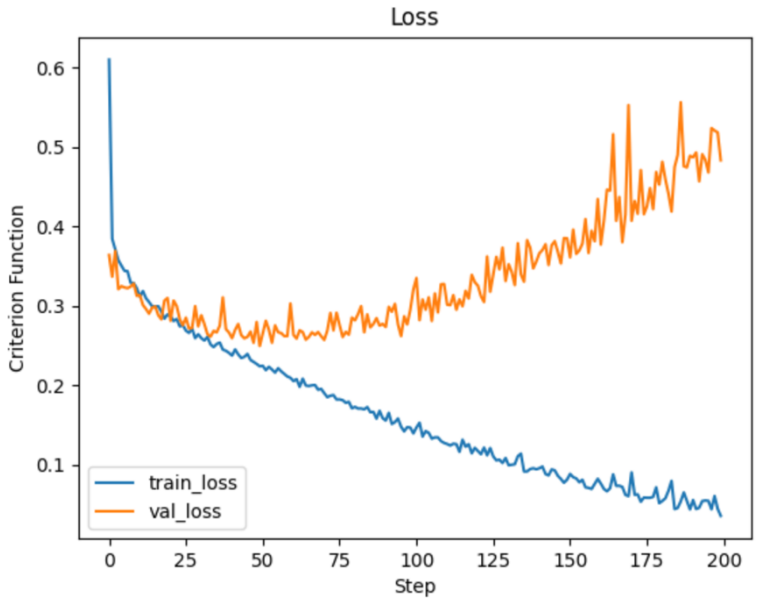
- 4 skryté vrstvy s počtami neurónov: 512, 512, 512, 512
- Výstupná vrstva so 4 neurónmi, jeden pre každú kategóriu výstupnej množiny
- Aktivačná funkcia na skrytých vrstvách: relu
- Aktivačná funkcia na výstupnej vrstve: softmax
- Použili sme „categorical_crossentropy“

- Miera učenia: 0,0002
- Počet epoch: 200
- Veľkosť dávky: 64
- EarlyStopping nebol použitý

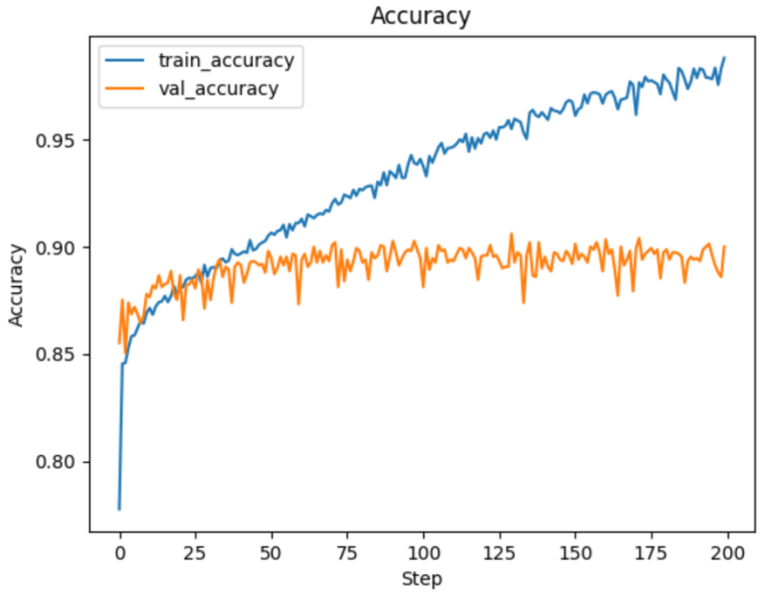
Vyhodnotenie finálnej siete (*použitý Keras*) – *Pretrénovanie*:

Sieť sme opäť vyhodnotili na základe konfúznych matíc a grafov zobrazujúcich zmeny v presnosti a v stratách počas procesu. Na základe zistených údajov sme mohli potvrdiť, že sieť sa nám podarilo pretrénovať. K tomuto záveru nás priviedlo niekoľko faktorov. V prvom rade, rozdiel medzi presnosťou na trénovacom sete (87,87%) a testovacom (98,61%) bol výrazný, vyše 10%. Ďalej na grafe ukazujúcom priebeh zmien strát sme mohli vidieť, že krivka predstavujúca stratu počas validácie, začala v polovici procesu stúpať, čo by nemala. A pri presnosti naopak začala klesať, čo tiež indikuje pretrénovanie.

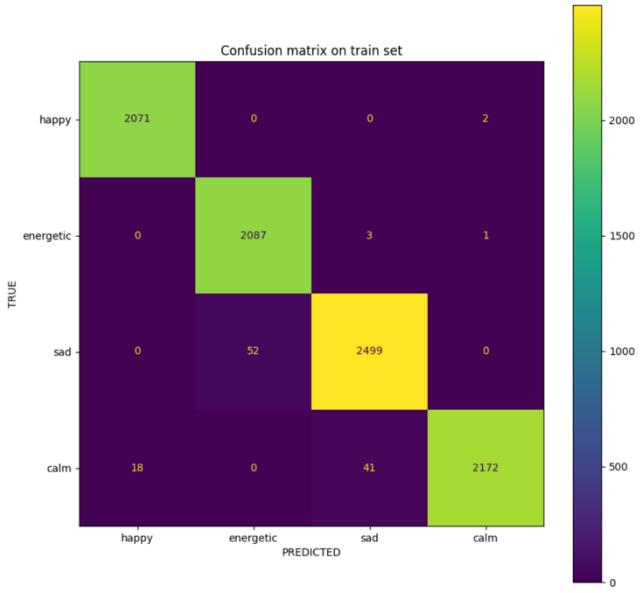
Pretrénovanie najviac ovplyvnil počet skrytých vrstiev a počet neurónov na nich.



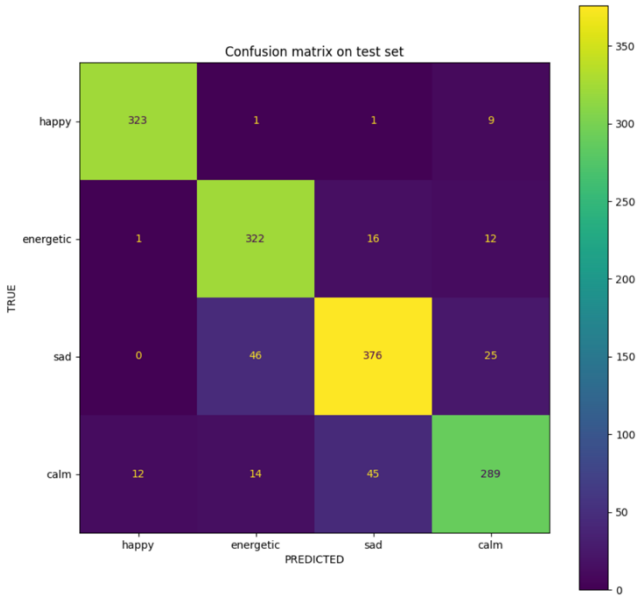
Pretrénovanie - Priebeh zmien straty počas procesu



Pretrénovanie - Priebeh zmien presnosti počas procesu



Konfúzna matica pretrénovaného modelu pre trénovacie dáta jednoduchej siete



Konfúzna matica pretrénovaného modelu pre testovacie dáta jednoduchej siete

Trénovanie finálnej siete (*použitý Keras*) – *EarlyStopping*:

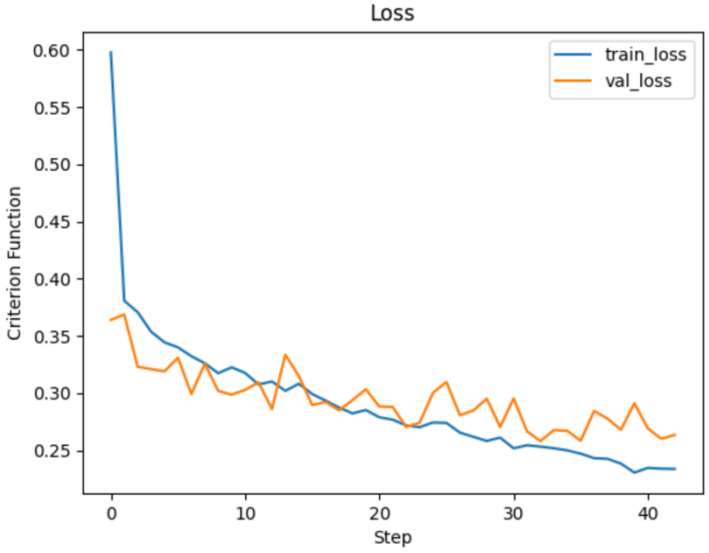
Aby sme odstránili pretrénovanie, bez zmeny zadaných parametrov, zakomponovali sme do procesu EarlyStopping, čo znamená, že sme nastavili sieť tak aby trénovací proces ukončila v optimálnom bode na základe strát počas validácie („val_loss“). Inými slovami, aby bol proces prerušený skôr ako začne hodnota „val_loss“ narastať.

```
# Early stopping definition
early_stopping = EarlyStopping(monitor='val_loss', patience=10, verbose=1,
restore_best_weights=True)

history = model.fit(x=X_train, y=y_train, validation_data=(X_valid,
y_valid), epochs=200, batch_size=64, callbacks=[early_stopping])
```

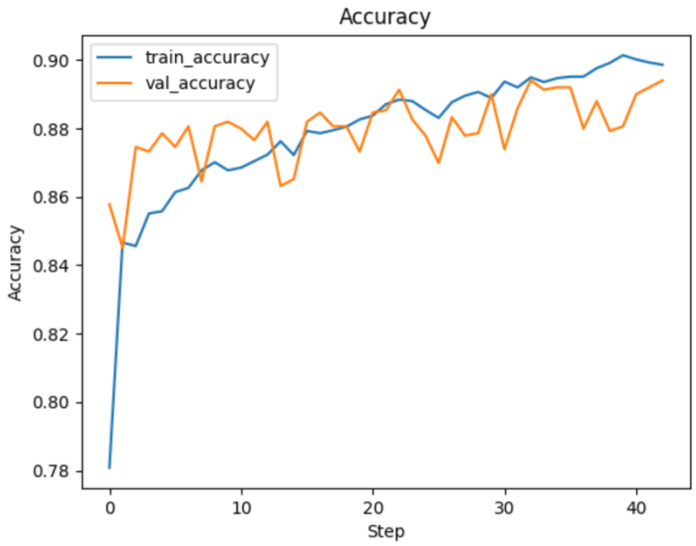
(útržky kódu ukazujú iba časti kódu kde došlo k zmene voči trénovaniu bez EarlyStoppingu.)

Vyhodnotenie finálnej siete (*použitý Keras*) – *EarlyStopping*:

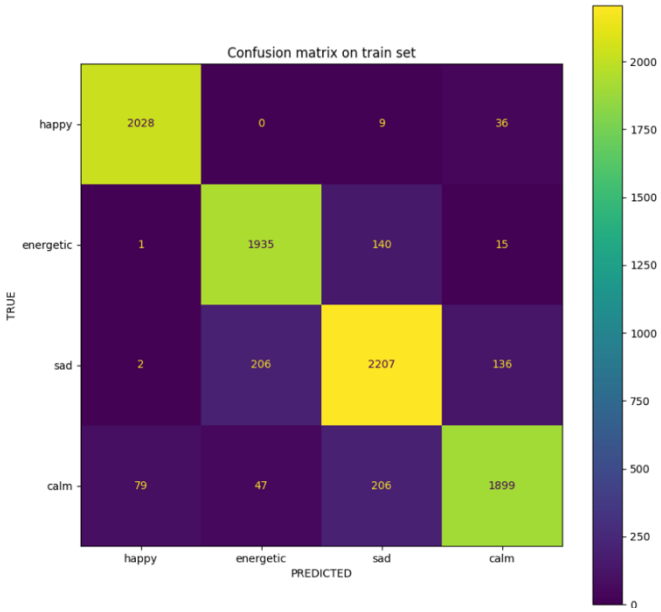
Na základe dát vyčítaných z grafov a konfúznych matíc je jasné, že EarlyStopping prebehol úspešne. Trénovací proces predčasne zastal na 43 epoche. Model dosiahol úspešnosť na trénovacom sete 87,33% a na testovacom 89,86%.



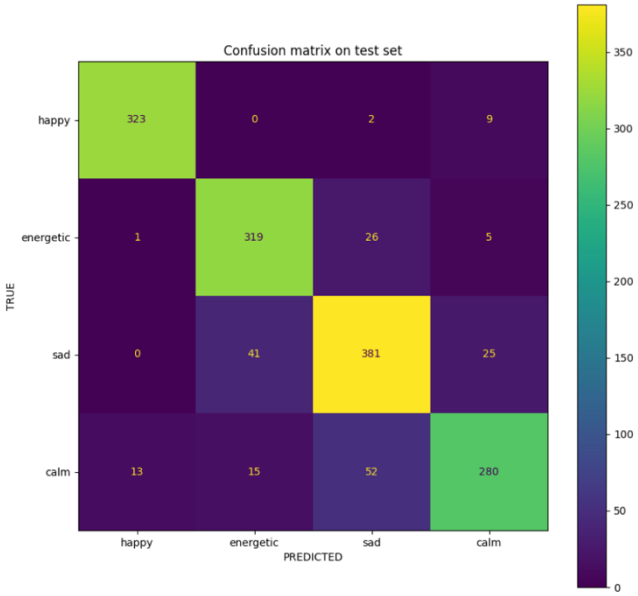
EarlyStopping - Priebeh zmien straty počas procesu



EarlyStopping - Priebeh zmien presnosti počas procesu



Konfúzna matica EarlyStopping modelu pre trénovacie dáta jednoduchej siete



Konfúzna matica EarlyStopping modelu pre testovacie dáta jednoduchej siete

Priložený kód zadania obsahuje práve tento model s pretrénovaním a EarlyStoppingom (keďže sme pre každý model nepísali vlastný kód, iba menili už existujúci)

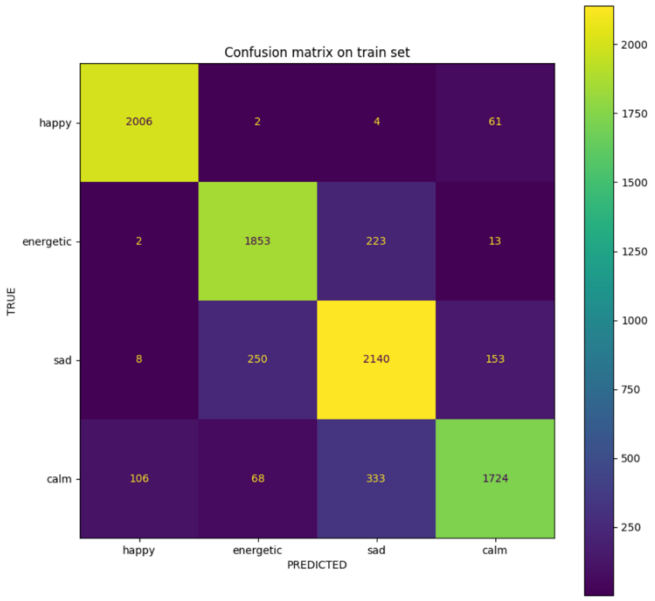
Trénovanie finálnej siete (*použitý Keras*) – *Ďalšie verzie*:

Na záver sme vytvorili ešte 5 ďalších modelov, pričom sme rôzne menili hyper-parametre, za účelom nájsť čo najlepší model. Modely boli vytvárané bez EarlyStoppingu. Naše rôzne nastavenia jednotlivých verzií dokumentuje nasledujúca tabuľka. (vrátane prototypu, pretrénovania a aj EarlyStoppingu)

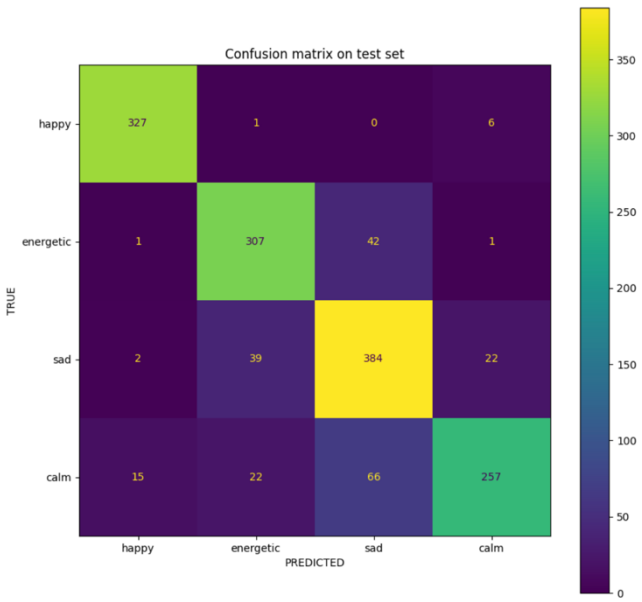
Neurons (Hidden Layers)	Learning rate	Epochs	Batch_size	Test acc	Train acc	Type of attempt
24, 15	0.002	100	64	0.8794	0.8853	1st var
24, 15	0.02	50	32	0.8700	0.8883	2nd var – bad results
24, 15	0.0005	300	64	0.8760	0.8879	3rd var
48, 24, 12	0.0005	300	32	0.8773	0.8963	4th var
256, 256, 256, 128	0.000002	450	64	0.8546	0.8633	5th var
24, 15	0.0002	20	32	0.8385	0.8493	PROTOTYPE
512, 512, 512, 512	0.0002	200	64	0.8787	0.9861	OVERTRAIN
512, 512, 512, 512	0.0002	200 – ES after 43rd epoch	64	0.8733	0.8986	EarlyStopping

Vyhodnotenie finálnej siete (*použitý Keras*) – *Najlepší model*:

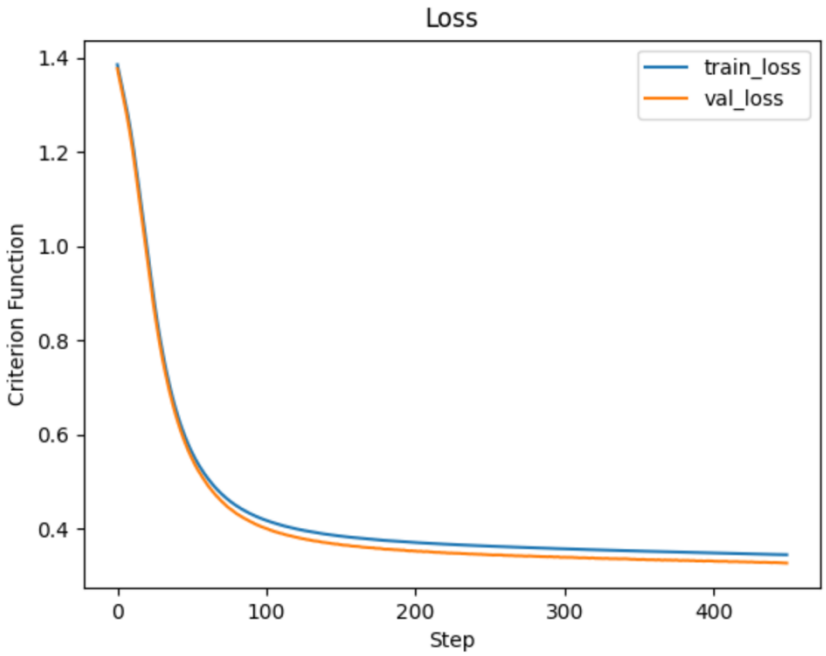
Pri finálnom vyhodnotení výberu najlepšieho modelu sme brali do úvahy výsledné presnosti, konfúzne matice, grafy priebehov trénovaní a dĺžky trénovaní. Najlepšiu presnosť dosiahol model s EarlyStoppingom a naša štvrtá variácia, avšak tieto modely nepovažujeme za najlepšie keďže grafické zobrazenie priebehu trénovania poukazuje, že sa nejedná o ideálne nastavenia parametrov (pri EarlyStoppingu to bolo príliš veľa neurónov na skrytých vrstvách a pri 4. variácií to bola vysoká miera učenia). Podobne nevhodné sú aj prvá a druhá variácia (vysoká miera učenia). Prototyp mal pekný priebeh, ale najnižšiu presnosť, preto sme za najlepší model určili piatu variáciu. (v tabuľke orámované červenou)



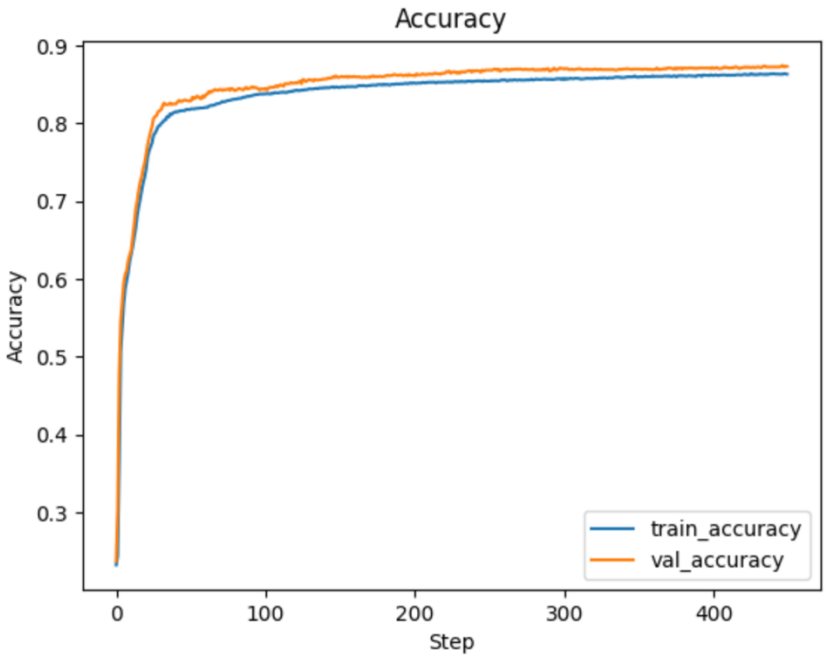
Konfúzna matica najlepšieho modelu pre trénovacie dáta jednoduchej siete



Konfúzna matica najlepšieho modelu pre testovacie dáta jednoduchej siete



Najlepší model - Priebeh zmien straty počas procesu



Najlepší model - Priebeh zmien presnosti počas procesu

ZÁVER

Pri tomto zadaní sme sa oboznámili s neurónovými sieťami, ich trénovaní ako aj vyhodnocovaním výsledkov a určovaním kvality modelu. Taktiež sme sa naučili pracovať so vstupnými dátami. Osvojili sme si ako takéto dáta upraviť aby boli použiteľné pre našu sieť. V neposlednom rade sme si vyskúšali EDA techniky pre odhalenie závislostí medzi jednotlivými kategóriami.

Výsledkom nášho zadania je neurónová sieť v jazyku Python, ktorá dokáže kategorizovať piesne z platformy Spotify do kategórií podľa nálady, na základe poskytnutých parametrov skladieb.

ZDROJE

Tak ako sú uvedené v priloženom zdrojovom kóde:

```
# RESOURCES
    # guide for the whole process: codes from seminars
(seminar2.py, sem3.py)

    # help with early stopping:
https://keras.io/api/callbacks/early_stopping/

    # help with one-hot encoding:
https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-
in-python/

    # additional help: keras documantation and stackoverflow
```