**UNIVERZITA SV. CYRILA A METODA V TRNAVE**

**FAKULTA PRÍRODNÝCH VIED**

# MOŽNOSTI STROJOVÉHO UČENIA PRE MOBILNÉ APLIKÁCIE

**Bakalárska práca**

**2022** **Mgr. Július Selnekovič**

**UNIVERZITA SV. CYRILA A METODA V TRNAVE**

**FAKULTA PRÍRODNÝCH VIED**

# MOŽNOSTI STROJOVÉHO UČENIA PRE MOBILNÉ APLIKÁCIE

**Bakalárska práca**

**Študijný program:** aplikovaná informatika
**Študijný odbor:** 18 - Informatika
**Školiace pracovisko:** Katedra aplikovanej informatiky
**Školiteľ:** doc. Ing. Michal Čerňanský, PhD.

**2022** **Mgr. Július Selnekovič**


UCM Trnava
Fakulta prírodných vied

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Mgr. Július Selnekovič
**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., externá forma)
**Študijný odbor:** 18. - informatika
**Typ záverečnej práce:** Bakalárska práca
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** Možnosti strojového učenia pre mobilné aplikácie

**Anotácia:**
1. Zoznámte sa s možnosťami technológií strojového učenia.
2. Analyzujte a overte možnosti ich použitia pri tvorbe mobilných aplikácií.
3. Navrhnite a implementujte aplikáciu, využite zvolené technológie.
4. Vytvorené riešenie overte a zhodnoťte.

**Vedúci:** doc. Ing. Michal Čerňanský, PhD.
**Katedra:** KAI - Katedra aplikovanej informatiky
**Vedúci katedry:** PaedDr. Mgr. Miroslav Ölvecký, PhD.

**Dátum zadania:** 07.12.2020

**Dátum schválenia:** 08.12.2020

PaedDr. Mgr. Miroslav Ölvecký, PhD.
vedúci katedry

## ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že na bakalárskej práci s názvom Možnosti strojového učenia pre mobilné aplikácie som pracoval samostatne pod vedením môjho vedúceho práce a prácu som vypracoval na základe vlastných teoretických a praktických vedomostí, konzultácií, štúdiom odbornej literatúry, ktorej všetky zdroje som uviedol v zozname použitej literatúry.

Trnava, 12. máj 2022 .....

Július Selnekovič

## POĎAKOVANIE

Ďakujem môjmu vedúcemu práce – doc. Ing. Michalovi Čerňanskému, PhD. za jeho cenné rady, usmernenia, odovzdané vedomosti a čas, ktorý mi venoval.

## Abstrakt

SELNEKOVIČ, Július: *Možnosti strojového učenia pre mobilné aplikácie*. [Bakalárska práca]. Univerzita Sv. Cyrila a Metoda v Trnave. Fakulta prírodných vied. Katedra aplikovanej informatiky. Školiteľ: doc. Ing. Michal Čerňanský, PhD. Trnava, 2022. Počet strán práce 39 s.

Bakalárska práca sa zaoberá využitím strojového učenia v oblasti mobilných aplikácií. Môže slúžiť ako rýchly úvod pre vývojárov mobilných aplikácií, ktorí by chceli do svojich aplikácií zaradiť strojové učenie. Popisuje základnú terminológiu, technológie strojového učenia a ich možnosti využitia. Hlavným cieľom práce je ukázať vytvorenie a implementáciu ML modelu do mobilnej aplikácie pre systémy iOS.



**Kľúčové slová:** Strojové učenie, Hlboké učenie, Klasifikácia zvukov, Mobilné aplikácie, TensorFlow, Create ML, Swift, iOS.

## Abstract

SELNEKOVIČ, Július: *Machine learning options for mobile applications*. [Bachelor thesis]. University of ss. Cyril and Methodius in Trnava. Faculty of Natural Sciences, Department od Applied Informatics. Supervisor: doc. Ing. Michal Čerňanský, PhD. Trnava, 2020. Number of pages 39 p.

The bachelor thesis deals with the use of machine learning in the field of mobile applications. It can serve as a quick introduction for mobile application developers who would like to include machine learning in their applications. Theses describes the basic terminology, technologies of machine learning and their possibilities of use. The main goal of the work is to show training and implementation of ML model into a mobile application for iOS systems.

**Keywords:** Machine learning, Deep learning, Sound classification, Mobile applications, TensorFlow, Create ML, Swift, iOS.

# OBSAH

# Úvod

Strojové učenie (ML) je nová elektrina. Tvrdenie, ktoré možno počuť v tejto alebo obdobnej podobe od mnohých IT odborníkov, skrýva v sebe mnoho pravdivého. Stretávame sa s ním na dennej báze, je súčasťou nášho života, hoci si to ani nemusíme uvedomovať. Pomocou neho odomykáme tvárou telefón, keď fotografujeme, môžeme si byť skoro istí, že upravilo našu fotografiu, aby vyzerala lepšie, ponúka nám hudbu, filmy, príspevky, ktoré by sa nám mohli páčiť alebo sa pomocou neho môžeme rozprávať s našimi smart zariadeniami - “OK Google”, “Alexa”, “Hey, Siri”. Strojové učenie bude čoskoro všade, bude súčasťou ďalšej generácie inteligentných zariadení či už v našich domácnostiach, v práci alebo vo verejnom priestore. Ak vezmeme do úvahy, koľko elektronických zariadení máme neustále okolo seba a koľko nespracovaných dát generujú ich senzory (zvukové, obrazové, pohybové, priestorové, biometrické), uvedomíme si obrovské možnosti, ktoré sa pre aplikácie strojového učenia otvárajú. Odhaduje sa, že v roku 2021 bolo viac ako 10 biliónov IoT zariadení pripojených do internetu, čo je zhruba polovica všetkých aktívnych zariadení. Predpokladá sa, že do roku 2025 bude ich počet dvojnásobný. A to je stále iba časť z celkového počtu zariadení, ktoré by mohli využívať strojové učenie. Môžeme k nim prirátať viac ako 6 milárd smartfónov používaných vo svete práve v tejto chvíli. „*Takmer každý dospelý človek nosí so sebou v súčasnosti nejaké zariadenie a odhaduje sa, že sa na svoje smartfóny pozeráme aspoň 50-krát denne, či už je to potrebné alebo nie. Tieto stroje ovplyvňujú naše každodenné rozhodovanie.*“ [1]

Možnosti strojového učenia pre mobilné aplikácie sú teda skutočne široké. Majú však aj svoje špecifiká, na ktoré je potrebné myslieť už pri návrhu aplikácií využívajúcich strojové učenie. Jedným z nich je, že mobilné zariadenia fungujú na batérie. Pre vývojárov to predstavuje určité obmedzenie. Hoci sa môže na prvý pohľad zdať, že niektoré súčasné smartfóny majú hardvérové parametre porovnateľné so stolovými počítačmi, o spracovanie vstupných dát sa častokrát nestará priamo CPU, ale dedikovaný mikrokontrolér s nižším výkonom. Vyššie zaťaženie CPU alebo GPU dokáže relatívne rýchlo spotrebovať kapacitu batérie, preto ML modely v aplikáciách mobilných zariadení musia byť jednoduchšie, menej náročné na výpočtový výkon, ale s dostatočnou presnosťou v porovnaní s modelmi používanými napríklad na serveroch. Redukciou parametrov, kvantizáciou, znížením hĺbky alebo šírky neurónovej siete môžeme natrénovať model, ktorý má nielen nižšiu výpočtovú

náročnosť, ale dokážeme ho spustiť aj na  mikrokontroléroch s veľmi nízkymi nárokmi na spotrebu elektrickej energie.

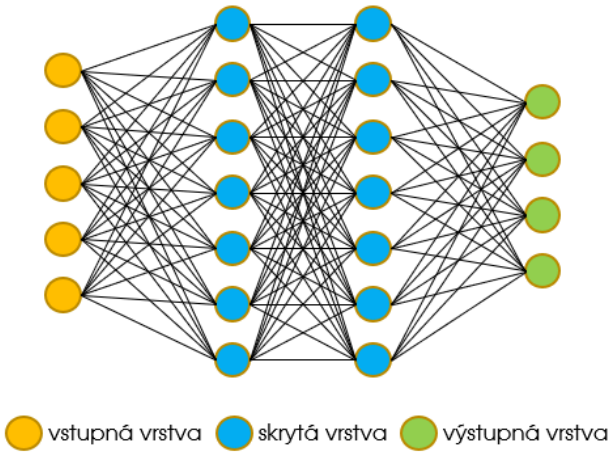
Vyššie spomenuté úskalia strojového učenia na mobilných zariadeniach by pri prvom pohľade mohlo riešiť odosielanie dát na spracovanie do datacentier. Majú dostatočnú výpočtovú silu a používajú robustné ML modely s lepšou predikčnou schopnosťou. Toto riešenie má však tiež svoje slabé stránky. Nie vždy máme k dispozícii internetové pripojenie alebo telefónny signál s dobrou rýchlosťou prenosu dát. V súčasnosti by bolo užívateľsky neakceptovateľné, keby sme mali čakať niekoľko desiatok sekúnd na detekciu orientačného bodu tváre na ukotvenie AR masiek, ktoré sa využívajú pri filtroch v aplikácii Snapchat. Nesmieme pri tom opomenúť ani súkromnie a bezpečnosť odosielaných dát. Určite by sa užívateľom iPhonov nepáčilo, keby každý sken ich tváre pri odomykaní telefónu bol zasielaný na analýzu spoločnosti Apple. Aj z tohto dôvodu sa do spoločnosti neposielajú žiadne údaje o tvári a rozpoznávanie sa uskutočňuje výlučne na mobilnom zariadení.

Hoci je oblasť strojového učenia veľmi rozsiahla a komplexné zvládnutie náročné, vytvorenie a použitie jednoduchého ML modelu v mobilnej aplikácii v súčasnosti zvládne aj skúsenejší programátor. Prácu sme preto rozdelili do štyroch častí, ktoré by mali čitateľovi pomôcť postupne preniknúť do sveta strojového učenia. V prvej časti sa oboznámime so základnými pojmami používanými v oblasti strojového učenia. V druhej časti si popíšeme vybrané nástroje (knižnice a rámce) používané pri tvorbe a trénovaní ML modelov. Príklady použitia strojového učenia v mobilných aplikáciách si predstavíme v tretej časti práce. Rozpoznávanie tváre, hlasu alebo odporúčacie algoritmy patria k rozšíreným spôsobom využitia strojového učenia všeobecne. V štvrtej časti využijeme poznatky z predošlých častí a názorne predvedieme vytvorenie vlastného ML modelu a jeho implementovanie do aplikácie v prostredí iOS. V našom prípade to bude model rozpoznávajúci druhy vtákov podľa ich spevu.

# 1. Strojové učenie

V nasledujúcom texte sa pokúsime osvetliť základné pojmy, s ktorými sa záujemca o strojové učenie bezpochyby stretne. Nenárokujeme si na presné a vyčerpávajúce definície, pristúpili sme k tomu pragmaticky, preto niekedy uprednostňujeme zrozumiteľnosť a zjednodušenie na úkor exaktnosti. Tento pomyselný slovník pojmov nie je zoradený abecedne, snaží sa postupne odkrývať pojmy a vzťahy, s ktorými sa stretneme pri vstupe do problematiky strojového učenia. V zátvorke vždy uvádzame anglický ekvivalent pojmu, prípadne skratku, ktorá sa používa v odbornej literatúre.

- **umelá neurónová sieť (artificial neural network / ANN)** - vychádza z analógie ľudského mozgu pozostávajúceho z miliárd neurónov, ktoré medzi sebou komunikujú prostredníctvom elektrických a chemických signálov. "*Umelé neurónové siete sú populárne techniky strojového učenia, ktoré simulujú mechanizmus učenia v biologických organizmoch.*" [2] Umelá neurónová sieť matematicky napodobňuje správanie neurónov, preto ju treba chápať ako počítačový algoritmus, nie ako biologickú štruktúru. Základom siete sú neuróny/uzly, môžeme si ich predstaviť ako výpočtové jednotky. Sú vzájomne prepojené, skladajú sa do vrstiev, čím vytvárajú šírku a hĺbku neurónovej siete.




Obrázok 1 Model umelej neurónovej siete s plne prepojenými skrytými vrstvami (prevzaté z https://umelainteligencia.sk/uvod-do-neuronovych-sieti)

Umelé neurónové siete môžeme rozdeliť na:

- *dopredné (feedforward neural network / FNN)* - jednoduchší typ siete, kde spojenia medzi neurónmi netvoria slučku, informácie sa pohybujú iba jedným smerom od vstupu k výstupu.
- *rekurentné (recurrent neural network / RNN)* - niektoré spojenia medzi neurónmi tvoria slučku, čo umožňuje spracovanie sekvencií vstupov alebo časových sérií. Používa sa napríklad pri rozpoznávaní zvukov alebo videí.

• **umelá inteligencia (artificial intelligence / AI)** - je interdisciplinárna vedecká oblasť, ktorá zastrešuje napr. robotiku, strojové učenie, expertné systémy, počítačové rozpoznávanie reči, počítačové videnie alebo techniky spracovania prirodzeného jazyka. Skúma možnosti využitia počítačov a strojov na napodobňovanie schopností ľudskej mysle učiť sa, rozhodovať a riešiť problémy.


Obrázok 2 Umelá inteligencia a jej podoblasti

• **strojové učenie (machine learning / ML)** - býva často chybne zamieňané za pojem umelá inteligencia. Strojové učenie je iba podmnožinou umelej inteligencie. Zaoberá sa technikami autonómneho učenia počítačov prostredníctvom dát a skúseností. Časť algoritmov strojového učenia je založená na štatistických metódach, pomocou ktorých umelá neurónová sieť vytvára klasifikácie alebo predpovede zo vstupných dát. Strojové učenie je preto dôležitou súčasťou dátovej vedy.

- **hlboké učenie (deep learning / DL)** - je špeciálnou oblasťou strojového učenia založenou na trénovaní hlbokých neurónových sietí prostredníctvom veľkého množstva údajov ako sú obrázky, hudba alebo text. Slovo "*hlboké*" odkazuje na siete a učenie, pri ktorom sa využívajú umelé neurónové siete s väčším počtom skrytých vrstiev. Hlboké učenie automatizuje veľkú časť procesu extrakcie príznakov (skúmaných vlastností objektu), čím sa odlišuje od klasického strojového učenia.


Obrázok 3 Rozdiel medzi strojovým a hlbokým učením

Názornejšie to ilustruje obr. 3. Pri strojovom učení si vyberieme trénovaný model a manuálne zadáme príznaky, na základe ktorých sa bude model učiť klasifikovať mačky, psy a myši - napr. výška, farba alebo veľkosť chvosta. Pri hlbokom učení si zvolíme iba architektúru siete, ktorá si automaticky extrahuje príznaky (features) z označených trénovacích dát (labels).

- **učenie s učiteľom (supervised learning)** - predstavuje v súčasnosti najpoužívanejší typ strojového učenia. "*Predpokladom pre jeho použitie je existencia dostatočne veľkého trénovacieho datasetu, ktorý obsahuje správne označené dvojice vstup – výstup. Na týchto dátach sa potom algoritmus natrénuje tak, aby vedel pre daný vstup vyprodukovať korektný výstup.*" [3a] Je dôležité zdôrazniť, že vstupné dáta musia byť označené, prípadne rozdelené do správnych kategórií. Učenie pozostáva z opakovania výpočtov nad každou sadou vstupov a porovnaním výstupu algoritmu s požadovaným/ správnym výsledkom. Rozdiel medzi výstupom siete a požadovaným výstupom sa

používa na nastavenie parametrov algoritmu tak, aby sa výstup algoritmu priblížil (alebo sa rovnal) správnemu výstupu.

- **učenie bez učiteľa (unsupervised learning)** - sa líši od učenia s učiteľom v tom, že algoritmus nemá žiadne informácie o kategórii priradenej ku každej položke v cvičnej sade - nepozná správne odpovede, označenia (labels). Algoritmy učenia bez učiteľa sa preto používajú v prípadoch, kde chceme objaviť nové vzory v existujúcich údajoch. Vďaka schopnosti odhaliť podobnosti a rozdiely v informáciách sú ideálnym riešením pre analýzu dát, stratégie krížového predaja, segmentáciu zákazníkov ale aj rozpoznávanie objektov.

- **učenie formou odmeňovania (reinforcement learning)** - je prístup, ktorý používa techniky podobné tým, akými sa učia aj ľudia - interakciou s prostredím a spájaním pozitívnych a negatívnych odmien s rôznymi činnosťami. Pri učení formou odmeňovania "*... na trénovanie modelu nepoužijeme žiadne označené, či neoznačené trénovacie príklady. Učenie tu prebieha tak, že vytvoríme systém – agenta, ktorého nasadíme do prostredia a necháme ho, nech sa učí prostredníctvom interakcie s prostredím.*" [3b] Agent má určitú množinu definovaných akcií, ktoré môže v danom okamihu vykonať, pričom s každou je spojená odmena alebo trest. Cieľom systému je maximalizovať celkovú odmenu za sériu rozhodnutí/akcií. Znalosti získané agentom tvoria akúsi politiku určujúcu činnosti, ktoré by mal agent vykonať, ak sa dostane do obdobnej situácie. "*Modelovým príkladom pre učenie formou odmeňovania je hra šach, kde vytvoríme agenta, definujeme mu povolené ťahy a pravidlo pre výhru. Odmeníme ho, ak vyradí súperovu figúrku alebo vyhrá, potrestáme ho, ak je vyhodená jeho figúrka alebo prehrá. Následne ho necháme, nech si zahrá sám proti sebe niekoľko (miliónov) partií. Výsledkom je umelá inteligencia, ktorú neporazia ani najväčší šachoví veľmajstri.*" [3b]

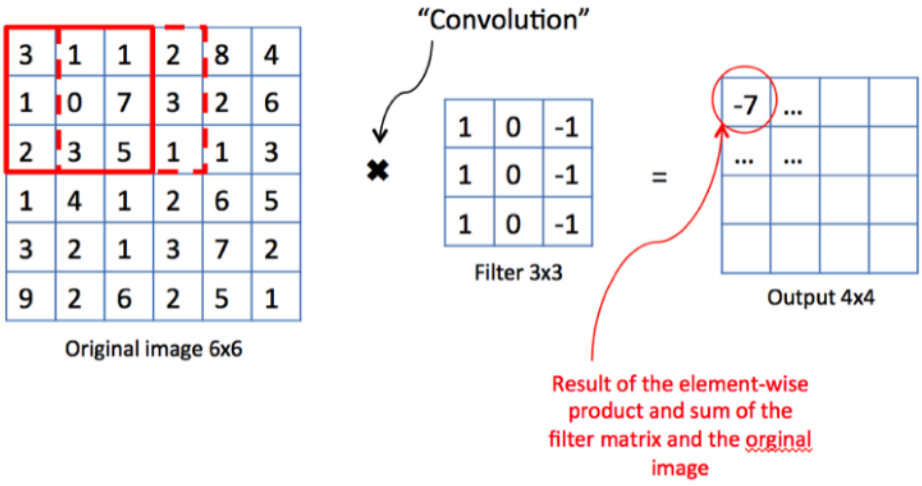
- **klasifikácia (classification)** - je proces rozdeľovania štrúktúrovaných (napr. csv, xml, json) alebo neštruktúrovaných (napr. multimediálne súbory) dát do tried. Klasifikačné algoritmy môžu triediť dáta do dvoch alebo viacerých kategórií. Matematicky ide o funkciu, ktorá zo vstupnej premennej (napr. email) predikuje, s akou pravdepodobnosťou patrí do určených skupín (spam - nie spam). Medzi klasifikačné algoritmy patrí napríklad logistická regresia, K-najbližší susedia (K-nearest

neighbors / KNN), rozhodovacie stromy, naivný Bayesov klasifikátor (Naive Bayes), podporné vektorové stroje (Support Vector Machines / SVG) alebo aj umelé neurónové siete.

- **klastrovanie/zhlukovanie (clustering)** - patrí medzi techniky učenia bez učiteľa. Používa neoznačené dáta, takže klasifikácia údajov do klastrov/skupín pri učení prebieha na základe podobných vlastností bez znalosti toho, do akej skupiny dáta patria. Keďže sa model snaží sám prísť na to, na základe akých vlastností najlepšie rozdeliť danú množinu do skupín, nemusí pri klastrovaní existovať správne/nesprávne riešenie. Rôzne algoritmy môžu dáta rozdeliť údaje do rôznych klastrov, preto je hodnotiacim kritériom skôr použiteľnosť modelu ako jeho presnosť. Medzi klastrovacie algoritmy patrí napríklad K-means (zhlukovanie metódou najbližších stredov), Gausian Mixture Model (zmes Gaussových modelov), BIRCH (Balance Iterative Reducing and Clustering using Hierarchies), DBSCAN (Density-Based Spatial Clustering of Applications with Noise)
- **vrstvy (layers)** - určujú architektúru modelu neurónovej siete, sú jej stavebnými blokmi. Vrstvy majú formu matíc/tenzorov, ktoré obsahujú údaje a výpočty, s ktorými neurónová sieť pracuje. V hlbokom učení sa najčastejšie stretneme s nasledujúcimi typmi:
  - *vstupná vrstva (input layer)* - stojí na začiatku neurónovej siete. Ako napovedá jej názov, slúži na vstup počiatočných údajov do siete. Je pasívna, čo znamená, že údaje v tejto vrstve sa nemenia.
  - *skrytá vrstva (hidden layer)* - všeobecne označuje všetky vrstvy medzi vstupnou a výstupnou vrstvou. Skladá sa z neurónov, tzv. uzlov (nodes). Počet uzlov v jednotlivých skrytých vrstvách určuje šírku neurónovej siete. Počet skrytých vrstiev v neurónovej sieti určuje hĺbku siete. Nazývame ju "*skrytou*", pretože skutočné hodnoty jej uzlov nie sú v tréningovom datasete údajov známe.
  - *výstupná vrstva (output layer)* - je poslednou vrstvou v neurónovej sieti a predstavuje výsledok jej výpočtov, predikciu. V závislosti od typu modelu môže ísť o jednu predikovanú hodnotu (číslo) alebo rozdelenie pravdepodobnosti medzi viaceré výstupy reprezentujúce určité kategórie, ako v prípade spam - nie spam.

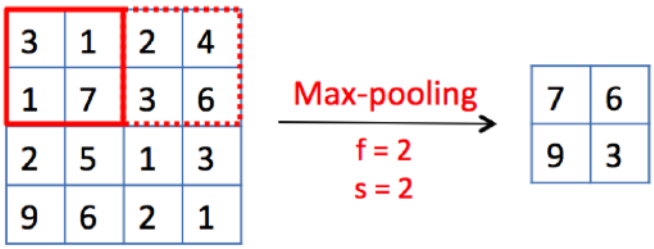
- *plne prepojená vrstva (fully-connected layer, dense layer)* - patrí medzi najpoužívanejšie skryté vrstvy v neurónových sieťach. Neuróny v tejto vrstve sú prepojené s každým neurónom z predchádzajúcej vrstvy.

- *konvolučná vrstva (convolution layer)* - používa sa v konvolučných neurónových sieťach pri spracovaní vizuálnych dát. Jej princípom je aplikácia filtra, najčastejšie veľkosti 5x5 alebo 3x3, ktorý počíta skalárny súčin medzi filtrom a vstupom. Táto matematická operácia sa nazýva konvolúcia. Pomocou konvolučných vrstiev sa napriklad z fotografie najprv extrahujú jednoduchšie vlastnosti ako línie a obrysy, ktoré sa postupne skladajú do zložitejších tvarov ako ústa, nos, oči.




Obrázok 4 Výpočet konvolúcie v CNN sieťach - (prevzaté z https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524)

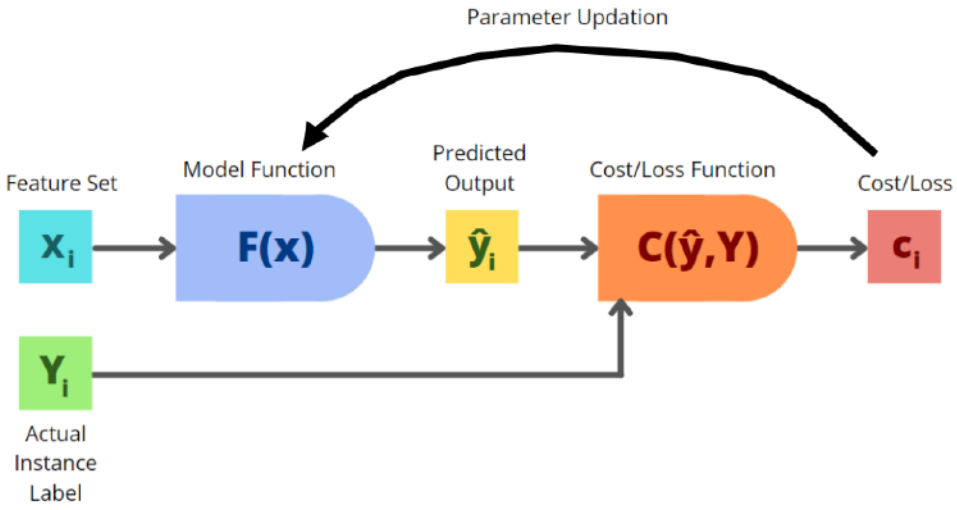
- *pooling vrstva (pool layer)* - ďalšia vrstva, ktorá sa často používa v konvolučných sieťach. Jej úlohou je redukovať veľkosť konvolučnej vrstvy tak, aby boli zachované dominantné vlastnosti dát. Poolingový filter má podobne ako




Obrázok 5 Algoritmus Max-pooling (prevzaté z https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524)

konvolučný filter veľkosť (size / f) a posun (stride / s). Na obrázku XY je znázornený poolingový algoritmus Max-pooling, ktorý z filtrovanej oblasti vyberie maximálnu hodnotu.

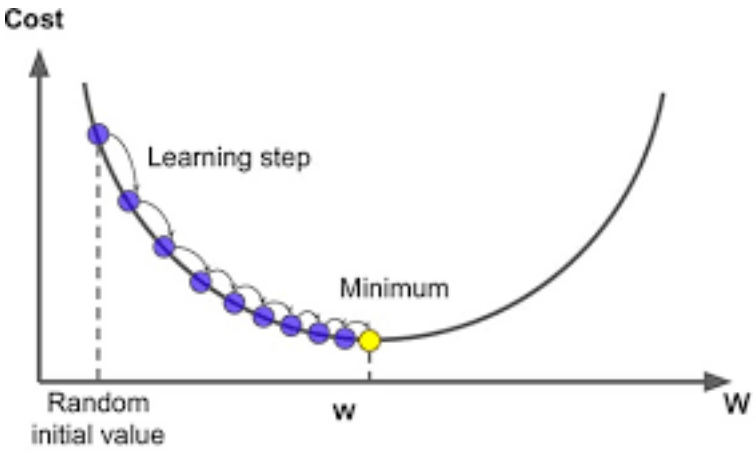
- **hyperparametre (hyperparamaters)** - sú parametre riadiace proces učenia. Oproti ostatným parametrom (napr. váhy, bias) nie sú výsledkom trénovania neurónovej siete. Ich hodnotami ovplyvňujeme komplexnosť, kvalitu a rýchlosť učenia modelu. Mnoho hyperparametrov je špecifických pre jednotlivé typy modelov, no vo všeobecnosti je to šírka a hĺbka modelu, rýchlosť učenia a počet epoch.
- **váhy (weights)** - patria medzi parametre neurónovej siete. Každý príznak má váhu, ktorá vypovedá o jeho dôležitosti. Príznaky s nulovou váhou nemajú v modeli žiadnu funkciu. Cieľom trénovania modelu je nájsť optimálnu váhu pre každý príznak.
- **bias (bias)** - má v strojovom učení viacero významov. V matematickom význame vyjadruje posun hodnoty od stredu súradnicovej sústavy. Základom výpočtov v strojovom učení je rovnica *predikcia = bias + váha * príznak*, ktorá vychádza z parametrickej rovnice priamky používanej v klasickej euklidovskej geometrii.
- **chyba (cost/loss)** - vyjadruje mieru, ako sa predpovede modelu líšia od správnych označení. Na jej výpočet sa používajú rôzne funkcie, pri lineárnej regresii je to najčastejšie stredná kvadratická chyba (mean square error / MSE), ktorú vypočítame tak, že sčítame všetky rozdiely skutočnej a predikovanej hodnoty umocnené na druhú a spriemerujeme - $MSE = (1/n) * \Sigma(\text{skutočná} - \text{predikovaná})^2$. Pri klasifikačných úlohách sa často používa krížová entropia (cross-entropy function) určujúca rozloženie pravdepodobnosti.
- **spätné šírenie chyby (backpropagation)** - sa používa ako všeobecný pojem pre algoritmy učenia neurónových sietí, kedy sa pomocou derivácií upravujú hyperparametre od výstupu k vstupu siete. Úzko súvisí s optimalizáciou.
- **optimalizácia (optimization)** - je proces opakovanej úpravy hyperparametrov (váh a bias-ov) za účelom minimalizácie chybovej funkcie. Robí sa pomocou jednej z optimalizačných techník. Medzi najpoužívanejšie patria zostup po gradient (Gradient Descent) a jeho variácie, ADAM (Adaptive Moment Estimation), RMSProp (Root Mean Square Propagation).




Obrázok 6 Priebeh učenia a optimalizácie siete (prevzaté z https://towardsdatascience.com/the-hitchhikers-guide-to-optimization-in-machine-learning-edcf5a104210)

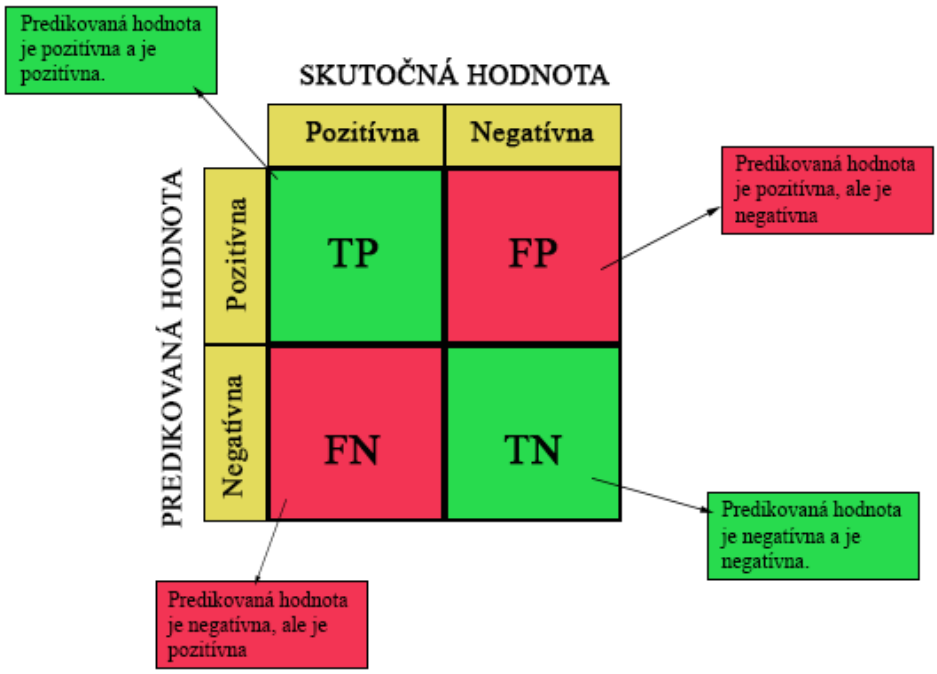
- **zostup po gradiente (gradient descent)** - je základnou metódou optimalizácie. Jej cieľom je minimalizovanie chybovosti modelu prostredníctvom nájdenia ideálnych hodnôt váh. Počíta sa pomocou derivácií a určuje smer zmeny parametru, teda či jeho hodnotu zvýšime alebo znížime. Pokiaľ sa počíta na celej množine trénovacích dát, hovoríme o deterministickom gradiente. V praxi sa skôr využíva stochastický zostup po gradiente (stochastic gradient descent / SGD) počítaný na jednotlivých vzorkách alebo dávkach dát.

- **rýchlosť učenia (learning rate)** - je hodnota posunu hodnoty váhy v smere gradientu. Táto hodnota môže byť konštantná počas celého trvania učenia alebo sa môže v priebehu učenia meniť (zvyčajne klesať). Veľkosť tejto hodnoty môže byť napríklad 0,001, 0,005 alebo 0,01.




Obrázok 7 Rýchlosť učenia pri hľadaní optimálnej váhy (prevzaté z https://medium.com/@shulavkarki88/gradient-descent-for-linear-regression-44b30160396b)

- **dávka (batch)** - určuje počet trénovacích príkladov v jednej iterácii. Patrí medzi hyperparametre. Môže označovať celú tréningovú sadu, ale častejšie sa používa pojem mini-dávka (mini-batch), ktorý označuje určitý počet príkladov z celkovej tréningovej množiny.
- **iterácia (iteration)** - vyjadruje, koľkokrát sa aktualizujú hodnoty váh v jednej epoche. Ak veľkosť dávky = 1, počet iterácií, resp. prechodov cez algoritmus, bude rovnaký ako počet tréningových príkladov.
- **epocha (epoch)** - označuje úplný prechod cez celý súbor tréningových dát tak, aby každý príklad bol videný raz. Epocha sa skladá z iterácií. Ak si vezmeme dataset so 100 príkladmi a určíme veľkosť dávky 20, jedna epocha bude obsahovať 5 iterácií.
- **aktivačná funkcia (activation function)** - sa v neurónových sieťach používa ako prvok nelineárnej transformácie hodnôt neurónov. Aplikuje sa na výstup jednotlivých uzlov a jej hodnota závisí od použitej funkcie, zvyčajne je to medzi -1 a 1. Nelinearita umožňuje modelu opisovať komplexnejšie vzťahy medzi vstupnými dátami. Základné aktivačné funkcie sú ReLU, Sigmoid, Tanh a Softmax.
- **metriky (metrics)** - sú spôsoby posudzovania výkonnosti modelu strojového učenia. Viacero výpočtov metrík vychádza z tzv. konfúznej matice.




Obrázok 8 Konfúzna matica

- **presnosť (accuracy)** - je najpoužívanejšia metrika pre klasifikačné algoritmy. Vypočíta sa ako pomer správnych predikcí k celkovému počtu predikcií:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1.1}$$

- **precíznosť (precision)** - je metrika merajúca podiel správnych pozitívnych predikcií k počtu správne aj nesprávne pozitívnych predikcií. Meria úroveň správnosti modelu.

$$Precision = \frac{TP}{TP + FP} \tag{1.2}$$

- **dataset (dataset)** - je súbor dát, na ktorých sa model učí a overuje správnosť predikcií. Rozdeľujeme ho na:
  - tréningovú množinu (training set) - slúži na trénovanie modelu
  - validačnú množinu (validation set) - slúži na dolaďovanie hyperparametrov modelu
  - testovaciu množinu (test set) - slúži na vyhodnotenie modelu.

Teraz, keď sme sa zoznámili so základnými pojmami používanými v oblasti strojového učenia, zameriame sa na nástroje - knižnice a rámce, ktoré slúžia na vytváranie a trénovanie modelov. Vyberieme si iba niekoľko najpoužívanejších, ktoré majú vzťah k tvorbe modelov strojového učenia použiteľných nielen všeobecne, ale aj pre mobilné aplikácie.

# 2. Knižnice a rámce v strojovom učení

Použitie dostupných knižníc a rámcov je v programovaní samozrejmé. Zrýchľuje vývoj aplikácie, určuje jej architektúru a umožňuje nám používať funkcie, ktoré by sme inak museli vyvíjať veľmi dlho. Rovnako je to aj pri strojovom učení. Neurónové siete môžeme vytvárať úplne od základov len s použitím niektorého z programovacích jazykov. Efektívnejšie je však využiť existujúce knižnice a rámce, z ktorých mnohé ponúkajú jednoduché high-level API, pričom stále umožňujú detailné nastavenia parametrov a manipuláciu s funkciami na najnižšej úrovni.

V nasledujúcej časti v krátkosti predstavíme niekoľko knižníc a rámcov určených pre strojové učenie. Pri každej je vhodné nazrieť aj do dokumentácie, ktorá sa nachádza na webovej stránke danej knižnice.

## 2.1 Scikit-learn

Scikit-learn alebo skrátene sklearn je knižnica algoritmov určených pre strojové učenie, ktorú pôvodne vyvinul David Cournapeu ako projekt v rámci Google summer of code v roku 2007. Je postavená na open source pythonovskej knižnici SciPy používanej na vedecké a technické výpočty. Rovnako ako aj iné ML knižnice a rámce, aj ona využíva na výpočty a zobrazovanie knižnice NumPy (počítanie s viacrozmernými poliami a maticami), Pandas (dátová manipulácia a analýza), a Matplotlib (vizualizácia dát).

So Sciki-learn môžeme pracovať v rôznych IDE určených pre Python, ako je PyCharm, Spyder alebo Jupyter Notebook. Inou možnosťou je použiť veľmi populárnu distribúciu Anaconda, ktorá v sebe zahŕňa viacero nástrojov strojového učenia pre jazyky Python a R, vrátane JupyterLab, RStudio Visual Studio Code, či už spomínaného Jupyter Notebook-u.

Medzi najpoužívanejšie skupiny algoritmov v Scikit-learn patria:

- učenie s učiteľom - lineárna regresia, Support Vector Machine (SVM), rozhodovacie stromy, K-nearest neighbours (KNN)
- učenie bez učiteľa - K-means clustering, principal component analysis (PCA), Restricted Boltzmann machines (stochastická NN) [4]

## 2.2 PyTorch

PyTorch je ML rámec dostupný ako open source pod licenciou BSD a je využívaný predovšetkým na výskum a vývoj DL aplikácií. Vznikol vo Facebook AI Research Lab v roku 2016 a zakladá sa na ML knižnici Torch, ktorá sa integráciou do PyTorch od roku 2018 aktívne nevyvíja. Podobne ako NumPy, pracuje s n-rozmernými poliami (tenzormi), pričom poskytuje silnú podporu výpočtov na GPU.

Funcionalita PyTorch je porovnateľná s rámcom TensorFlow. PyTorch ponúka jednoduché API, používa Python a je optimalizovaný na beh v C++ prostrediach, tvorí celý ekosystém navzájom prepojených nástrojov a knižníc uľahčujúcich vývoj a nasadenie DL modelov. Hoci poskytuje end-to-end workflow PyTorch Mobile, ktorý je určený pre mobilné aplikácie, zatiaľ nemá kompletnú funkcionalitu a široké možnosti nasadenia ako TensorFlow. V súčasnosti je PyTorch Mobile v beta verzii. Prvá stabilná verzia by mala byť dostupná v priebehu roka 2022.

Pre prácu s PyTorch je možné využiť lokálne vývojové prostredia ako PyCharm alebo Spyder rovnako ako niektorú z cloudových služieb, napr. Amazon Web Services, IBM Watson Studio alebo Microsoft Azure. [5]

## 2.3 Keras

Keras nie je klasickou ML knižniciou, ale skôr jednoduchým a užívateľsky prívetivým rozhraním, ktoré poskytovalo high-level API pre viaceré knižnice strojového učenia - TensorFlow, Theano, PlaidML, MXNet alebo Microsoft Cognitive Toolkit. Vzniklo v roku 2015 v rámci Open-ended Neuro-Electronic Intelligent Robot Operating System a jeho autorom je AI výskumník spoločnosti Google Francois Chollet. V súčasnosti je Keras centrálnou súčasťou ekosystému TensorFlow 2. Môže tak využívať všetky možnosti, ktoré platforma TensorFlow ponúka pre nasadenie modelov bežiacich priamo v prehliadači, prostredníctvom TensorFlow Lite na operačné systémy Android a iOS, alebo single-board počítače Raspberry Pi či rôzne vnorené systémy.

V spojení s TensorFlow 2 tvorí Keras zrejme najobľúbenejší rámec používaný pri vývoji a výskume hlbokého učenia. [6]

## 2.4 TensorFlow / TensorFlow Lite

TensorFlow je open-source rámec vyvinutý tímom Google Brain, ktorý umožňuje vytváranie modelov a ich nasadenie na širokú škálu zariadení od serverov, cez mobilné zariadenia, až po vnorené systémy. Podporuje viacero programovacích jazykov, ako sú Python, Swift, C++, Java alebo JavaScript.

Na vytváranie modelov určených pre mobilné zariadenia a vnorené systémy je špeciálne určený rámec TensorFlow Lite, ktorý priamo vychádza z TensorFlow. Vytvorené modely v TensorFlow je možné jednoducho konvertovať pomocou TensorFlow Lite a použiť ich v zariadeniach s nižším výpočtovým výkonom alebo s menším objemom pamäte. Modely TensorFlow Lite môžu využívať centrálne procesorové jednotky (CPU), grafické procesorové jednotky (GPU), digitálne signálne procesory (DSP), ale aj neurónové procesorové jednotky (NPU). Použitie konkrétneho druhu procesora zavisí od ML modelu, zariadenia, na ktorom bude bežať a požadovanej rýchlosti. Pre jednoduché modely je zvyčajne najlepšou voľbou CPU. Ak by sme si vzali mobilný telefón s rôznymi druhmi procesorov, použitím GPU namiesto CPU sa pri rovnakom ML modeli môže zvýšiť výkon 5-násobne, pri použití DSP 10-násobne a pri NPU viac ako 40-násobne v porovnaní s CPU.

Vytvorenie základného modelu pomocou TensorFlow vyžaduje iba niekoľko riadkov kódu.

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dropout(0.3),
        tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(
   optimizer='adam',
   loss='sparse_categorical_crossentropy',
   metrics=['accuracy'])

model.fit(x_train, y_train, epochs=10)
model.evaluate(x_test, y_test)
```

V high-level API, ktorým je v TensorFlow 2 Keras, nie je potrebné definovať váhy ani bias jednotlivých vrstiev neurónovej siete. Vyššie uvádzame implementáciu NN s jednou skrytou vrstvou obsahujúcou 128 neurónov, ktorá je trénovaná na datasete MNIST. Tá obsahuje 60000 čiernobielych obrázkov s rozmermi 28x28px a slúži na tréning rozoznávania ručne písaných číslic.

TensorFlow ponúka stovky predtrénovaných modelov a testovacích datasetov, s ktorými je možné jednoducho pracovať v IDE Jupyter Notebook alebo Google Colab, čo je cloudová služba spoločnosti Google Research založená na platforme Jupyter Notebook. Colab podporuje programovací jazyk Python, spúšťa sa priamo vo webovom prehliadači, je zdarma, ale vyžaduje vytvorenie alebo previazanie s existujúcim gmail účtom. [7]

## 2.5 Create ML / Core ML

Create ML spoločnosti Apple je rámec určený na trénovanie a vytváranie modelov v priestore ekosystému produktov Apple, ktorý spoločnosť uviedla v roku 2018. Ponúka užívateľsky intuitívne prostredie umožňujúce vytváranie ML modelov bez potreby programovania. Natrénované modely je možné jednoducho integrovať do vlastných aplikácií pomocou nástroja Core ML s použitím iba niekoľkých riadkov kódu.

V súčasnosti je Create ML súčasťou vývojového prostredia s názvom Xcode, ktoré slúži na vývoj softvéru pre macOS, iOS, iPadOS, watchOS, and tvOS. Ako sme napísali vyššie, pre samotné vytvorenie ML modelu nie sú potrebné znalosti žiadneho programovacieho jazyka, ale na samotnú integráciu modelu v Xcode je potrebná znalosť jazyka Swift. Ten postupne nahradil Objective-C, ktorý bol spočiatku primárnym programovacím jazykom pre iOS a macOS.

Predtrénované modely Create ML sa rozdeľujú do niekoľkých kategórií podľa domény aplikácie:

- obraz - *image classification* (rozpoznávanie obrázkov podľa ich obsahu), *object detection* (detekcia jedného alebo viacerých objektov v obrázku), *hand pose classification* (klasifikácia póz rúk pomocou tréningu s obrázkami rúk ľudí)
- video - *action classification* (rozpoznávanie pohybov ľudského tela), *hand action classification* (klasifikácia pohybov rúk alebo giest)

- pohyb - *activity classification* (klasifikácia dát zo snímača pohybu)
- zvuk - *sound classification* (rozpoznávanie zvukov zo zvukových súborov alebo audio streamu)
- text - *text classification* (rozpoznávanie schém v prirodzenom jazyku, napr. pocit vyjadrený v texte), *word tagging* (klasifikácia textu na úrovni slov)
- tabuľkové - *tabular classification* (klasifikácia dát do diskrétnych kategórií), *tabular regression* (odhad spojitých hodnôt ako cena, čas, teplota), *recommendation* (odporúčania pre užívateľa na základe podobností, zoskupovania alebo hodnotenia)

Keďže Create ML používa predtrénované modely, samotné trénovanie modelu na vlastných dátach nevyžaduje veľký výpočtový výkon ani čas. Napríklad vytvorenie modelu na detekciu určitého objektu vo fotografii môže trvať na priemernom MacBook-u len niekoľko minút a stačí k tomu pár desiatok fotografií daného objektu.

Pomocou pythonovského balíka coremltools je možné konvertovať natrénované ML modely z knižníc a rámcov TensorFlow, PyTorch, XGBoost, LibSVM a scikit-learn. Po konverzii môžu byť integrované do aplikácie pomocou Core ML rovnako, ako modely vytvorené v Create ML.

Core ML zabezpečuje pre všetky modely jednotnú reprezentáciu, optimalizuje výkon na zariadení využívaním CPU, GPU a ANE (Apple's Neural Engine), čím minimalizuje nároky na pamäť a spotrebu energie. Umožňuje aj aktualizáciu modelov v aplikáciách pomocou používateľských údajov na zariadení. Tá sa uskutočňuje priamo na zariadení, takže modely zostávajú relevantné bez ohrozenia súkromia používateľov. [8]

# 3. Inteligentné aplikácie

Využitie strojového učenia v mobilných zariadeniach je veľmi rôznorodé, od aplikácií určených výhradne na zábavu až po monitorovanie zdravotného stavu. "*Mobilné zariadenia, ktoré sa jednoducho používali na telefonovanie a odosielanie textových správ, sa teraz zavedením AI premenili na smartfóny. Tieto zariadenia sú teraz schopné využiť stále sa zvyšujúci výkon AI na učenie správania sa a preferencií používateľov, vylepšovanie fotografií, uskutočňovanie plnohodnotných konverzácií a oveľa viac.*" [9] S rastom hardvérových možností a zrýchľovaním internetového pripojenia rastú aj požiadavky a očakávania užívateľov. Funkcionalita, ktorú by sme si pred desiatimi rokmi vedeli ťažko predstaviť, je v dnešnej dobe bežnou súčasťou aplikácií. Tento trend je nezvratný a núti vývojárov implementovať do aplikácií prvky strojového učenia tak, aby ich produkty boli aj naďalej schopné konkurencie. Niektoré z príkladov a výhod použitia strojového učenia v aplikáciách (nielen mobilných) si v krátkosti priblížime nižšie:

- zábava - použitie strojového učenia v hrách slúži na zvýšenie hráčskeho zážitku, zvýšenie faktoru "zábavnosti" aplikácie - napr. použitie filtrov na fotografie alebo hlas
- vzdelávanie - rozpoznávanie objektov a ich určovanie, pomoc pri výučbe cudzích jazykov, analýza prirodzeného jazyka a oprava gramatiky
- zvýšená bezpečnosť - prostredníctvom bezpečnostných prvkov ako rozpoznávanie tváre a iných biometrických údajov, detekciou potenciálne podvodných finančných transakcií a aktivít
- personalizácia zákazníckej skúsenosti - na základe analýzy správania sa zákazníka model vytvára relevantné odporúčania na produkty (tovar, služby, multimédiá), ktoré by zákazníka mohli zaujímať
- zdravie a fitnes - monitorovanie životných funkcií a predikcia kritických stavov, vyhodnocovanie aktivity a optimalizácia cvičenia
- manažment dát - spracovanie reči do textovej podoby, optické rozpoznávanie znakov, extrakcia, spracovanie a ukladanie údajov z fotografií

Veľa mobilných aplikácií je vytvorených ako súčasť väčšieho ekosystému, ktorý môže zahŕňať webovú alebo aj desktopovú platformu. Napríklad obľúbená hudobná streamovacia služba Spotify, ktorá vo veľkej miere využíva strojové učenie, je multiplatformová. Je dostupná pre desktopové operačné systémy Windows a macOS, mobilné zariadenia so systémom Android alebo iOS, ale aj ako webový prehrávač. Aj z tohto dôvodu nevyužívajú mnohé mobilné aplikácie strojové učenie priamo na smartfóne, ale odosielajú dáta na spracovanie do datacentier. Výhodou takéhoto riešenia je agregácia dát a ich lepšie spracovanie, nevýhodou nevyhnutnosť internetového pripojenia a možné bezpečnostné riziká. Toto riešenie však otvára možnosti aj pre menších vývojárov aplikácií, ktorí nemajú prostriedky na prevádzku vlastných serverov určených na strojové učenie. Napríklad v roku 2016 sprístupnila firma Google prostredníctvom Cloud Machine Learning Platform (neskôr Cloud AI Platform) svoje modely cez rozhranie API verejnosti. V súčasnosti ponúka modely pre počítačové videnie (detekcia tváre, anotácie fotografií, rozpoznávanie znakov), spracovanie reči, spracovanie videa (editácia, anotácia), odporúčacie systémy a spracovanie prirodzeného jazyka (syntax, sentiment). Samozrejme, Google nie je jedinou firmou, ktorá ponúka spracovanie úloh strojového učenia prostredníctvom API. Medzi ďalších veľkých hráčov patrí v tejto oblasti napríklad Amazon (Amazon Machine Learning), IBM (Watson Discovery) alebo Microsoft (Azure Cognitive Service).

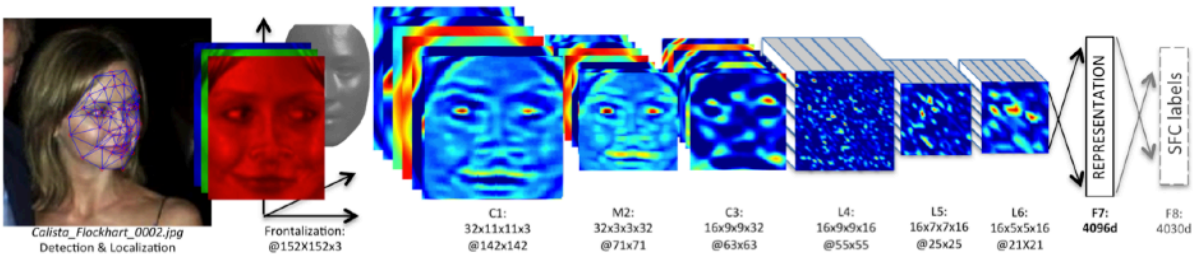
V nasledujúcom texte si bližšie popíšeme tri spôsoby využitia strojového učenia (nielen) v mobilných aplikáciách.

## 3.1 Rozpoznávanie tváre

Technológia rozpoznávania tváre je známa už niekoľko desaťročí, ale až v ostatných rokoch našla vďaka hlbokému učeniu široké uplatnenie. Používa sa nielen na odomykanie zariadení a autentifikáciu, ale aplikácie ako Facebook, Snapchat, či Instagram ju používajú na označovanie užívateľov na fotografiách alebo vytváranie zábavných fotografických filtrov.

Existujú dva spôsoby rozpoznávania tváre. Prvý prístup je založený na extrakcii príznakov - uzlových (nodal) alebo orientačných (landmark) bodov. Tie môžu zahŕňať

vzdialenosť medzi očami, šírku nosa, vzdialenosť brady od čela. Každá aplikácia používa rôzne body, ktoré môžu spoločne tvoriť desiatky jedinečných príznakov. Tento prístup je viac založený na klasickom strojovom učení a využíva ho vo svojich filtroch napríklad aplikácia Snapchat. Druhý prístup využíva hlboké učenie a konvolučné neurónové siete. Príkladom tohto prístupu môže byť Facebook, ktorý používa na rozpoznávanie tváre a označovanie fotografií vlastnú neurónovú sieť DeepFace. "*Táto hlboká sieť zahŕňa viac ako 120 miliónov parametrov používajúcich niekoľko lokálne prepojených vrstiev bez zdieľania váh, a nie štandardných konvolučných vrstiev. Preto sme ju trénovali na doteraz najväčšom datasete tvárí, súbore údajov 4 miliónov označených tvárí, ktoré patria k viac ako 4 000 identitám.*" [10]




Obrázok 9 Štruktúra neurónovej siete DeepFace (prevzaté z [10])

Rozpoznávanie prebieha v štyroch krokoch - detekcia, zarovnanie, reprezentácia, klasifikácia. Pri detekcii sieť zisťuje, či sa na obrázku nachádza tvár. Cieľom druhého kroku je vygenerovať zo vstupného obrázku dvojrozmerný čelný pohľad na tvár, pretože na obrázku môže byť zachytená v rôznych pozíciach a uhloch. Architektúru reprezentácie a klasifikácie tvorí konvolučná sieť, ktorej vstupom je zarovnaný RGB obrázok veľkosti 152x152 pixelov.

Na trénovanie vlastnej konvolučnej siete je možné použiť voľne dostupné datasety fotografií ľudských tvárí ako Flickr-Faces-HQ Dataset. K dispozícii sú aj predtrénované modely neurónových sietí ako VGG-Face, Google FaceNet, OpenFace, vyššie spomenutý Facebook DeepFace alebo DeepID, prípadne API riešenia Microsoft Computer Vision, Lambda Labs, Face++ a iné.

## 3.2 Odporúčacie systémy

Personalizované služby a obsah sú tak bežnou súčasťou digitálneho sveta, že si ich prítomnosť častokrát ani neuvedomujeme. Odporúčacie systémy pomáhajú užívateľovi lepšie sa orientovať v množstve informácií. Sú navrhnuté tak, aby na základe rôznych faktorov predpovedali najpravdepodobnejšie produkty alebo informácie, ktoré by užívateľa mohli zaujímať. Bežne ich používajú eshopy, sociálne siete ako Facebook alebo Instagram, streamovacie aplikácie Spotify, Netflix, či YouTube a mnoho iných aplikácií, ktoré sa snažia svoje služby personalizovať.

Základom sú dve metódy ako odporúčacie systémy pracujú. Prvá je založená na filtrovaní na základe obsahu (content-based), kde sa najprv pomocou kľúčových slov vytvorí popis položiek. Následne sú podľa histórie interakcií (prezerané položky, nákupy, hodnotenia) ponúkané užívateľovi položky podobné tým, ktoré sa mu v minulosti páčili. Tento spôsob je vhodnejší pre situácie, kedy poznáme údaje o položkách, ale nie o užívateľovi. Filtrovanie na základe obsahu používa zo strojového učenia klastrovú analýzu, rozhodovacie stromy, bayesovské klasifikátory alebo neurónové siete. Druhou metódou je kolaboratívne filtrovanie (collaborative filtering). Táto metóda je založená na predpoklade, že užívatelia, ktorým sa v minulosti páčil rovnaký produkt, budú mať aj v budúcnosti podobné preferencie. Najprv je teda potrebné nájsť zhodu medzi užívateľmi a následne im odporúčací systém môže ponúknuť položky, ktoré by sa im mohli páčiť. Zo strojového učenia sa pri kolaboratívnom filtrovaní používa klasifikačný algoritmus KNN (K-najbližší susedia). [11]

Okrem vytvorenia vlastného odporúčacieho systému sú tieto systémy dostupné aj ako SaaS (software as a service). Skúšobnú verziu zdarma ponúka napríklad Google Cloud Recommendations AI.

## 3.3 Rozpoznávanie reči

Automatické rozpoznávanie reči (ASR), skrátene rozpoznávanie reči, transkribuje hovorenú reč do písanej podoby. Treba zdôrazniť, že sa odlišuje od rozpoznávania hlasu, ktoré sa zameriava iba na rozpoznávanie hovoriaceho a nie na to, čo hovorí. Tieto pojmy sa

častokrát vzájomne zamieňajú, pretože rozpoznávanie hlasu môže byť prvým krokom pri rozpoznávaní reči, ale rovnako dobre môže figurovať aj ako samostatná úloha.

Zrejme najznámejším príkladom využitia rozpoznávania reči sú osobní asistenti ako Siri, Cortana alebo Alexa, ktorých metódy transkripcie patria skôr do oblasti umelej inteligencie zaoberajúcej sa spracovaním prirodzeného jazyka (NLP). Zo strojového učenia sa využívajú predovšetkým hlboké neurónové siete a Skryté Markovove modely používané na rozpoznávanie reči v speech-to-text aplikáciách ako Dragon, Speechnotes, Windows speech recognition alebo Transcribe.

Rozpoznávanie reči sa zvyčajne skladá z nasledujúcich krokov [12]:

- pre-processing - zahŕňa kroky na zlepšenie audiosignálu redukovaním šumu, filtrovaním nepotrebných frekvencií alebo úpravu dĺžky ukážok
- extrakcia príznakov - vychádza z Fourierovej transformácie, pri ktorej sa prevádza audiosignál z časovej oblasti do frekvenčnej. Na extrakciu príznakov sa používa metóda MFCC (Mel Frequency Cepstral Coefficients). Je to malá množina príznakov (pre reč postačuje aj 13) opisujúca celé frekvenčné spektrum ako obrázok.
- klasifikácia - prebieha pomocou hlbokých neurónových sietí, zvyčajne konvolučných alebo rekurentných
- jazykové modelovanie - zachytáva syntax a sémantiku jazyka, vykonáva opravy na výstupnom texte.

Bližšie sa so spôsobom spracovania a klasifikácie zvuku budeme zaoberať v nasledujúcej kapitole. Rozdiel medzi rozpoznávaním reči a klasifikáciou zvuku je iba v poslednom kroku - jazykové modelovanie, ktoré sa pri klasifikácii zvuku nerobí.

# 4. Návrh a implementácia aplikácie v prostredí iOS - Birdyy

Teoretické poznatky získavajú svoju skutočnú hodnotu, až keď ich pretavíme do praxe. V nasledujúcej kapitole si preto popíšeme vytvorenie a implementáciu modelu strojového učenia na konkrétnom príklade. Zvyčajne sa ako ilustratívny príklad používa jednoduchý model logistickej regresie alebo kategorizácia obrázkov spadajúca do oblasti počítačového videnia. Aj z tohto dôvodu sme sa rozhodli ísť inou cestou, vytvoriť model rozpoznávajúci nie obrázky ale zvuky. Konkrétne spev vtákov. Úlohou modelu, resp. aplikácie, ktorú sme nazvali Birdyy, je rozpoznávať rôzne druhy vtákov podľa ich spevu. Keďže naším cieľom nebolo vytvoriť plne funkčnú aplikáciu, ale prakticky popísať vytvorenie a implementáciu modelu strojového učenia do mobilnej aplikácie, vybrali sme pre naše účely päť druhov vtákov, ktoré sa bežne vyskytujú v mestách na Slovensku. Takto je možné aplikáciu jednoducho otestovať aj v reálnych podmienkach na skutočnom mobilnom zariadení.

Postup našej práce kopíruje štandardnú postupnosť krokov vytvárania modelu strojového učenia.

1. definovanie problému
2. zber dát
3. predpríprava a čistenie dát
4. modelovanie
5. validácia

Vo väčšine prípadov nejde o proces lineárny ale iteratívny. Aj pri modelovaní alebo validácii môžeme zistiť, že máme nedostatočné alebo nevhodné dáta a sme nútení vrátiť sa k predošlým krokom.

Model sme implementovali a testovali v prostredí iOS na zariadení iPhone 12 Mini. Výber konkrétnej platformy a hardvérového zariadenia má zanedbateľný vplyv na fungovanie nášho modelu, pretože softvérovo je spôsob implementácie na platformách Android a iOS veľmi podobný a hardvérovo využíva model iba mikrofón mobilného zariadenia. Dôvodom výberu bola fyzická dostupnosť daného zariadenia a predošlá skúsenosť s vývojom aplikácií na platforme iOS.

## 4.1 Definovanie problému

Počujeme spev vtáka, ktorý je skrytý v korune stromu a nás by zaujímalo, aký druh spevavca to je. Táto situácia sa nám môže stať nielen v lese ale aj v meste. Rozoznávanie spevu vtákov nepatrí medzi bežné zručnosti, ktoré by sa učili v škole, preto nevieme s istotou určiť ani niektoré bežné druhy, ktoré vizuálne poznáme veľmi dobre.

Okrem toho, každý vtáčí druh má viacero typov spevu, ktoré sa mierne líšia od jedinca k jedincovi, preto je vhodným kandidátom na riešenie problému rozoznávania (klasifikácie) vtákov podľa spevu práve aplikácia využívajúca model hlbokého učenia.

## 4.2 Zber dát

Pre účely tejto práce sme sme obmedzili počet druhov vtákov, ktoré by mal model Birdyy rozoznávať na päť - sýkorká veľká (parus major), vrabec domový (passer domesticus), hrdlička záhradná (streptopelia decaocto), havran čierny (corvus frugilegus) a drozd čierny (turdus merula). Zámerne sme vybrali tri druhy, ktorých spev sa podobá - sýkorka, vrabec a drozd, a dva druhy, ktorých spev je výrazne odlišný - hrdlička a havran.

Keďže ide o druhy, ktoré sa bežne vyskytujú v slovenských mestách, mohli sme dáta potrebné na učenie modelu získať z vlastných audionahrávok. Tento prístup bol však časovo náročný, preto sme hľadali iný zdroj dát. Jednou z možností bolo využiť dostupné datasety z webovej stránky Kaggle (www.kaggle.com), ktorá sa zaoberá strojovým učením a je dobrý zdrojom rôznych voľne dostupných datasetov. Rozhodli sme sa však pre databázu xeno-canto (www.xeno-canto.org) určenú na zdieľanie spevu vtákov z celého sveta. Dôvodom bol väčší počet audionahrávok a možnosť ich filtrovania podľa druhu, krajiny a kvality nahrávky. Každú nahrávku sme si museli vypočuť a posúdiť, pretože v niektorých prípadoch kvalita audiozáznamu nepostačovala pre potreby učenia modelu.

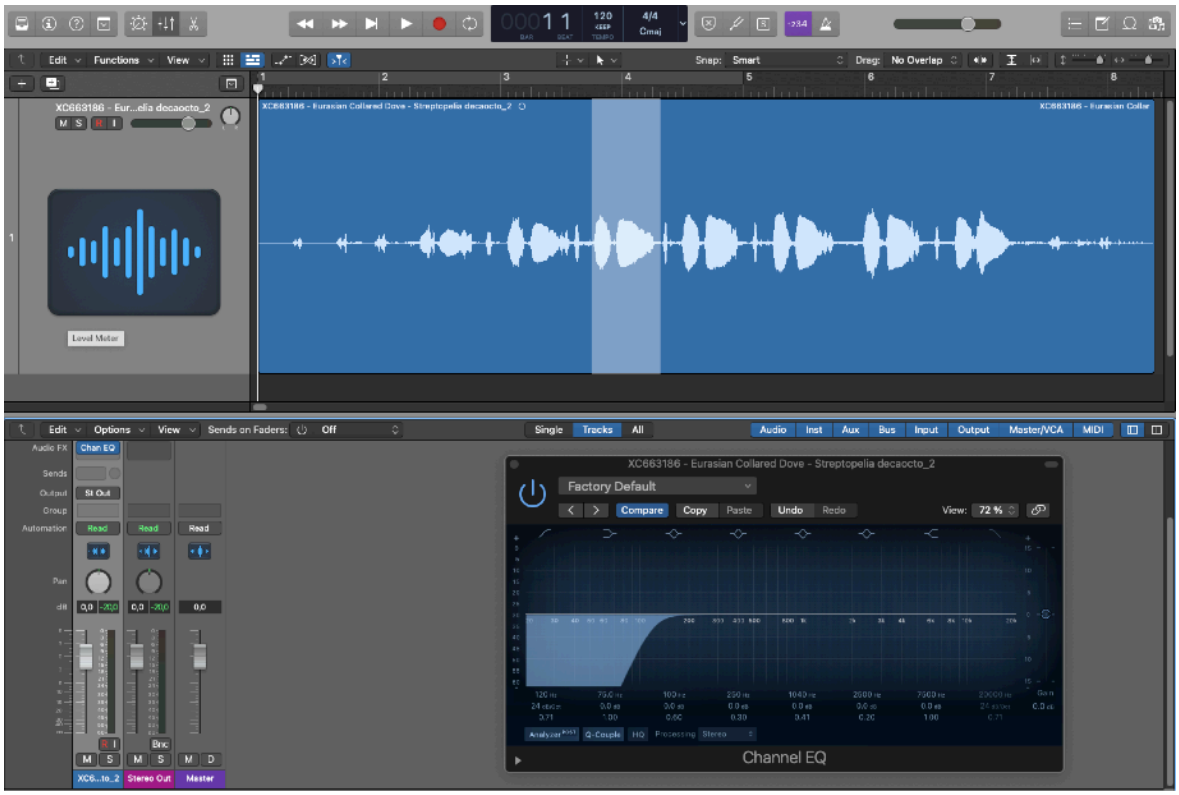
Náš dataset sme neskôr rozšírili ešte o dve kategórie zvukov. Prvou z nich bol spev iných vtákov, ktoré model Birdyy nevedel identifikovať. Druhou kategóriou bol bežný hluk mesta, v ktorom nie je počuť spev vtákov. Pre tieto kategórie sme použili vlastné audionahrávky, ktoré sme doplnili voľne dostupnými nahrávkami z webovej stránky xeno-canto a SoundBible (www.soundbible.com).

## 4.3 Predpríprava dát

Základom dobre fungujúceho modelu je dostatočné množstvo kvalitných dát. Ak ide o tabuľkové dáta, je potrebné ich správne naformátovať, normalizovať alebo štandardizovať hodnoty s veľkým rozptylom, skontrolovať chýbajúce hodnoty a rozdeliť dáta na trénovaciu a testovaciu množinu. Podobné je to v prípade multimediálnych súborov.

V našom prípade boli nahrávky spevu vtákov zo stránky xeno-canto v dvoch rôznych audio formátoch (wav a mp3) a obsahovali príliš veľa "šumu" - predovšetkým zvukov pozadia a spevu iných vtákov. Čistý čas spevu požadovaného druhu vtáka tvoril často iba 50 percent času nahrávky, čo viedlo k zníženiu predikčnej schopnosti modelu.

Na konverziu formátov, frekvenčné filtrovanie a odstránenie tichých častí z nahrávky je možné použiť napríklad pythonovské knižnice Librosa a SciPy. Nedokážu však riešiť problém spevu iných vtákov na nahrávke. Z tohto dôvodu sme sa rozhodli urobiť celú predprípravu dát v programe Logic Pro určenom na úpravu audia, ktorý umožňuje strihanie, frekvenčné filtrovanie a ukladanie záznamu v požadovanom formáte. K dispozícii sú však aj voľne dostupné softvéry ako Audacity alebo Reaper.




Obrázok 10 Úprava audionahrávok v programe Logic Pro

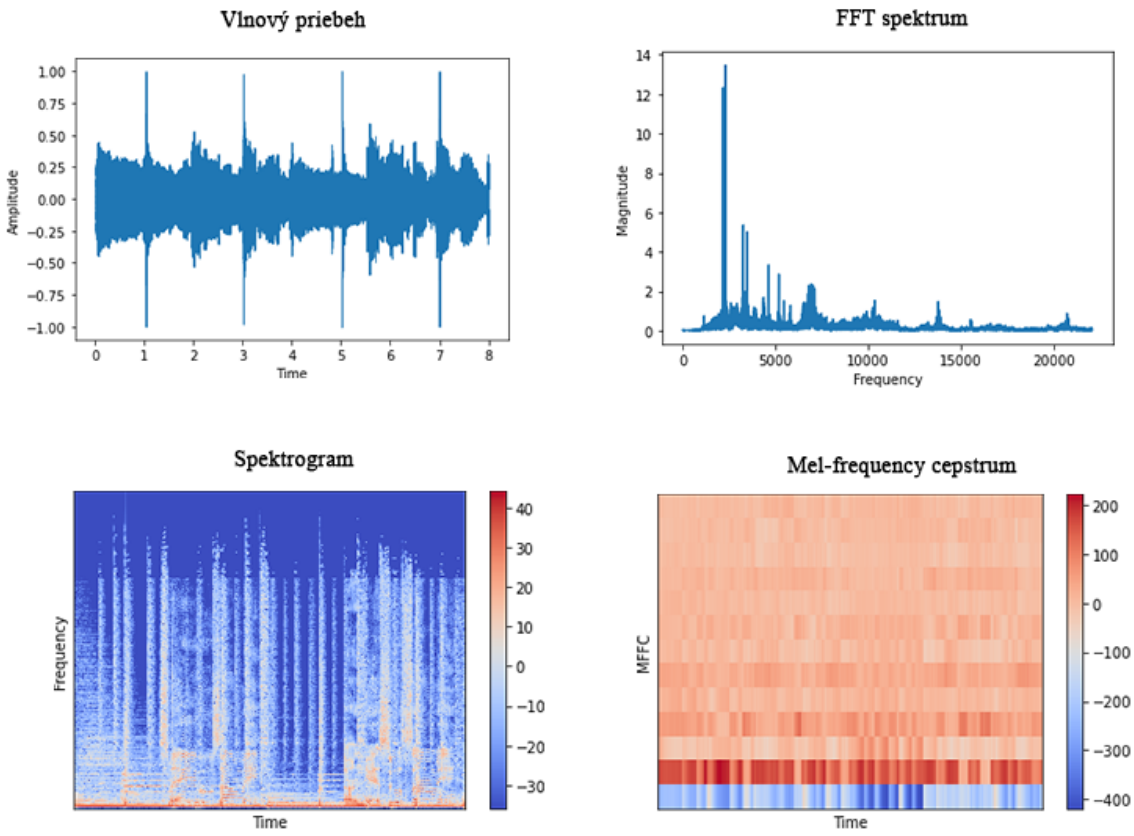
Zo stiahnutých nahrávok sme vytvorili zhruba päťsekundové klipy spevu zvolených vtákov, v ktorých sme pomocou funkcie odstránenia nízkych frekvencií (low cut) odfiltrovali frekvencie nižšie ako 120 Hz. Takto upravené klipy sme pomenovali (označili) a uložili vo formáte wav.

Vytvorili sme tak zhruba 1250 audioklipov, ktoré boli správne pomenované a rozdelené do samostatných priečinkov podľa jednotlivých druhov vtákov. K nim sme pridali priečinky obsahujúce bežný ruch mesta a spev iných vtákov. Celkovo tak vznikol dataset pozostávajúci zo siedmich kategórií, ktorý sme rozdelili na trénovaciu a testovaciu množinu v pomere 95% a 5 %.

## 4.4 Extrakcia príznakov

V prípade vytvárania modelu pre platformu iOS je najjednoduchším spôsobom využiť rámec Create ML, ktorý je súčasťou vývojového prostredia Xcode a extrakciu príznakov vykonáva automaticky pri učení modelu. My si však bližšie popíšeme extrakciu príznakov v aplikácii Jupyter Notebook.

Digitálny záznam zvuku je charakterizovaný vzorkovaciou frekvenciou (početom záznamov za sekundu) a bitovou hĺbkou (počtom možných hodnôt úrovní signálu). Modely pracujúce so zvukom zvyčajne využívajú architektúru konvolučných alebo rekurentných sietí učiacich sa z transformovanej podoby zvukového signálu. Tú je možné v grafickej podobe zobraziť viacerými spôsobmi (obrázok 11). Pre účely trénovania modelu Birdyy sme použili transformáciu pomocou Mel-frequency cepstral coefficients (MFCCs), ktorý sa zvyčajne používa na extrakciu príznakov pri rozpoznávaní reči a klasifikácii zvukov. Princíp algoritmu MFCC je založený na analýze kratších časových úsekov signálu (30 - 40 ms) a ich spracovaní podľa Melovej frekvenčnej škály, ktorá lepšie zodpovedá rozdeleniu frekvenčných pásiem na základe ľudského vnímania zvuku. Prostredníctvom Fourierovej transformácie sa vypočíta amplitúdové spektrum (FFT spektrum). Kepstrálne koeficienty sa následne vypočítajú pomocou diskrétnej kosínusovej transformácie z logaritmov súčtov amplitúd v jednotlivých pásmach. Počet pásiem môže byť od 12 až po 128, pričom 12-13 sa považuje za dostatočný počet na analýzu ľudskej reči. Z tohto dôvodu sme pri vytváraní MFCC zvolili počet koeficientov 13.




Obrázok 11 Grafické zobrazenie audiosignálu

Na vytvorenie MFCCs sme využili knižnicu Librosa, ktorú sme importovali do vytvoreného zošitu v Jupyter Notebook:

```
import librosa

mfcc = librosa.feature.mfcc(
        y = signal,             # vstupny signal
        sr = 44100,             # samplovacia frekvencia
        n_fft = 2048,           # velkost okna Fourierovej transformacie
        n_mfcc = 13,            # pocet kepstralnych koeficientov
        hop_length = 1024)      # velkost okna pri MFCC transformacii
```

Premenná *mfcc* obsahuje maticu hodnôt 13 kepstrálnych koeficientov v jednotlivých časových rámcoch, ktorých dĺžka je 1024 samplov, čo pri vzorkovacej frekvencii 44 100 zodpovedá približne úseku 43 ms. Údaje sme uložili vo formáte json tak, aby sme ich mohli následne použiť pri trénovaní neurónovej siete. Kompletný kód funkcie uvádzame v prílohe našej práce.

## 4.5 Vytvorenie modelu

Model sme vytvárali dvomi spôsobmi. Na trénovanie pomocou Tensorflow sme použili ako vstup súbor s extrahovanými príznakmi vytvorený v predošlej kapitole. Create ML extrahuje príznaky automaticky pred trénovaním modelu, preto sme ako vstup použili wav súbory z datasetu.

### 4.5.1 Vytvorenie modelu v TensorFlow

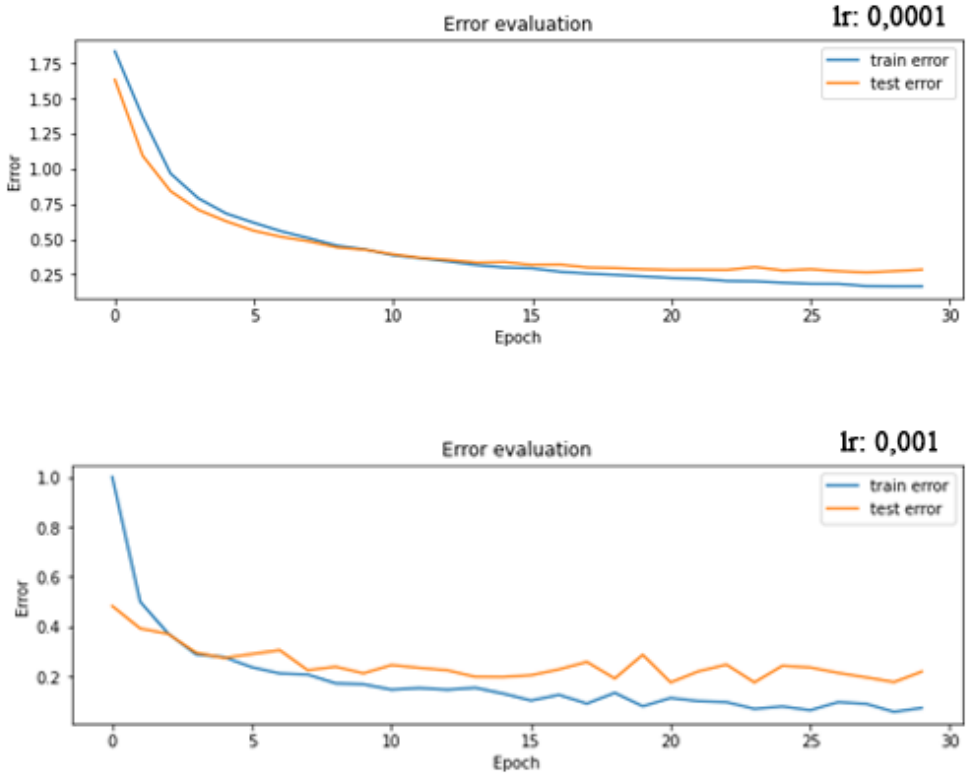
V Jupyter Notebook sme importované dáta konvertovali na numpy pole a rozdelili ich na trénovaciu, validačnú a testovaciu množinu. Pre model sme zvolili architektúru rekurentnej siete Long Short-Term Memory (LSTM) s dvomi LSTM vrstvami a jednou plne prepojenou vrstvou.

```
model = keras.Sequential()

# LSTM vrstvy
model.add(keras.layers.LSTM(64, input_shape=input_shape,
                            return_sequences=True))
model.add(keras.layers.LSTM(64))

# plne prepojená vrstva
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(0.3))

# výstupná vrstva
model.add(keras.layers.Dense(7, activation='softmax'))
```

| Layer (type) | Output Shape | Param # |
|---|---|---|
| lstm_19 (LSTM) | (None, 22, 64) | 19968 |
| lstm_20 (LSTM) | (None, 64) | 33024 |
| dense_20 (Dense) | (None, 64) | 4160 |
| dropout_9 (Dropout) | (None, 64) | 0 |
| dense_21 (Dense) | (None, 7) | 455 |

Obrázok 12 Zhrnutie architektúry LSTM siete

Po niekoľkých testoch s rôznymi konfiguráciami hyperparametrov sme na optimalizáciu použili algoritmus Adam s rýchlosťou učenia 0.0001, chybovú funkciu Sparse categorical crossentropy, veľkosť dávky 32 a počet epôch 30. Na ilustráciu ako

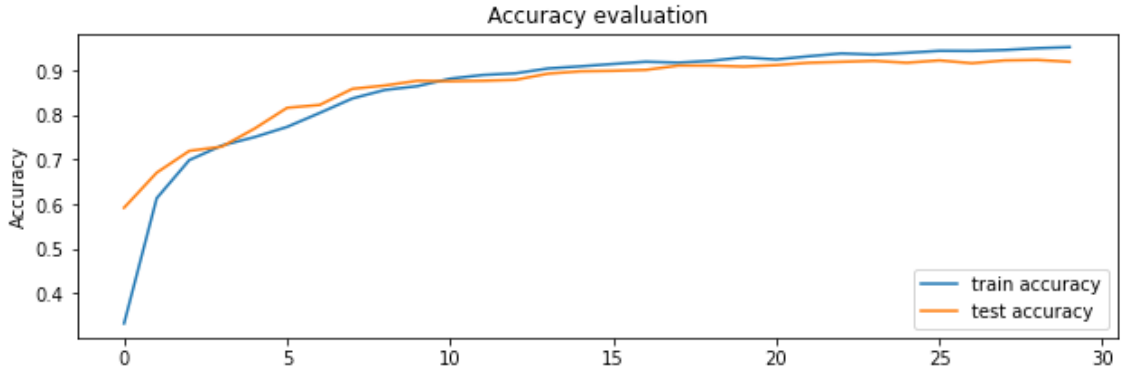



Obrázok 13 Vplyv zmeny rýchlosti učenia na chybovosť predikcií

zmena parametrov môže zmeniť chybovosť predikcií, uvádzame obrázok 13, kde sme v jednom prípade pri trénovaní použili rýchlosť učenia 0,0001 a v druhom 0,001.

Keďže sme na trénovanie použili relatívne malý dataset a zvolili sme architektúru siete iba so štyrmi skrytými vrstvami, trénovanie modelu trvalo iba niekoľko minút a presnosť na testovacej množine bola 91,7 %.

```
Accuracy on test set is 0.9171974658966064
```




Obrázok 14 Vyhodnotenie presnosti modelu

### 4.5.2 Vytvorenie modelu v Create ML

Create ML je súčasťou vývojového prostredia Xcode a ponúka vývojárom mobilných aplikácií vytvorenie modelu neurónovej siete bez potreby programovania. Ako sme spomínali vyššie, extrakcia príznakov prebieha automaticky a na výber sú dva algoritmy:

- Audio Feature Print (AFP) - extraktor optimalizovaný pre klasifikáciu zvukov, ktorý podporuje flexibilnú veľkosť trvania analyzovaného úseku (okna). Modely vytvorené pomocou AFP majú niekoľkonásobne menšiu veľkosť ako VGGish, ale sú zároveň menej presné.
- VGGish - extraktor, ktorý sa bežne používa na klasifikáciu zvukov. Má fixnú veľkosť trvania okna - 975 milisekúnd.

Pre porovnanie sme v Create ML vytvorili dva modely, ktoré sa líšia iba extrakčným algoritmom. Pri oboch modeloch sme použili rovnakú veľkosť okna - 975 ms, prekrytie okna 50% a počet iterácií 25.

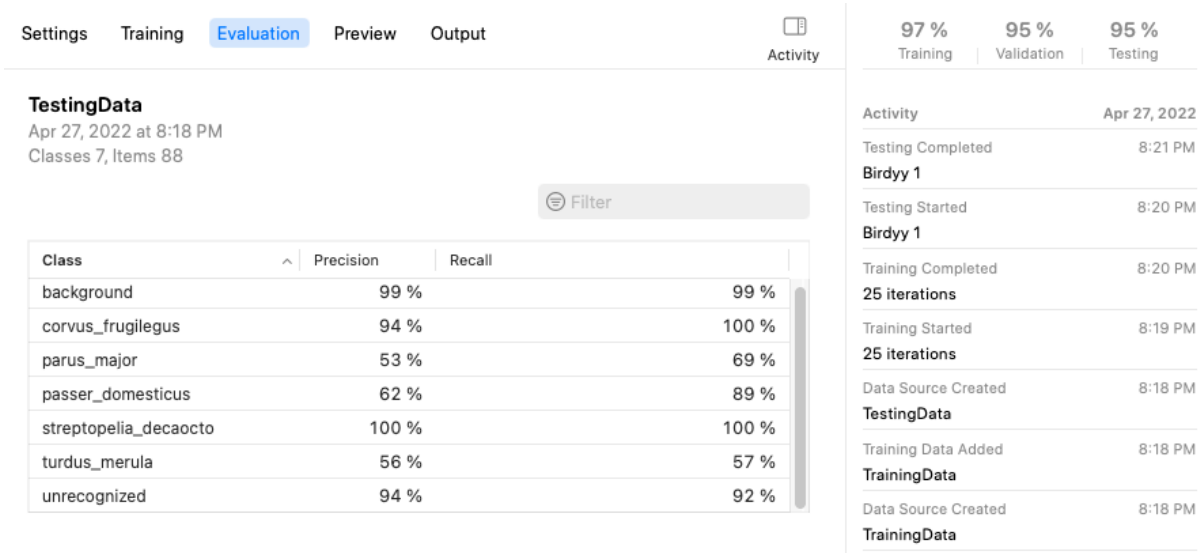
• **Birdyy 1**:

extrakcia: Audio Feature Print

presnosť na testovacej množine: 95%

čas učenia: 2 minúty

veľkosť modelu: 26 kB




Obrázok 15 Model Birdyy 1 - Audio Feature Print

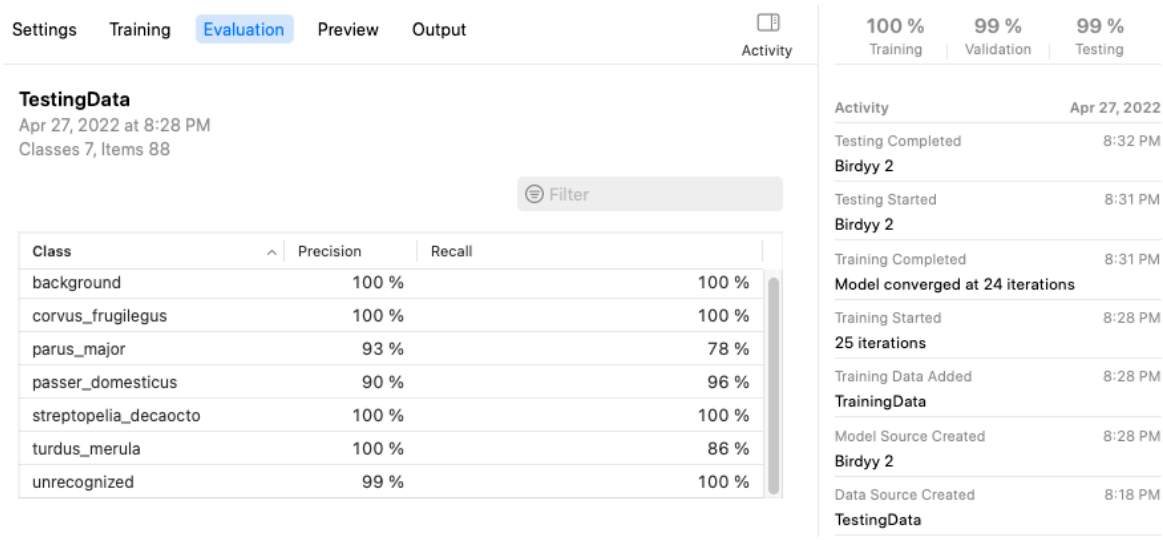
- **Birdyy 2**:

  extrakcia: VGGish

  presnosť na testovacej množine: 99%

  čas učenia: 4 minúty

  veľkosť modelu: 5,1 MB




Obrázok 16 Model Birdyy 2 - VGGish

Ako môžeme vidieť z obrázkov 15 a 16, model vytvorený s použitím extraktoru Audio Feature Print je viac ako 20-násobne menší, pričom presnosť na testovacej množine je menšia iba o 4 percentá.

## 4.6 Implementácia

Pre účely implementácie sme sa rozhodli použiť model vytvorený v Create ML s extraktorom VGGish, ktorý mal najvyššiu presnosť na testovacej množine dát. Model sa importoval do vývojového prostredia Xcode jednoduchým drag and drop spôsobom. Xcode automaticky vytvoril triedu Birdyy(), takže model bol bezprostredne použiteľný. Pri implementácii sme vychádzali z oficiálnej dokumentácie a viacerých prezentačných videí zaoberajúcich sa analýzou zvuku, ktoré boli zverejnené na stránkach firmy Apple určených pre developerov (https://developer.apple.com/videos/ml-vision).

Klasifikácia zvukov mikrofónu využíva API Sound Analysis Framework, a preto bolo na jej základnú implementáciu potrebných iba niekoľko objektov:

- **SNClassifySoundRequest** - objekt zodpovedný za vytvorenie požiadavky na klasifikáciu s použitím modelu Birdy

```
let request = try SNClassifySoundRequest(mlModel: birdy model)
```

- **SNAudioStreamAnalyzer** - vytvorenie analyzéru, ktorého vstupom je signál z mikrofónu mobilného zariadenia

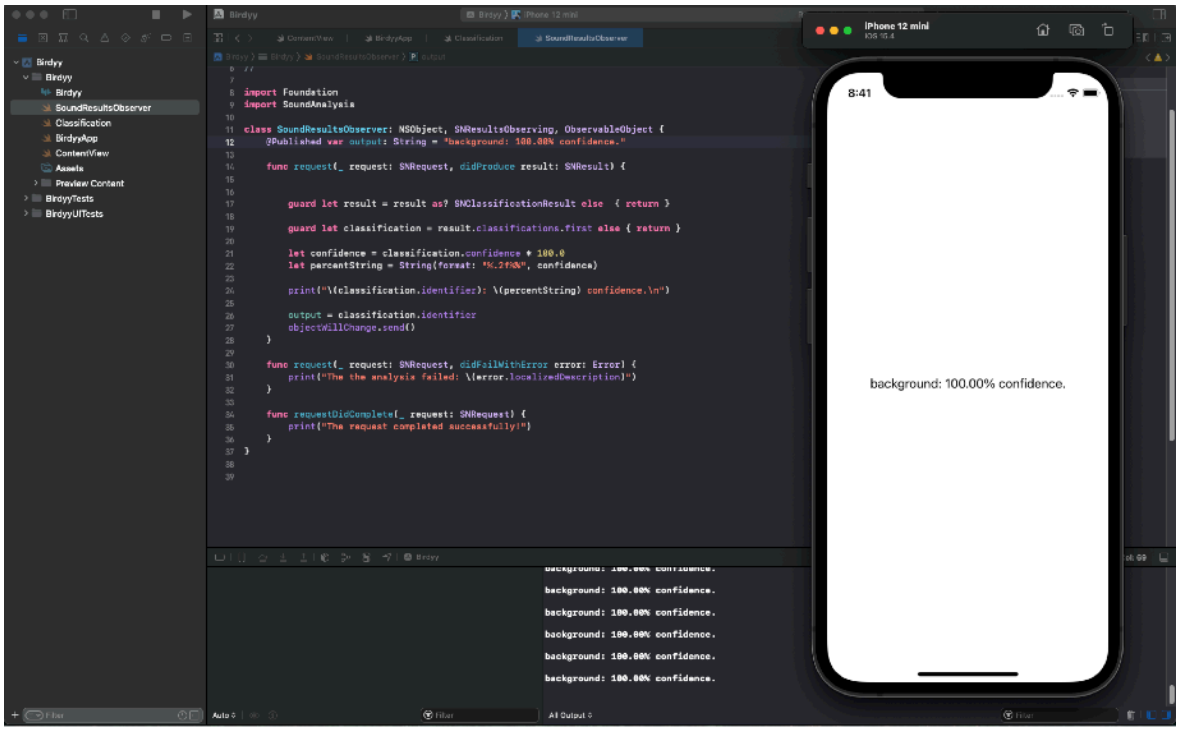
```
let streamAnalyzer = SNAudioStreamAnalyzer(format: inputFormat)
```

- **Observer** - trieda, ktorá implementuje SNResulsObserving protokol a je zodpovedná za spracovanie výsledkov analýzy/klasifikácie.

```
let resultsObserver: SNResultsObserving
```

Následne sa požiadavka pridá do analyzéra a vykoná sa samotná analýza signálu z audio vstupu.

```
streamAnalyzer.add(request, withObserver: resultsObserver)
streamAnalyzer.analyze(buffer, atAudioFramePosition: time.sampleTime)
```

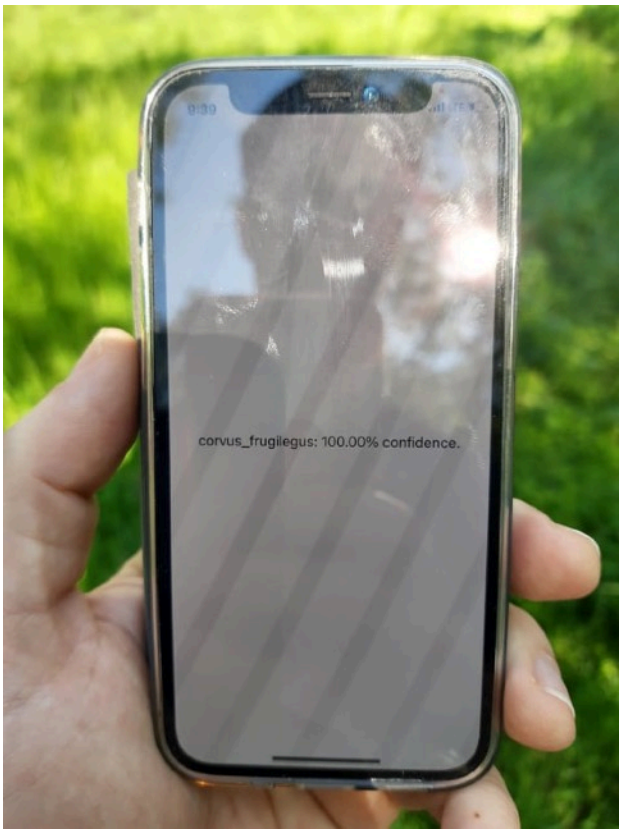



Obrázok 17 Implementácia modelu Birdyy vo vývojovom prostredí Xcode. Výsledky klasifikácie sa vypisujú v simulátore a v konzole.

## 4.7 Testovanie

Aplikáciu Birdyy sme najprv testovali na simulátore iPhone 12 mini priamo v prostredí Xcode tak, že sme pomocou externého mikrofónu snímali nahrávky spevu vtákov z počítača. Model veľmi dobre rozoznával spev vtákov od bežného ruchu prostredia. Predikcie pri speve hrdličky a havrana, ktoré sú výrazne odlišné od ostatných aj od seba navzájom, boli počas testovania vždy správne. Pri ostatných druhoch, ktorých spev je viac podobný, presnosť kolísala.

Následne sme aplikáciu preniesli do skutočného zariadenia iPhone 12 mini, ktoré sme mali k dispozícii. V hardvérovom zariadení boli predikcie modelu obdobné ako v simulátore. Pri prehrávaní nahrávok hrdličky a havrana boli predikcie vždy správne, pri ostatných druhoch presnosť kolísala. Aplikáciu sme vyskúšali aj v reálnom prostredí mesta, kde sa výraznejšie ukázala závislosť správnych predikcií od kvality a citlivosti mikrofónu zariadenia. Model dobre rozoznával ruch prostredia a spev vtákov. Pri určovaní konkrétneho druhu musel byť jeho spev dostatočne výrazný a hlasný, inak ho model nevedel odlíšiť od pozadia, prípadne ho určoval ako nerozpoznaný. Celkovo však aplikácia fungovala bezproblémovo.




Obrázok 18 Testovanie aplikácie Birdyy v reálnych podmienkach

# Záver

Strojové učenie je veľmi široká oblasť, ktorá sa ani zďaleka nedá vyčerpať jednou prácou. Naším cieľom bolo uviesť čitateľa do tejto témy na príklade použitia strojového učenia v mobilných aplikáciách. Vysvetlili sme základnú terminológiu a rámce používané v strojovom učení, ktoré sme neskôr využili na vytvorenie vlastného modelu. Ten rozoznával päť druhov vtákov - hrdlička, havran, sýkorka, vrabec a drozd, podľa ich spevu. Podarilo sa nám vytvorený model Birdyy, ako sme našu aplikáciu/model nazvali, implementovať v prostredí iOS do mobilného zariadenia iPhone a vyskúšať v reálnych podmienkach. Fungovala bezproblémovo a predikcie boli vo veľkej miere správne.

Dokázali sme tým, že využitie strojového učenia v mobilných aplikáciách je dostupné aj pre začínajúcich vývojárov. Pre platformy iOS je možné využiť aplikáciu Create ML, v ktorej na vytvorenie modelu, v tomto prípade hlbokého učenia, nepotrebujeme žiadne znalosti programovania. Obdobným spôsobom je možné využiť aj rôzne online nástroje. Tými sme sa však pre obmedzený rozsah práce nevenovali.

Popísali sme vytvorenie modelu prostredníctvom rámca TensorFlow. Ten ponúka väčšiu kontrolu pri tvorbe modelu, čo so sebou nesie aj potrebu určitých znalostí. Predovšetkým pri spracovaní a predpríprave dát, ktoré zostávajú časovo najnáročnejšou časťou procesu. V našom prípade sme vstupné dáta pre model získavali z audionahrávok transformáciou na Mel-frequency cepstral coefficients. Použitie tohto modelu v aplikácii by pri implementácii vyžadovalo funkciu vykonávajúcu túto transformáciu.

Využitie strojového učenia v mobilných aplikáciách má veľký potenciál. Jeho integrácia do aplikácií je v mnohých ohľadoch jednoduchšia ako v minulosti, stačí na to odhodlanie a základné znalosti programovania. Ako ho využijeme, je iba na našej kreativite.

# Literatúra

[1] Gopalakrishnan, R. - Venkateswarlu, A.: *Machine Learning for Mobile*. Packt Publishing, Birmingham, 2018. ISBN 978-1-78862-935-5

[2] Charu C. Aggarwal: *Neural Networks and Deep Learning. Springer*, 2018. ISBN 978-3-319-94463-0

[3a] Muráň, J.: *Algoritmy strojového učenia I. - Učenie s učiteľom* [online]. [cit. 29-12-2021]. Dostupné na: https://umelainteligencia.sk/algoritmy-strojoveho-ucenia/

[3b] Muráň, J.: *Algoritmy strojového učenia III. - Učenie formou odmeňovania* [online]. [cit. 29-12-2021]. Dostupné na: https://umelainteligencia.sk/algoritmy-strojoveho-ucenia-iii-ucenie-formnou-odmenovania/

[4] scikit-learn [online]. [cit. 02-02-2022]. Dostupné na: https://scikit-learn.org

[5] PyTorch [online]. [cit. 03-02-2022]. Dostupné na: https://pytorch.org

[6] Keras [online]. [cit. 08-02-2022]. Dostupné na: https://keras.io

[7] TensorFlow [online]. [cit. 08-02-2022]. Dostupné na: https://www.tensorflow.org

[8] Crate ML [online]. [cit. 10-02-2022]. Dostupné na: https://developer.apple.com/machine-learning/create-ml

[9] Singh, A., Bhadani, R. - Mobile Deep Learning with TensorFlow Lite, ML Kit And Flutter. Packt Publishing, Birmingham, 2020. ISBN 978-1-78961-121-2

[10] Taigman, Y., Yang, M., Ranzato, M., Wolf, L. - DeepFace: Closing the Gap to Human-Level Performance in Face Verification

[11] Banik, R. - Hands-On Recommendation Systems with Python. Packt publishing, Birmingham. 2018. ISBN 978-1-78899-375-3

[12] Kamath, U., Liu, J., Whitaker, J. - Deep Learning for NLP and Speech Recognition. Springer, 2019. ISBN 978-3-030-14596-5

# Prílohy

## Príloha A: Transformácia audiosignálu na MFCCs

```
import os
import librosa
import math
import json

DATASET_PATH = "dataset_birdyy"
JSON_PATH = "birdyy.json"
SAMPLE_RATE = 44100
DURATION = 5
SAMPLES_PER_TRACK = SAMPLE_RATE * DURATION

def mfcc_transformation(dataset_path, json_path, n_mfcc=13, n_fft=2048, hop_length=1024, num_segments=10):

    # slovník na ulozenie dat
    data = {
        "mapping" : [],
        "mfcc" : [],
        "labels" : []
    }

    num_samples_per_segment = int(SAMPLES_PER_TRACK / num_segments)
    expected_num_mfcc_vectors_per_segment = math.ceil(num_samples_per_segment / hop_length)

    # prechod cez vsetky priecinky v trenovacom datasete
    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(dataset_path)):
        if dirpath is not dataset_path:

            # extrakcia kategorii a ich ulozenie do slovnika
            dirpath_components = dirpath.split("/")
            semantic_label = dirpath_components[-1]
            data["mapping"].append(semantic_label)
            print("\nProcessing {}".format(semantic_label))

            # spracovanie suborov jednotlivych kategorii
            for f in filenames:
```

```
            file_path = os.path.join(dirpath, f)
            signal, sr = librosa.load(file_path, sr=SAMPLE_RATE)

            # mfcc transformacia segmentov a ulozenie hodnot do slovnika
            for s in range(num_segments):
                start_sample = num_samples_per_segment * s
                finish_sample = start_sample + num_samples_per_segment

                mfcc = librosa.feature.mfcc(y=signal[start_sample:finish_sample],
                                            sr=SAMPLE_RATE,
                                            n_fft=n_fft,
                                            n_mfcc=n_mfcc,
                                            hop_length=hop_length)
                mfcc = mfcc.T

                if len(mfcc) == expected_num_mfcc_vectors_per_segment:
                    data["mfcc"].append(mfcc.tolist())
                    data["labels"].append(i-1)
                    print("{}, segment:{}".format(file_path,s))

# ulozenie dat vo formate json
with open(json_path, "w") as fp:
    json.dump(data, fp, indent=4)
```

## Príloha B: Vytvorenie modelu Birdyy pomocou TensorFlow

```
import json
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow.keras as keras
import matplotlib.pyplot as plt

DATASET_PATH = “mfcc_birdyy.json"

# funkcia na import datasetu a konverziu dát
def load_data(dataset_path):
    with open(dataset_path, "r") as fp:
        data = json.load(fp)
    # konverzia dát zo zoznamu do numpy poľa
    X = np.array(data["mfcc"])
    y = np.array(data["labels"])
    return X, y

# funkcia na rozdelenie datasetu na trénovaciu, validačnú a testovaciu množinu
def prepare_datasets(test_size, validation_size):
    X, y = load_data(DATASET_PATH)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size)
            X_train, X_validation, y_train, y_validation = train_test_split(X_train, y_train,
test_size=validation_size)
    return X_train, X_validation, X_test, y_train, y_validation, y_test

# funkcia na vytvorenie modelu
def build_model(input_shape):
    model = keras.Sequential()

    # LSTM vrstvy
    model.add(keras.layers.LSTM(64, input_shape=input_shape, return_sequences=True))
    model.add(keras.layers.LSTM(64))

    # plne prepojená vrstva
    model.add(keras.layers.Dense(64, activation='relu'))
    model.add(keras.layers.Dropout(0.3))
```

```
    # výstupná vrstva
    model.add(keras.layers.Dense(7, activation='softmax'))

    return model

X_train, X_validation, X_test, y_train, y_validation, y_test = prepare_datasets(0.25, 0.2)

input_shape = (X_train.shape[1], X_train.shape[2])
model = build_model(input_shape)

optimizer = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer,
              loss="sparse_categorical_crossentropy",
             metrics=["accuracy"])

model.summary()

history = model.fit(X_train, y_train, validation_data=(X_validation, y_validation), batch_size=32,
epochs=30)

test_error, test_accuracy = model.evaluate(X_test, y_test, verbose=1)
print("Accuracy on test set is {}".format(test_accuracy))
```

## Príloha C: Aplikácia na CD

Priečinok s aplikáciou Birdyy je nazvaný prakticka_cast a obsahuje nasledovné:

- Priečinok Birdyy – obsahuje zdrojové kódy a iné súbory potrebné pre spustenie aplikácie v Xcode.
- Priečinok Model – obsahuje mlmodel Birdyy vytvorený v aplikácii Create ML