Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Aplikace pro rozpoznávání obličeje pomocí neuronových sítí

Bc. Filip Mička

Diplomová práce
2025

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip Mička**
Osobní číslo:	**I22168**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Aplikace pro rozpoznávání obličeje pomocí neuronových sítí**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem práce je vytvořit aplikaci pro rozpoznávání obličeje. Aplikace může být vytvořena buď jako webová, pro mobilní zařízení nebo pro PC. V rámci aplikace by mělo dojít k využití některé z knihoven určených pro strojové učení (např. TensorFlow). Po nastudování dané knihovny bude vytvořena aplikace, která ukáže možnosti dané knihovny např. pro rozpoznávání obrazu. Aplikace zpracuje buď nahrané nebo vyfocené digitální obrázky. Aplikace tak bude umět detekovat obličej, klasifikovat jednotlivé části a rozpoznat např. náladu snímaného objektu. Předpokládá se znalost programovacího jazyka JAVA nebo Kotlin, popř. JavaScript.

Rozsah pracovní zprávy: **50-60 stran**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. *Deep learning*. Cambridge: MIT Press, [2016]. Adaptive computation and machine learning (MIT Press). ISBN 978-026-2035-613.
TUČKOVÁ, Jana. *Úvod do teorie a aplikací umělých neuronových sítí*. Praha: Vydavatelství ČVUT, 2003. ISBN 80-010-2800-3.

Vedoucí diplomové práce: **Ing. Jan Panuš, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **8. listopadu 2023**
Termín odevzdání diplomové práce: **17. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 30. listopadu 2023

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 20. 5. 2025

Filip Mička

PODĚKOVÁNÍ

Zde bych rád poděkoval vedoucímu mé práce za cenné rady a trpělivost během zpracování této práce. Velké poděkování patří také celé mé rodině za jejich podporu nejen při vzniku této práce, ale i v průběhu celého studia.

ANOTACE

Tato práce se zabývá vývojem webové aplikace pro rozpoznávání obličeje s využitím knihovny pro strojové učení TensorFlow.js v prostředí JavaScriptu. V teoretické části je popsána problematika detekce obličeje, strojového učení a konvolučních neuronových sítí. Dále je podrobněji rozebrána knihovna TensorFlow.js včetně průzkumu alternativních knihoven. Praktická část obsahuje návrh zmíněné aplikace a její samotnou implementaci.

KLÍČOVÁ SLOVA

TensorFlow.js, JavaScript, Webové prostředí, Strojové učení, Neuronové sítě, Analýza obličeje

TITLE

Face Recognition Application Using Neural Networks

ANNOTATION

This thesis focuses on the development of a web application for face recognition using the machine learning library TensorFlow.js within the JavaScript environment. The theoretical part describes the topics of face detection, machine learning, and convolutional neural networks. Furthermore, the TensorFlow.js library is analyzed in more detail, including a comparison with alternative libraries. The practical part includes the design and implementation of the application.

KEYWORDS

TensorFlow.js, JavaScript, Web Environment, Machine Learning, Neural Networks, Face Analysis

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK

DOM	Document Object Model
API	Application Programming Interface
GPU	Graphics Processing Unit
CPU	Central Processing Unit
UML	Unified Modeling Language
SPA	Single-Page Application
ANN	Artificial Neural Network
FNN	Feedforward Neural Network
CNN	Convolutional Neural Networks
JSON	JavaScript Object Notation
ML	Machine Learning
UX	User Experience

ÚVOD

V posledních letech se stále více využívají technologie strojového učení v oblasti počítačového vidění. Jedním z častých využití je detekce obličejů, nad kterými lze následně provádět analýzu různých vlastností, jako je věk, pohlaví nebo emoce. Tyto poznatky nacházejí uplatnění v celé řadě aplikací – od bezpečnostních systémů, přes marketing, až po personalizované služby.

Aby bylo možné lépe porozumět základům těchto technologií a principů, na kterých stojí, je teoretická část práce rozdělena na několik navazujících celků. Na začátku jsou popsány základní principy detekce obličeje. Dále je představeno, jak obecně funguje strojové učení. Popsány jsou zde jednotlivé typy strojového učení a několik stěžejních pojmů z této oblasti. Následně je podrobněji rozebrán princip fungování konvolučních neuronových sítí, které jsou efektivní pro zpracování obrazových dat a tvoří podstatnou část praktické části této práce.

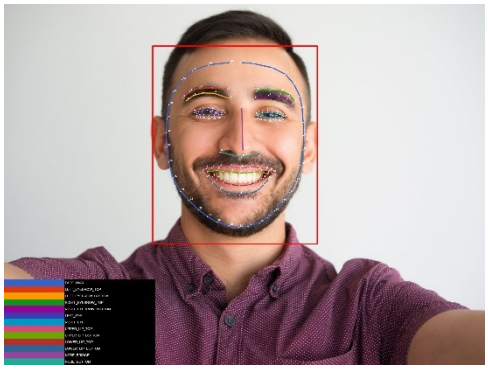
Cílem praktické části je pak vytvoření aplikace pro rozpoznávání obrazu. Pro možnost snadné publikace bude zvoleno řešení pomocí aplikace pro webové prostředí. Dále bude nutné zvolit vhodnou knihovnu strojového učení. Zvolená knihovna bude představena v závěru teoretické části, a to včetně možných alternativ.

Rozpoznávání obrazu bude zaměřeno na obličejová data. Aplikace tak bude umět detekovat obličeje v obrazových datech a provádět analýzu obličeje ve formě rozpoznání nálady, klíčových částí obličeje, pohlaví nebo odhadu věku osoby. V případě, že knihovna pro dané účely neposkytuje žádné řešení, bude vytvořeno vlastní v podobě modelu strojového učení pro daný účel.

V rámci vytváření modelů tak bude představeno několik možností tvorby modelů strojového učení. Představeny budou možnosti pomocí API Keras, a to natrénování modelu na známé topologii konvoluční neuronové sítě a natrénování modelu s vlastní topologií konvoluční neuronové sítě. Dále bude představena možnost vytvoření modelu pomocí nástroje Google Teachable Machine.

1 ROZPOZNÁVÁNÍ OBLIČEJE V OBRAZE

Je technologie, pomocí které se určuje přítomnost obličejů, v obrazových datech. V případě nalezení pak pozice a velikost části obrazu obsahující obličej. Výstupem je separovaná část obrazu obsahující pouze obličejovou část. Takovýto výstup může být dále využit pro další analýzu obličeje (např. detekce emoce, pohlaví, autentizace). Ve své podstatě se tedy jedná o speciální případ detekce objektů v obraze. Ačkoliv pro člověka je tento úkol velmi jednoduchý, z hlediska počítače, se nejedná o triviální úlohu. [1] [2]



Obrázek 1: Detekce obličeje. Zdroj [3].

1.1 Využití

- **Klasifikace pohlaví** – Z detekovaného obličeje lze detekovat, zda má obličej mužské nebo ženské rysy.
- **Fotografování** – Fotoaparát dovede detekovat obličej ve snímku a následně na něj automaticky zaostřit. Dalším příkladem může být automatické třídění fotografii do alb, to umožňuje snadnější vyhledávání fotografií.
- **Extrakce rysů** – Určení pozice očí, nosu, pusy nebo uší, pokud jsou viditelné, lze využít například v oblasti virtuální reality.
- **Biometrika** – Ověření identity osoby na základě detekovaného obličeje, běžně používané například v mobilním telefonu.
- **Marketing** – Z detekovaných obličejů lze například nabízet cílené reklamy na základě detekovaného věku, pohlaví, či nálady. [1]

1.2 Vývoj

První pokusy o detekci obličeje sahají na počátek 70. let. Zpočátku se využívalo různých heuristik a antropometrií. Tyto metody nenabízely příliš velkou flexibilitu a jakákoliv anomálie ve snímku zapříčinily nesprávnou funkčnost řešení, kdy byla nutná jeho revize. To mělo za

příčinu, že vývoj poměrně ustrnul až do 90. let. Následně začaly vznikat sofistikovanější segmentační systémy se snahou o zobecnění. Větší využití statistiky a strojového učení přineslo další zlepšení v podobě lepší flexibility, kdy se zdokonalila detekce ve složitějších snímcích. [4]

1.3 Omezení

Existuje několik důvodů, které mohou způsobit zhoršení úspěšnosti a přesnosti detekce obličeje v obraze.

- **Kvalita obrazu** – je častou příčinou, kdy obraz může být v některých částech příliš přesvětlený nebo naopak tmavý a může tak být složité rozpoznat důležité rysy obličeje. Další příčinou může být zhoršení kvality obrazu v podobě rozmazanosti nebo špatného rozlišení.
- **Zakrytí obličeje** – obličej může být v obraze v různých jeho částech zakryt nějakým objektem. Nejčastěji brýlemi, rouškou, vlasy nebo rukou.
- **Pozadí** – pozadí za obličejem, který je detekován může obsahovat množství různých objektů, které poté mohou algoritmu způsobovat komplikace.
- **Množství obličejů** – v obraze může být hned několik obličejů pro detekci. Obličeje mohou být blízko sebe nebo se mohou různě částečně překrývat a zhoršovat tak úspěšnost detekce.
- **Orientace obličeje** – obličej může být v obraze různě natočen.
- **Variace výrazů** – lidský obličej může vytvořit nespočetné množství různých výrazů a algoritmus detekce tak nemusí vždy dobře zafungovat.
- **Barva pleti** – u obličeje s tmavší pletí nemusí být typické rysy pro obličej tak výrazné.
- **Vzdálenost obličeje od kamery** – obličej může být příliš daleko od kamery a nemusí tak být dobře rozpoznatelné jeho rysy. [1]

Obrázek 2: Problémy při detekci obličejů. Zdroj [5].

1.4 Principy detekce obličeje

Pro detekci obličeje v obrazu lze využít dva základní přístupy s poněkud odlišnou strategií. Přístupy se odlišují na základě toho, jakým způsobem jsou využívány znalosti o obličeji.

Přístup založený na rysech – „feature-based“

Metoda funguje na principu extrakce charakteristických rysů obličeje – oči, nos, pusa, které jsou detekovány například detektory hran. Tyto rysy jsou následně porovnány s předem známými poznatky o obličejových rysech. Jsou tak využívány geometrické techniky jako měření vzdálenosti a úhlů mezi obličejovými rysy v obraze. Cílem tedy je definovat přítomnost obličeje na základě prostorového uspořádání obličejových prvků. [1]

Přístup založený na obrazech – „image-based“

Druhou možností je metoda založená na znalostech o obličeji získaném z velkého množství obrazů. Tato technika nepracuje přímo se znalostmi o konkrétních rysech obličeje. Místo toho využívá vzory naučené z velkého množství trénovacích obrázků. S pomocí metod statistického zpracování dat a strojového učení modely rozpoznávají typické vizuální charakteristiky obrázků obsahujících obličej a obrázků bez obličeje. Model tak ví, jak obrázek, kde se obličej vyskytuje typicky vypadá, a může tak definovat, zda se v nově předloženém obraze obličej vyskytuje nebo ne. [1] [4]

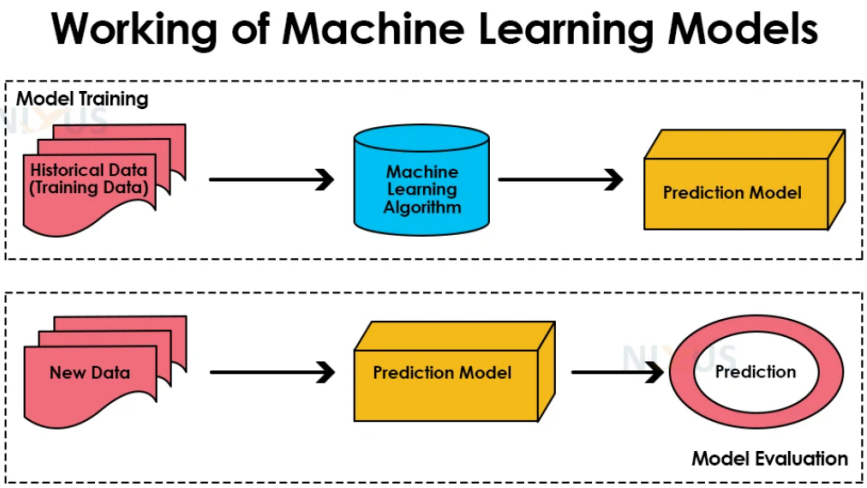
S ohledem na název práce bude v praktické části využíváno právě přístupu „image-based“, kde jednou z technik tohoto přístupu je právě využití umělých neuronových sítí. Ačkoliv tato technika je zde uvedena v souvislosti s detekcí obličeje, neuronové sítě lze využít i k následujícím analýzám detekovaných obličejů jako je detekce pohlaví, emoce, jednotlivých rysů nebo i odhadnutí věku. V následujících kapitolách proto budou představeny základní principy strojového učení a jednoho konkrétního typu neuronové sítě, jež budou následně i použity v praktické části – aplikaci.

2 STROJOVÉ UČENÍ

Pro definování strojového učení se hodí přirovnání k učení člověka. Zde to lze definovat jako osvojování si jistých znalostí či dovedností, a to na základě předchozí zkušenosti. Jednoduchým příkladem je učení malých dětí. Ty zpočátku také neumí například určit, zda zvíře na obrázku je pes, nebo kočka. Až na základě opakovaného vysvětlování rodičů dítě získá zkušenost a dovede již samo rozpoznávat o jaký druh zvířete se jedná. Podobně jako u člověka je to i v počítačovém prostředí.

Nabízí se i srovnání učení člověka a počítače. Výhodou počítače je možnost zpracovávat obrovské množství dat a poskytovat poznatky z nich. Je vysoce specializován na konkrétní úkoly, ovšem citlivý na nové a neočekávané situace. Oproti tomu je člověk mnohem více flexibilní, je schopen dovozovat si kontextové informace a zobecňovat poznatky. Nelze tedy jednoznačně říct, kdo je lepší. V praxi se vzájemně doplňují. [6]

Strojové učení nebo zkráceně ML z anglického názvu Machine Learning je proces, který umožňuje počítači získávat znalosti z dat. Tento proces zahrnuje poměrně širokou škálu algoritmů strojového učení, a to od jednodušších jako je lineární regrese, až po složitější – například umělá neuronová síť. Výstupem je pak model strojového učení, který vzniká aplikací konkrétního algoritmu na konkrétní data – trénovací dataset. Model tak lze definovat jako algoritmus strojového učení, jehož parametry jsou během trénování vhodně nastaveny pro konkrétní úlohu. Na natrénovaný model pak lze pohlížet jako na program, který na základě předložených vstupů dává výstupy tzv. predikce jež se liší dle konkrétní úlohy. [7]



Obrázek 3:Schéma principu strojového učení. Zdroj [8].

2.1 Úlohy pro strojové učení

Strojové učení nachází široké uplatnění v mnoha oblastech, přičemž lze jeho úlohy rozdělit do několika základních kategorií:

- **Klasifikace** – Modelu jsou předkládány vstupy, běžným výstupem je vstupu přiřadit pravděpodobnost příslušnosti k předem známým třídám.
- **Regrese** – Model se snaží předpovídat číselnou hodnotu na základě vstupních dat.
- **Transkripce** – Úkolem modelu je převod dat z nestrukturované podoby (například audio) do textové podoby.
- **Překlad** – Model převádí řetězce symbolů jednoho jazyku do jiného.
- **Detekce anomálií** – Model slouží k procházení setu dat a označování těch které jsou neobvyklé.
- **Shluková analýza** – Model se snaží o nalezení skupin (shluků v datech) na základě podobnosti jednotlivých instancí dat bez znalosti charakteru a počtu těchto skupin. [9]

2.2 Data

Pro vytvoření modelu strojového učení je nutné nejprve získat trénovací dataset, neboli soubor instancí. Ty mají své vlastnosti, neboli atributy, které mohou nabývat určitých hodnot a popisují jednu konkrétní instanci. Instance mohou být také označeny správnou výstupní hodnotu pro danou instanci. Správná výstupní hodnota bude dále označována jako etiketa. Zda je etiketa potřebná nebo ne určuje typ úlohy na jakou je model zaměřen a typ učení, který je využit. Jednoduše lze jednotlivé pojmy představit názorným příkladem na datasetu ovoce, kdy by cílem bylo vytvořit klasifikační model určující o jaký druh ovoce se jedná. Dataset by tak mohl obsahovat například atributy barva, tvar nebo hmotnost. Jednotlivé instance by pak obsahovali etiketu s druhem ovoce (jablko, banán, hruška). [10 str. 3]

Data bývají často ve špatném stavu, kdy obsahují nepřesnosti, či chyby. Proto je vhodné data nejprve také zkontrolovat a případně předpřipravit. To zahrnuje vyčištění dat, kam patří nalezení a ošetření chybějících hodnot a nalezení odlehlých hodnot, které se výrazně liší od ostatních a mohou tak obsahovat chybné údaje. Nutné je také odstranit nekonzistentní hodnoty. Dalším přípravným krokem je transformace dat, kam patří standardizace numerických dat nebo převod kategorických, textových dat do číselné podoby použitelné ve strojovém učení. V poslední přípravné fázi se dataset dělí na dvě části. Na trénovací množinu, která slouží pro natrénování modelu a testovací množinu pro následné ověření, jaké úspěšnosti bude model

dosahovat na nových datech. Na testování se obvykle vyčleňuje 20-30 % z celkového počtu instancí dat. [11]

2.3 Typy strojového učení

Strojové učení se dělí do dvou hlavních kategorií. Každý typ má své specifické využití a vyžaduje jiný přístup k trénování modelů. V následujících podkapitolách jsou popsány tyto dva základní přístupy i s možností jejich kombinace. Pro každý z těchto přístupů je uveden princip fungování, typické příklady využití a přehledově konkrétní používané algoritmy.

2.3.1 Učení s učitelem

Prvním typem strojového učení je učení s učitelem. Trénovací algoritmy zde potřebují ke svému fungování dataset obsahující etikety. Princip je založen na tom, že algoritmus nejdříve analyzuje instanci dat, vyhledá vzory a poté vytvoří predikci výstupu, kterou následně porovná s etiketou instance dat zastupující roli učitele. Na základě tohoto porovnání vypočítá chybu výsledku, podle které jsou upravovány parametry algoritmu. Tento postup se opakuje a algoritmus postupně mění parametry tak, aby dosahoval stále lepších výsledků pro danou úlohu. Výsledný model tedy vzniká konečným nastavením parametrů algoritmu, kdy je následně možné modelu předkládat nová data pro vytváření predikcí. Učení s učitelem se používá typicky pro úkoly typu klasifikace (binární i vícetřídní) nebo regrese. Využívá například algoritmů KNN, Bayesovské klasifikace, nebo pokročilejší algoritmy jako jsou umělé neuronové sítě. [12]

2.3.2 Učení bez učitele

Tento typ učení využívá algoritmy pro identifikaci skrytých vzorů v datech. Je zde využíván dataset bez etiket, ty se snaží algoritmus v datech nalézt. Cílem je tak detekovat korelaci a vztahy mezi daty ve velkých datasetech. V menších datasetech by člověk mohl korelace nalézt svépomocí, avšak v případě obrovských datasetů je to pro člověka téměř nemožné. Využívá se tedy schopnost počítačů zpracovávat obrovské množství dat. [12]

Jelikož nejsou správné výstupy známy, nelze tak zde měřit přesnost modelu jako při učení s učitelem. Vyhodnocení lze dělat pomocí různých metrik, které poskytují algoritmy k tomu určené. Příkladem je Calinsky-Harabasz index. [13]

Učení bez učitele se používá pro úkoly typu hledání shluku v datech nebo pro snížení dimenzionality dat, jejíž cílem je sloučením důležitých atributů při zachování vlastností, což přináší poté nižší výpočetní požadavky při dalším zpracování. Algoritmy využívané pro tento

typ učení jsou například PCA (Principal Component Analysis), nebo LDA (Linear Discriminant Analysis). [12]

2.3.3 Kombinované učení

Jedním z možných problémů pro učení s učitelem je tzv. „overfitting“. To může nastat při použití malého množství dat, kdy se model naučí v datech hledat špatné vzory. Jednoduše tento problém lze demonstrovat na příkladu klasifikace obrázků psů a koček. Pokud bude většina obrázků koček pořízena v domácím prostředí a většina psů ve venkovním prostředí, model si tak může mylně spojit venkovní prostředí se psem a vnitřní s kočkou. Následně při předkládání koček ve venkovním prostředí je může označovat za psy. Řešením může být zvětšení datasetu. Ovšem ne vždy může být dostupné větší množství dat s etiketou. Kombinované učení přináší řešení tohoto problému. Využívá se menšího množství dat s etiketou a velkého množství dat bez etikety. Aby bylo možné použít data bez etiket pro trénování modelu s učitelem, je nutné nejdříve nějakým způsobem pro ně etikety získat. [14]

První možností je natrénovat model na datech s etiketami a pomocí tohoto modelu získat tzv. pseudo etikety pro data bez etiket. Další možností, jak získat očekávané výstupy k datům, je pomocí shlukování s učením bez učitele. Takto je možné zjistit, že data bez očekávaného výstupu patří do některého shluku, ve kterém budou i data, u kterých je známa etiketa, kdy v daném shluku ji budou mít všechna data stejnou. V obou případech získaná etiketa je správná jen s určitou pravděpodobností. Pro výstupy, které mají nízkou pravděpodobnost správnosti je možné dále využít aktivního učení a zapojit člověka, který určí správný výstup. Další zefektivnění u kombinovaného učení je také možné pomocí snížení dimenzionality dat učením bez učitele. [14]

2.4 Chybová funkce („Loss function“)

Ve strojovém učení udává, jak moc se model svými výstupy liší od správné hodnoty. Pokud je model ve svých predikcích přesný, hodnota funkce je nízká. Naopak pokud se mýlí, hodnoty ztrátové funkce jsou vysoké. Tato funkce je spojena s učením s učitelem, jelikož je pro výpočet potřebné znát správný výstup. Cílem při trénování je nastavit modelu parametry tak, aby hodnota chybové funkce byla co nejmenší. Kromě toho, že chybová funkce je vhodná pro vyhodnocení přesnosti modelu, detekci přeučení a podučení modelu – vysvětleno níže, slouží také pro učení modelu, kde pomocí jeho gradientu (derivace) jsou upravovány parametry modelu. Chybových funkcí existuje více typů. Volba typu chybové funkce záleží na typu úlohy. Níže jsou uvedeny nejpoužívanější funkce a jejich stručný popis. [15]

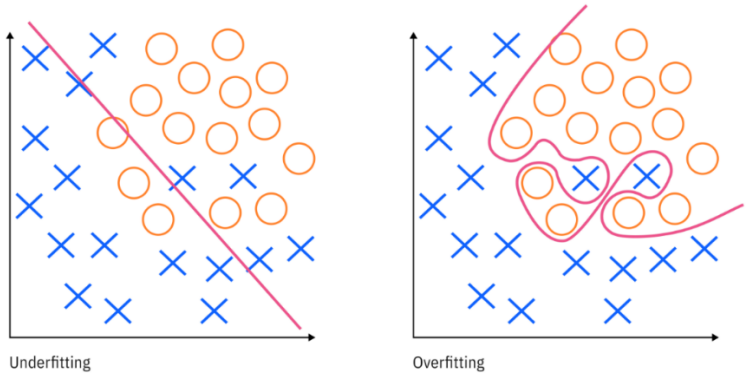
- **Mean squared error (MSE)** – Používá se pro úlohy typu regrese. Funkce udává průměrnou čtvercovou odchylku mezi skutečnou a predikovanou hodnotou. Umocnění odchylky přináší výhodu v efektivnějším učení, model se snaží vyhnout velkým chybám, které jsou více penalizovány. Ovšem tím je pak učení modelu citlivé na odlehlá, chybná data, která mohou způsobit o to vetší nechtěné změny v modelu.
- **Mean absolute error (MAE)** – Používá se pro úlohy typu regrese, kde udává průměrnou absolutní odchylku mezi skutečnou a predikovanou hodnotou. Oproti předešlé funkci není tolik náchylná na odlehlá, chybná data. Proto se hodí tam, kde se nějaká taková data potenciálně mohou vyskytnout.
- **Binary cross-entropy** – Tato funkce se využívá v binárních klasifikačních úlohách. Udává rozdíl mezi skutečnou hodnotou (0 nebo 1) a pravděpodobnostním odhadem modelu, což je pro tuto úlohu standartní výstup. Pokud tak model udá nesprávný výsledek s vysokou mírou jistoty, penalizace při učení je vyšší, než pokud by se mýlil s nízkou mírou jistoty.
- **Categorical cross-entropy** – Funkce je svou povahou velmi podobná předchozí s rozdílem, že přináší zobecnění pro vícetřídní klasifikaci, kde je také využívána. [15]

2.5 Epocha a velikost dávky

Epocha označuje jeden průchod všech instancí trénovacích dat algoritmem, respektive modelem, který se učí. Délka trénování modelu tak bývá určena počtem epoch. Velikost dávky (batch size) určuje, po průchodu kolika instancí bude provedena průběžná úprava parametrů algoritmu. Pro co nejmenší chybovost výsledného modelu je nutné zvolit správnou hodnotu těchto argumentů. Při volbě špatného počtu epoch může nastat jeden z problémů, které jsou známé jako přeučení a podučení. Nízká hodnota velikosti dávky přináší rychlejší učení, které je způsobeno častější aktualizací parametrů, ale vlivem náhodné odchylky v datech mohou být prováděny příliš velké úpravy parametrů modelu. To způsobuje menší stabilitu učení. Analogicky pro vysokou hodnotu velikosti dávky je potom učení pomalejší, ale stabilnější. Hodnoty obou argumentů se často volí metodou pokus-omyl. Další možností pro správné nastavení počtu epoch je pro učení využít metody „early stopping“, kdy se učení nezastaví až po předem definovaném počtu epoch, ale v momentě, kdy již neklesá hodnota chybové funkce na validačních datech – model by se již začal přeučovat. [16]

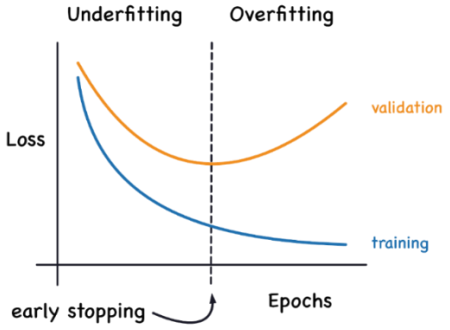
2.6 Přeučení a podučení

Poměrně častým problémem při trénování modelu je přeučení („overfitting“) a podučení („underfitting“). Přeučení nastává, pokud se model příliš přizpůsobí trénovacím datům a ztratí tak schopnost generalizace poznatků na nově předložená data, na kterých pak selhává. Naopak podučení nastává, když model nedostatečně zachycuje vzory v datech. [17]



Obrázek 4: Grafické znázornění přeučení a podučení. Zdroj [17].

Typickým rysem přeučení je, když model chybuje velmi málo na trénovacích datech, přičemž na nových datech – testovacích dosahuje podstatně vyšší chybovosti. Na chybové funkci se projeví přeučení tak, že na testovacích datech začíná růst, zatímco na trénovacích datech dále klesá. Příčin přeučení může být více. Často je to příliš vysoký počet epoch nebo příliš složitý model – například příliš mnoho vrstev neuronové sítě. Aspektem, který tento problém podněcuje je také kvalita datasetu, kdy modely trénované na malých a šumem zatížených trénovací datasetech jsou náchylnější na přeučení. Podučení lze na chybové funkci spatřovat v trvale vysokých hodnotách, a to jak na testovacích, tak i trénovacích datech. Příčin podučení může být opět několik, mezi které patří příliš malý počet epoch, příliš jednoduchý model nebo špatná kvalita dat. [17]



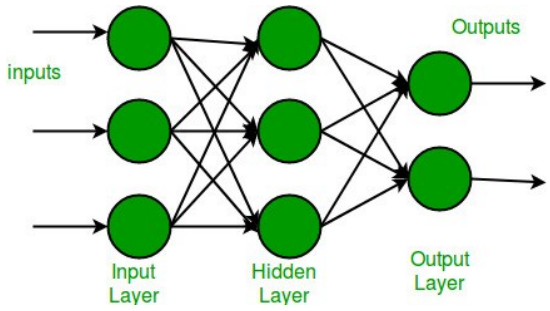
Obrázek 5: Znázornění problému přeučení a podučení na chybové funkci. Zdroj [18].

3 KONVOLUČNÍ NEURONOVÉ SÍTĚ

Zkráceně CNN z anglického názvu Convolutional Neural Networks je typ umělé neuronové sítě, který je vhodný a často používaný pro zpracování obrazových dat. Jedná se o specifický typ algoritmu strojového učení spadající do kategorie učení s učitelem. Řadí se také do odvětví strojového učení označovaného jako hluboké učení. To označuje část strojového učení, kde se využívá hlubokých neuronových sítí – sítě s vícero skrytými vrstvami neuronů. [19]

Pro lepší pochopení konvolučních neuronových sítí je důležité lehce nastínit, jak funguje umělá neuronová síť (ANN), konkrétněji jeden z jejích typů – dopředná neuronová síť (FNN).

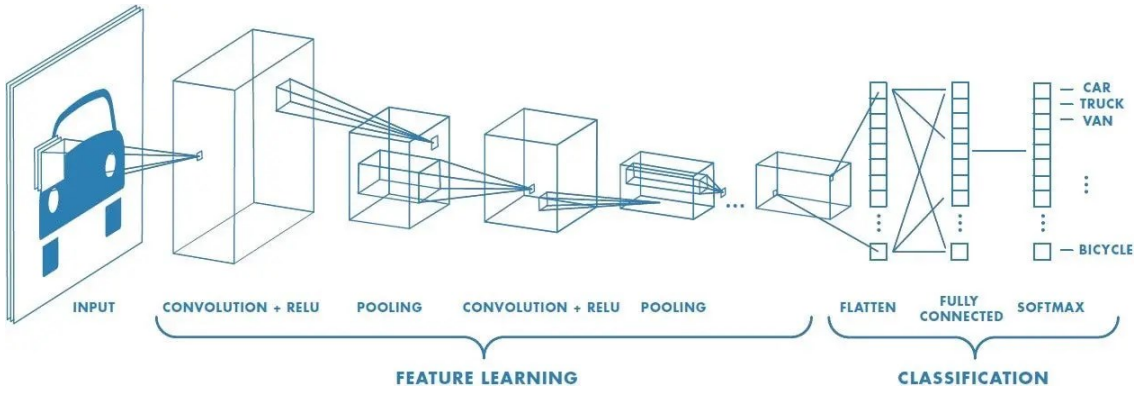
Neuronová síť je tvořena umělými neurony, které jsou rozděleny do vrstev. Umělý neuron si lze představit jako výpočetní jednotku, která přijímá vstupy, provede jejich zpracování a následně produkuje výstupy. U dopředné neuronové sítě existuje propojení mezi neuronem jedné vrstvy a neurony vrstvy následující. Jinými slovy výstup neuronu jedné vrstvy je ovlivněn výstupy všech neuronů předešlé vrstvy. První vrstva sítě, kterou vstupují data do sítě, je označována jako vstupní vrstva. Následně data dále prostupují dalšími skrytými vrstvami neuronové sítě, až k poslední výstupní vrstvě sítě – tok pouze jedním směrem u FNN. Spojení mezi neurony je ohodnoceno vahou, která udává důležitost tohoto spojení. Výpočet výstupní hodnoty jednotlivých neuronů je tak ovlivněn váženými vstupními hodnotami a aktivační funkcí. Jaký má význam aktivační funkce bude vysvětleno v textu níže. Neuronová síť je učená pomocí učících algoritmů. Cílem těchto algoritmů je v průběhu trénování sítě postupně upravovat váhy spojení mezi neurony pro dosažení co nejlepších výsledků pro požadovanou úlohu. Váhy spojení tak tvoří znalosti umělé neuronové sítě. [20]



Obrázek 6: umělá neuronová síť. Zdroj [21].

Konvoluční neuronové sítě řeší problémy výše zmíněné, dopředné neuronové sítě. Zde při vyhodnocování vizuálních dat dochází k rychlému nárustu parametrů (vah neuronů). Jelikož je každý neuron sítě propojen se všemi neurony vrstvy následující, například jediný neuron první vrstvy musí mít počet vah, který je roven počtu pixelů v obrázku – pro barevný obrázek ještě

3x více, pro jednotlivé kanály. To přináší výpočetní a paměťovou náročnost. Další problém, který mají klasické neuronové sítě je overfitting neboli přeučení. Vzhledem k velkému počtu parametrů má síť tendenci se naučit trénovací data příliš detailně, přičemž ztrácí schopnost zobecňování, čímž selhává na nově předložených datech. Konvoluční sítě přináší značné vylepšení, kdy pracují s malými částmi obrazu, výrazně snižují počet parametrů, a tak i výpočetní náročnost. Jsou schopny lépe zachovat prostorové vztahy v obraze a efektivněji se učit. Struktura sítě zpravidla obsahuje několik, po sobě jdoucích vrstev. První část sítě je tvořena vrstvami konvolučními a pooling, které se zpravidla opakují. Tyto vrstvy zajišťují extrakci lokálních rysů obrazu. Druhou částí je již zmíněná dopředná síť, sloužící k rozhodnutí o konečném výstupu z extrahovaných rysů. Ačkoliv tedy typické úlohy těchto sítí jsou stejné – klasifikace a regrese, tak pro obrazová data je vhodnější použít CNN. [22]

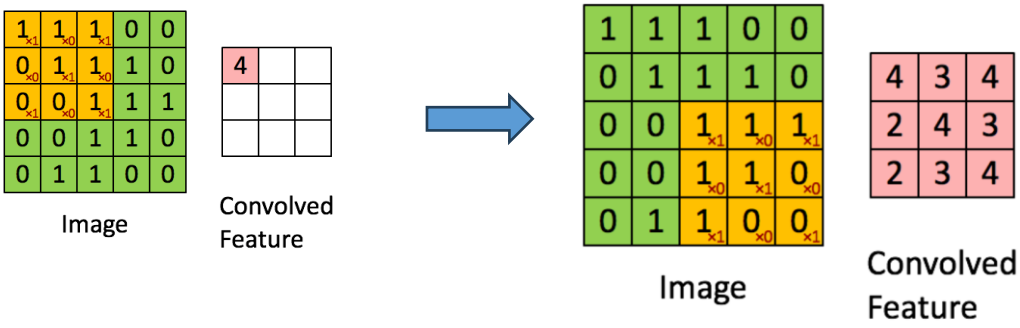


Obrázek 7: Zpracování obrazu konvoluční neuronovou sítí. Zdroj [23].

3.1 Konvoluční vrstva

Je typickou vrstvou pro tuto síť. Několik těchto vrstev za sebou umožňuje konvoluci, neboli extrakci lokálních klíčových vlastností z obrazu, jako jsou například různé hrany nebo tvary. Každá tato vrstva je tvořena filtry (detektory), které se posouvají s určitým krokem po vstupní matici. Vstupní matice je u první konvoluční vrstvy samotný obraz, u dalších konvolučních vrstev je to výstup předešlé vrstvy. Velikost kroku filtru je označována jako stride. Filtr je reprezentován maticí čtvercového formátu (běžně například 3x3), jejíž hodnoty reprezentují klíčovou vlastnost. Na každé pozici v obrazu je prováděn skalární součin hodnot matice filtru s hodnotami v obrazu. Výstupem konvoluční vrstvy je matice nazývaná matice vlastností. Jednotlivé prvky matice vlastností lze chápat jako umělé neurony, přičemž filtr reprezentuje váhu spojení. Vstupy neuronů nejsou tvořeny všemi hodnotami vstupní matice, proto tyto vrstvy lze označit jako síť, která není plně propojena. Výše zmíněná výhoda konvoluční

neuronové sítě ve snížení počtu parametrů je způsobena nejen tím, že konvoluční vrstvy nejsou plně propojeny, ale také tím, že neurony v jedné vrstvě sdílejí váhy (používají stejný filtr). Důležité je také připomenout, že i zde se síť při učení snaží najít ideální váhy, zde tedy hodnoty matice filtrů neboli správné detektory. [24]



Obrázek 8: Princip fungování konvoluční vrstvy. Zdroj [23], upraveno.

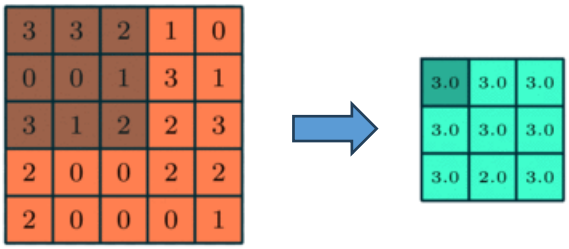
3.2 Aktivační funkce

Ačkoliv se nejedná přímo o vrstvu, aktivační funkce je důležitou součástí neuronových sítí, kde slouží především k zavedení nelinearity. Ta je důležitá, protože umožňuje síti zachytit nelineární vztahy mezi vstupem a výstupem. Jednoduchým příkladem pro představu může být detekce psa v obrázku, kde přítomnost psa nelze popsat lineárně, protože pozice zvířete, jeho vzhled a spoustu dalších vlastností je značně variabilní. Aktivační funkce se u konvolučních neuronových sítí zpravidla využívá za každou konvoluční vrstvou, kde je aplikována na jednotlivé prvky (neurony) matice vlastností. Dále se používá pro neurony plně propojené vrstvy. Existuje několik typů aktivačních funkcí, přičemž nejpoužívanější je ReLU. Poněkud odlišnou funkci má aktivační funkce ve výstupní vrstvě. Zde se často využívá funkcí Softmax, nebo Sigmoid. [25]

- **ReLU** – Převádí záporné hodnoty na nulu, kladné zůstávají beze změny. Existují alternativy jako ELU nebo Leaky ReLU.
- **Softmax** –Funkce pro vícetřídní klasifikaci. Převádí výstupní hodnoty neuronů výstupní vrstvy, které reprezentují jednotlivé třídy, na hodnotu pravděpodobnosti příslušnosti dané třídy.
- **Sigmoid** – Funkce určená pro účely binární klasifikace. Převede výstup na pravděpodobnost přítomnosti objektu. [25]

3.3 Pooling vrstva

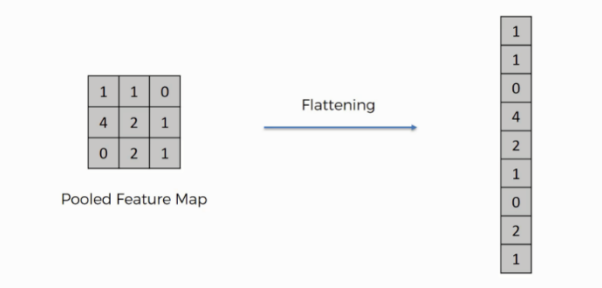
Tyto vrstvy bývají umístěny za konvoluční vrstvou a po vykonání aktivační funkce nad neurony. Slouží pro extrakci dominantních vlastností z určité části mapy vlastností. Sjednocuje skupiny vlastnosti a vybírá z nich tu nejvíc dominantní. Takzvaný pooling bývá prováděn nad čtvercovou částí, typicky oblast 2x2. Klíčovou funkcí této vrstvy je tedy snížit objem dat, přičemž ale zachovat dominantní vlastnosti. Nejčastěji používanými typy jsou max pooling, který z matice vlastnosti pro danou oblast vybírá maximální hodnotu a average pooling, který počítá průměrnou hodnotu dané oblasti. [24]



Obrázek 9: Princip fungování max pooling vrstvy. Zdroj [23], upraveno.

3.4 Flatten vrstva

Je poměrně jednoduchou vrstvou jejíž jediným účelem je převedení hodnot z dvourozměrného prostoru (matice) na vektor. Tato vrstva je potřebná pro přípravu dat pro další vrstvu, která potřebuje vstupní data právě v tomto formátu. [24]



Obrázek 10: Princip fungování flatten vrstvy. Zdroj [26].

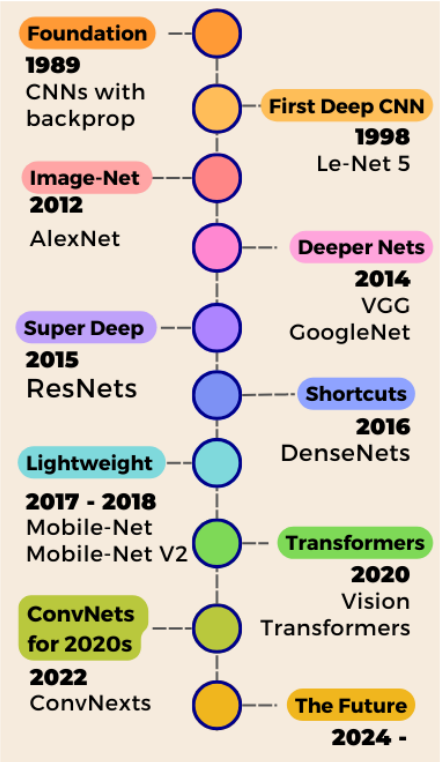
3.5 Plně propojené vrstvy

Vrstva, která následuje po konvolučních vrstvách je tvořena dopřednou vícevrstvou neuronovou sítí, jež byla zmíněna v úvodu této kapitoly. Zatímco cílem konvolučních vrstev je extrahovat rysy obrazu, zde je již cílem z těchto rysů rozhodnout o výsledku. Neurony vstupní vrstvy této dopředné vícevrstvé neuronové sítě jsou propojeny se všemi výstupy předchozí flatten vrstvy. To umožňuje síti pracovat s globálními vztahy mezi jednotlivými rysy. Podoba

poslední vrstvy celé sítě závisí na typu úlohy, kterou síť řeší. Pokud například řeší problém vícetřídní klasifikace, obsahuje tolik neuronů, kolik je tříd. Pro regresní úlohy je tvořena jedním neuronem pro předpověď spojité hodnoty. [27]

3.6 Topologie

Při volbě topologie neuronové sítě je poměrně velké množství různých možností struktury sítě. Hledání vhodné varianty může zabrat poměrně hodně času, kdy je nutné vždy síť natrénovat a zjistit, jakých výsledků dosáhla. Proto běžnou praktikou je zvolit některou z již známých a osvědčených topologii. [24] Na obrázku níže je uveden stručný přehled známých topologií, včetně doby jejich vzniku.



Obrázek 11: Historický vývoj používaných topologií. Zdroj [28].

4 KNIHOVNA TENSORFLOW.JS

Knihovna byla vyvinuta společností Google. Vznikla na základě popularity jazyku JavaScript a nárustu využití strojového učení ve webových aplikacích. Knihovna je určená pro jazyk JavaScript. Je využívána pro algoritmy strojového učení stejně jako samotná knihovna TensorFlow. Je ovšem přizpůsobena pro fungování v prostředí webového prohlížeče. [29]

4.1 Výhody

Obecnou výhodu, kterou přináší všechny knihovny strojového učení určené pro prohlížeče je jednoduchost publikace, kdy uživatel nemusí nic instalovat, pro využívání aplikace mu postačí webový prohlížeč. Dále pak standardizovaný a jednoduchý přístup k hardwarovým komponentám zařízení, kterým je nejčastěji kamera, či mikrofon. [29]

Konkrétní výhody této knihovny spočívají v široké podpoře možných backendů (tento pojem bude představen níže) a možnosti použití různých architektur, kde především možnost běhu v prohlížeči bez serverové infrastruktury je vítaným přínosem. Výhodou je také poměrně široký ekosystém TensorFlow, kde je oficiálně podporována a dokumentována konverze modelů z TensorFlow do TensorFlow.js. Další výhodou jsou snadná integrace knihovních modelů s DOM elementy, odpadá tak nutnost manuálního převodu dat. V neposlední řadě je kladem knihovny také to, že je vyvíjena známou společností Google. To přináší benefity v podobě dlouhodobé stability projektu a aktualizací. [30]

4.2 Nevýhody

Mezi obecné nevýhody webového řešení patří především omezení v podobě nižší výkonosti. JavaScript jakožto interpretovaný jazyk nedosahuje výkonosti kompilovaných jazyků jako například C++, které naopak lze připojit při využití Pythonu. Dále pak při běhu v prohlížeči není z bezpečnostních důvodů možné přímo přistupovat ke grafické jednotce zařízení. Ta je využívána často k numerickým výpočtům. Využívají se zde pro přístup API WebGL nebo WebGPU, ale výkonost je stále nižší než při přímém přístupu ke GPU. Celkově se pak výkonnost odvíjí od konkrétního zařízení. [29]

4.3 Tensor

Důležité je zmínit s jakým typem dat knihovna pracuje. TensorFlow.js využívá tensory jako reprezentaci číselných dat ve strukturované podobě. Slouží k předávání informací z a do modelu. Tyto tensory odpovídají matematickému pojetí vícerozměrných polí a jsou uzpůsobeny pro efektivní numerické zpracování v prostředí knihovny. [31]

4.4 Architektura a výběr backendu

Knihovna nabízí flexibilitu při volbě architektury. Je možné vytvářet client-side aplikace běžící v prohlížeči na straně klienta. Druhou možností je vytvářet aplikace typu server-side, kdy se o běh aplikace stará server, zde typicky Node.js. [29] Ačkoliv možnost client-side většinou nedisponuje výkonností jako robustnější serverové řešení, hlavní výhody lze spatřovat ve větší bezpečnosti – například citlivá data nejsou nikam odesílána, absenci latence, která může nastávat při přenosu dat z klientského zařízení na server nebo také v jednodušší škálovatelnosti. [30]

Při využití řešení client-side je důležité také zmínit pojem backend. Tím se v TensorFlow.js rozumí výpočetní prostředí pro strojové učení, které se stará o provádění potřebných matematických operací a výpočtů. TensorFlow.js nabízí několik backendů, přičemž vždy je využíván právě jeden z nich. Ve výchozím nastavení je vybrán ten nejlepší, který je pro dané prostředí dostupný. Backend může být ovšem i uživatelsky změněn. Backendy jsou zde dvojího typu, a to podle toho, zda využívají pro výpočty CPU nebo GPU. [32]

CPU backend – Je nejméně výkonným a nejjednodušším backendem. Operace jsou implementovány v JavaScriptu, čímž je omezena možnost paralelního zpracovávání. Operace také běží v hlavním vlákně uživatelského rozhrání, což může způsobovat jeho blokování. Z výše zmíněného je patrné, že tato možnost není zcela ideálním řešením, proto je využívána spíše jako náhradní, pokud na zařízení není možnost využití efektivnějšího WebGL. [32]

WASM – Pod celým názvem WebAssembly je nabízen zajímavější backend opět využívající CPU zařízení. Byl představen v roce 2015 a od roku 2017 je podporován běžně používanými prohlížeči (Chrome, Firefox nebo Edge). Je mnohem rychlejší, než CPU backend díky tomu, že výpočty probíhají v předem zkompilovaném binárním formátu, který je efektivnější než interpretovaný JavaScript. Tím umožňuje téměř nativní výkon CPU v prohlížeči. WASM backend se hodí pro spuštění menších modelů, kde může dosahovat i lepších výsledků než backendy využívající GPU. [32]

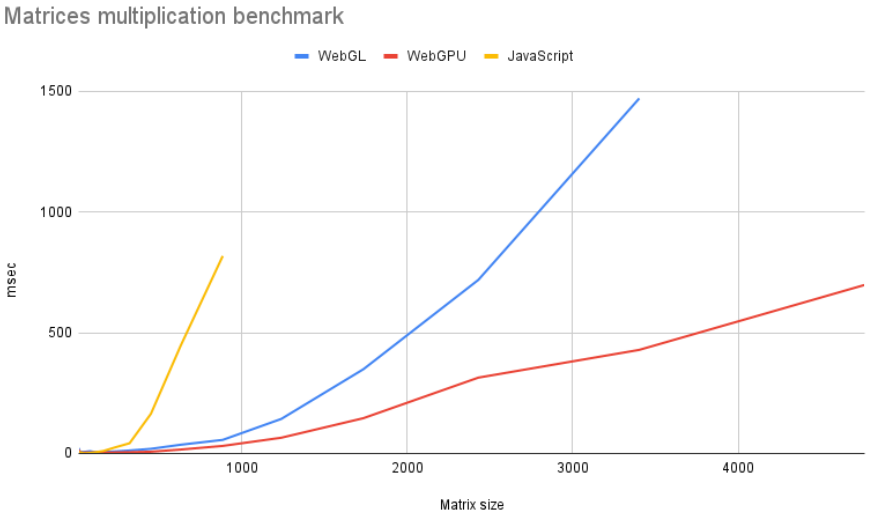
WebGL – Je výkonnějším backendem oproti předchozím možnostem. Výpočty jsou prováděny na GPU, kdy tensory jsou ukládány jako WebGL textury a matematické operace jsou prováděny pomocí WebGL shaderů (programů pro GPU). Dochází tak k převodu matematických operací strojového učení na grafické úlohy. Důležité je také zmínit, že kompilace shaderů probíhá na CPU, což přináší zpomalení. Zkompilované shadery jsou proto kešovány, aby při dalším volání stejné operace byl průchod rychlejší. [32]

Z tabulky níže je patrno, že při použití jednodušších modelů je výkon WASM a WebGL velmi podobný. Rozdíl nastává až při použití složitějších modelů, zde již jednoznačně dominuje WebGL.

Smaller models				
Model	**WebGL**	**WASM**	**CPU**	**Memory**
BlazeFace	22.5 ms	15.6 ms	315.2 ms	.4 MB
FaceMesh	19.3 ms	19.2 ms	335 ms	2.8 MB
Larger models				
Model	**WebGL**	**WASM**	**CPU**	**Memory**
PoseNet	42.5 ms	173.9 ms	1514.7 ms	4.5 MB
BodyPix	77 ms	188.4 ms	2683 ms	4.6 MB
MobileNet v2	37 ms	94 ms	923.6 ms	13 MB

Tabulka 1: Výkonnost backendů na daných modelech. Zdroj [32].

WebGPU – Ačkoliv tato možnost ještě není zmíněna na stránkách s oficiální dokumentací, v repozitáři TensorFlow.js na GitHubu je k dispozici implementace backendu WebGPU. Ta v současné chvíli podporuje jen několik modelů a není ještě ve finální podobě. WebGPU oproti WebGL není limitováno pouze na grafické úlohy a může tak lépe využívat potenciál GPU i pro výpočty strojového učení. Ačkoliv tento backend přináší nejlepší výkon z výše zmíněných, jeho limitace tkví v tom, že je poměrně nový a podporují ho jen některé prohlížeče, například populární Chrome ho začal podporovat až od verze 113 v květnu 2023. [33]



Obrázek 12: Porovnání výkonu WebGL a WebGPU pro násobení matic. Zdroj [34].

4.5 Případová studie

Knihovna TensorFlow.js má široké uplatnění v praxi, kdy ji využívají některé známé firmy. Níže je uvedeno několik konkrétních příkladů.

ModiFace

Je platforma pro kosmetický průmysl, která umožňuje virtuální líčení. Od roku 2008 spolupracuje se známou kosmetickou značkou L'Oreal. Na stránkách této značky si tak mohou zákazníci konkrétní kosmetické produkty virtuálně vyzkoušet na svém vlastním obličeji. ModiFace využívala svůj vlastní model, který byl ovšem poměrně velký a dlouho trvalo jeho načtení. Dalším problémem bylo řešení server-side, kdy docházelo ke zpoždění díky síťové latenci. Proto ModiFace začala využívat knihovnu Tensorlfow.js, kdy využila dostupného modelu, který byl o 40% menší a i přesnější, navíc mohla použít řešení client-side. [35]

InSpace

Platforma pro video konference, která používá TensorFlow.js pro detekci toxicity ve zprávách. Použití zdůvodňuje jednoduchostí integrace a také možnosti běhu modelu v prohlížeči, kdy tak uživatele na toxický text varuje ještě před odesláním a zpracováním serverem. [36]

Adobe

Společnost nabízí aplikaci Photoshop běžící ve webovém prostředí pomocí WebAssembly. Aplikace využívala řešení server-side, pro pokročilé funkce strojového učení jako například nástroj pro výběr objektů. Nevýhoda byla při spouštění těchto menších často využívaných modelů, kde docházelo k latenci při přenosu dat na server, což zhoršoval „User Experience“. Nasazením TensorFlow.js umožnilo využití grafického procesoru pomocí WebGL na klientském zařízení. To mělo za důsledek, že menší modely, kterým postačuje výkon zařízení uživatele, mohou běžet na straně uživatele. Zatímco velké, výpočetně náročné modely mohou být nadále spouštěny na straně serveru. [37]

Linkedln

Poměrně zajímavé využití knihovny TensorFlow.js přináší také profesní sociální síť Linkedln. Ta vytvořila model pro předpovídání výkonnosti zařízení uživatele a rychlosti sítě. Na základě této předpovědi je umožněno uživatelům s nepříznivým výsledkem předpovědi předkládat například obrázky v menším rozlišení, a tak pro ně zrychlit průchod prostředím, zatímco uživatelům s kladnou předpovědí poskytovat kvalitnější zobrazení. [38]

Google DermAssist

Je webová aplikace od společnosti Google, která pomáhá uživatelům s rozpoznáním problémů jejich pokožky. Aplikace knihovnu používá pro detekci kvality pořízené fotografie a zlepšení tak výsledků analýzy. Uživateli je tak předložena informace o špatné kvalitě obrazu ještě před jeho odesláním ke zpracování serverem. [39]

4.6 Možnosti knihovny pro zpracování obrazu

Knihovna nabízí několik předpřipravených modelů, které jsou navrženy pro konkrétní úlohy. Výhodou těchto modelů je snadnost jejich použití, modely jsou již předtrénovány. Lze je tak ihned využít pro vytváření predikcí. Knihovna k nim nabízí navíc poměrně hezky zpracovanou dokumentaci, která sice není nikterak obsáhlá, ale je přehledná a poskytuje všechny potřebné informace pro nasazení modelů. Kromě modelů zaměřených na zpracování obrazu knihovna také nabízí modely zaměřené na text a zvuk. Textový model tak nabízí například možnost vyhodnocení toxicity textu. Všechny níže zmíněné úlohy, respektive modely, které je řeší jsou založeny na konvolučních neuronových sítích. Modely pro detekci obličeje a detekce klíčových bodů obličeje budou využity v praktické části, proto zde budou rozebrány podrobněji.

Detekce obličeje

Knihovna TensorFlow.js zde pro tyto účely využívá modelu BlazeFace z knihovny MediaPipe. Model dovede ve snímku detekovat vícero obličejů, pomocí souřadnic tzv. „bounding boxu“. Kromě toho umí také přibližně určit souřadnice šesti klíčových bodů obličeje – oči, špička nosu, pusa a uši. Dalším výstupem je pak míra jistoty předloženého výsledku. Výhodou modelu je jeho malá velikost – pouze 224KB. Díky tomu je model velmi rychlý, kdy je deklarován jako „super-real-time“. Limitace modelu tkví v jeho náchylnosti na polohu, velikost a orientaci obličeje v obraze. Vstupem modelu jsou barevné snímky o rozměru 128x128 pixelů. Model je natrénován na datasetu obsahujícím snímky s různorodými světelnými i šumovými podmínkami. Snímky datasetu jsou také různorodé v počtu lidí ve snímku, jejich barvě pleti, pohlaví a geografické příslušnosti. Celková úspěšnost BlazeFace je velmi vysoká — model dosahuje skvělé přesnosti (95-100 %) napříč různými skupinami snímků (pohlaví, odstíny pleti, geografickými oblastmi), což znamená, že spolehlivě detekuje většinu obličejů a zároveň má velmi nízkou chybovost při jejich označování. [40]

Detekce klíčových bodů obličeje

Zde je opět využito modelu z knihovny MediaPipe a to Face Mesh. Tento model je založený na topologii MobileNetV2, jež je ještě mírně upravena pro real-time výkonnost. Velikost modelu je přibližně 1,2 MB. Model ve snímku dovede detekovat celkem 478 bodů na obličeji, a to v 3D souřadnicích. Některé body obsahují také příslušnost k obličejové části – oči, pusa, obočí. Na vstupu model přijímá barevné snímky o rozměru 192x192 pixelů. Struktura modelu je rozdělena na několik částí. Nejprve je použit detektor obličeje. Následně je z obličeje provedena extrakce klíčových vlastností. V poslední části je model dělen na submodely, kdy jeden model provede predikci všech bodů obličeje, zatímco další submodely jsou zaměřeny na detailnější

predikci bodů pro klíčové části obličeje. Stejně jako předchozí model i zde je trénink proveden na kvalitním a různorodém datasetu. Přesnost modelu je na dobré úrovni. Průměrná chyba určení polohy bodů je měřena relativně k vzdálenosti středu očí a dosahuje hodnoty 3 %. [41]

Klasifikace obrázků

Knihovna pro tyto účely poskytuje model založený na topologii MobileNet. Je to malý model zaměřený na klasifikaci obrázků. Výstupem je množina objektů, které byly v obrázku detekovány i s jakou pravděpodobností byly detekovány. Model nedosahuje oproti větším modelům takové přesnosti. Jeho výhoda naopak spočívá v nízké latenci. [42]

Detekce objektů

Pro tyto účely je dostupný model Coco SSD. Oproti předchozímu modelu neklasifikuje obrázek jako celek, ale provede lokalizaci objektů v obrázku a jejich klasifikaci. Je založen na datasetu COCO a je schopný detekovat 80 objektů v obrázku, mezi které paří například několik druhů dopravních prostředků, zvířat, jídla nebo elektroniky. Pomocí počáteční konfigurace je zde možnost volby různých topologií (MobileNetV1, MobileNetV2 nebo Lite MobileNetV2). [43]

Sémantická segmentace

Dostupný je opět jeden model a to DeepLab. Model je opět založen na topologii MobileNet, konkrétněji jeho druhé verzi (MobileNetV2). Jeho výstupem je segmentační mapa, kde pixelům obrázku je přiřazen popisek se skupinovou příslušností (člověk, budova apod.). [44]

Sledování polohy těla

Pro tyto účely jsou poskytovány 3 modely (MoveNet, BlazePose, PoseNet). Na výstupu jsou dostupné souřadnice klíčových bodů těla včetně skóre důvěryhodnosti detekovaných bodů. Dále segmentační maska, pokud ji daný model podporuje. Ta určuje, které pixely obrázku patří osobě a které pozadí. Jednotlivé modely se liší v počtu detekovaných bodů, možnosti detekce na více osobách, souřadnicovém systému bodů nebo tím, zda umožňují vytvářet segmentační masku. Topologie modelů je specifikovaná pouze u modelu PoseNet, který může využívat buď MobileNetV1 nebo ResNet50. [45]

Segmentace lidského těla

Zde jsou modely zaměřeny na oddělení člověka od pozadí. Knihovna poskytuje 2 modely. MediaPipe SelfieSegmentation je rychlý model schopný poskytovat výsledky v reálném čase, a to i na mobilních zařízeních. Název napovídá, že model je určen primárně na práci s pozadím, kdy osoba je v těsné blízkosti kamery. Výstupem je segmentovaná část obrazu, která obsahuje všechny detekované osoby dohromady. Model BodyPix nabízí navíc možnost segmentovat

každou osobu v obraze zvlášť. Dále umožňuje segmentovat detekované osoby na 24 částí těla. Výstupem obou modelů je segmentační maska. Ta pro každý bod obrazu určuje pravděpodobnost, zda je bod součástí osoby v obraze. [46]

Detekce polohy dlaně

V knihovně je pro tyto účely dostupný model MediaPipe Hands. Umožňuje detekovat 21 bodů na dlani, a to i pro více předložených dlaní v obraze. Dále umí detekovat, jestli se jedná o pravou nebo levou ruku a pravděpodobnost správnosti předloženého výsledku. Model je opět velmi rychlý s dostupností výpočtu v reálném čase. Topologie zde není blíže specifikována. [47]

Odhad hloubky v portrétu

Model AR Portrait Depth slouží k odhadu hloubky v portrétu, tedy odhaduje pro jednotlivé pixely vzdálenost od kamery. Nabízí tak například možnost převodu portrétu z 2D do 3D podoby. [48]

5 PRŮZKUM ALTERNATIVNÍCH KNIHOVEN

V praktické části práce bude využito výše představené knihovny TensorFlow.js především kvůli tomu, že umožňuje zpracování obrazu přímo v prohlížeči uživatele. Díky tomu lze detekci provádět v reálném čase bez nutnosti odesílat data na server, což přináší výhody z hlediska absence latence přenosu dat a ochrany soukromí. TensorFlow.js navíc obsahuje předtrénované modely pro práci s obličejem, které jsou optimalizované pro prohlížečové prostředí a které lze snadno propojit s obrazovými vstupy z kamery. To vše výrazně zjednodušuje vývoj, nasazení a použitelnost výsledné aplikace. Knihovna navíc ve srovnání s ostatními alternativami nijak zásadně nezaostává ani ve výkonnosti. Pro úplnost budou v této části představené alternativní knihovny k Tensorflow.js, které byly zvažovány při výběru vhodného řešení.

5.1 Brain.js

Knihovna vyniká oproti ostatním svojí jednoduchostí. Pro její použití není nutné mít hlubší znalosti z oblasti strojového učení. Knihovna nabízí možnost vytváření jak client-side aplikací, tak i server-side aplikací. Pro tvorbu vlastních modelů je zde výběr běžných neuronových sítí, kdy ale nejsou podporovány konvoluční neuronové sítě. Knihovna je rychlá. Pro urychlení výpočtů umí používat grafický procesor zařízení. To je zde umožněno pomocí knihovny GPU.js. Ta provádí převod funkcí JavaScriptu do shaderů, které po kompilaci mohou být spouštěny na GPU pomocí WebGL. [49]

Svými vlastnostmi se hodí spíše pro začátečníky a pro tvorbu jednodušších modelů, například pro klasifikaci textu. Naopak není příliš vhodná pro zpracování obrázků díky absenci konvolučních sítí.

5.2 ConvNetJS

Knihovna vyvinuta studentem Stanfordu Andrejem Karpathym. Knihovna je určená pro běh v prohlížeči. Stejně jako předešlá knihovna není určena pro tvorbu rozsáhlejších projektů, ale je spíše vhodná pro prototypování a testování modelů. Výhodou je opět jednoduchost. Jak z názvu vyplývá, umožňuje použití i konvolučních neuronových sítí, což by bylo vhodné pro práci s obrázky. Ovšem mezi nevýhody určitě patří nemožnost využívat grafického procesoru, a tak i nižší výkonost. Dále pak fakt, že je poměrně zastaralá, kdy poslední verze knihovny byla vydána v roce 2014. [50]

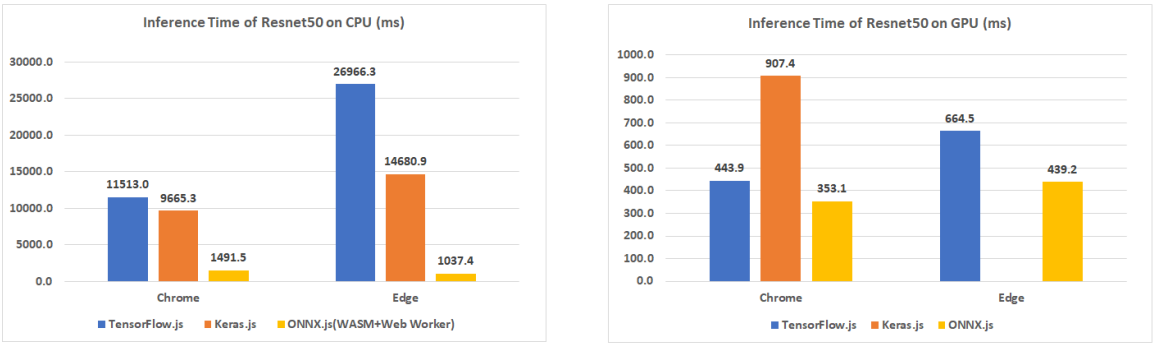
5.3 ml5.js

V tomto případě se nejedná přímo o knihovnu, ale API pro strojové učení. To přináší velké zjednodušení při vývoji. API je postaveno nad knihovnou TensorFlow.js. Nabízí několik předtrénovaných modelů, například pro klasifikaci obrázků, zvuku nebo také pro práci s lidským tělem. Dále lze vytvořit a natrénovat vlastní neuronovou sít zaměřenou na úkoly, jako je klasifikace dat, nebo regrese. Dostupná je také velmi intuitivní dokumentace. [51]

Toto řešení je vhodné, jak i samotná knihovna deklaruje, pro studenty nebo uživatele méně zkušené se strojovým učením, kteří ocení odstínění od detailů implementace. To ovšem přináší menší flexibilitu a množství pokročilejších možností.

5.4 ONNX.js

Poměrně zajímavé řešení pro tyto účely nabízí také společnost Microsoft s názvem Open Neural Network Exchange. Tato knihovna umožňuje běh modelů ve formátu ONNX, a to jak v prohlížeči, tak i na serveru Node.js. Jistým omezením je, že knihovna je určená pouze pro spouštění modelů. Samotné vytváření a trénování modelu musí být tedy provedeno v jiné knihovně například TensorFlow nebo PyTorch a následně vyexportováno do požadovaného formátu. Silnou stránkou této knihovny je její výkon. Umí využívat, jak CPU pomocí WebAssembly, tak i GPU pomocí WebGL. Pro CPU využívá technologii Web Workers, která umožňuje paralelizaci výpočtu mezi více vlákny CPU. Knihovna také přináší optimalizaci v přenosu dat mezi CPU a GPU, čímž snižuje počet přenášených dat. Je zde dostupné také takzvané Model Zoo, které nabízí poměrně rozsáhlý výběr předtrénovaných modelů. [52]



Obrázek 13: Porovnání výkonnosti knihoven. Zdroj [52].

6 NÁVRH APLIKACE

6.1 Požadavky

První fází vývoje je specifikování a analyzování požadavků na aplikaci. Cílem je zjistit, co přesně má systém dělat a jaké jsou potřeby uživatelů. Funkční požadavky specifikují, jaké konkrétní funkce by měl systém poskytovat, zatímco nefunkční požadavky říkají, jakým způsobem by měl systém fungovat.

6.1.1 Funkční

ID	DEFINICE
FR1	Aplikace umožní na vstupu zpracovávat obrázek nahraný z uložiště
FR2	Aplikace umožní zpracovat na vstupu video z kamery zařízení
FR3	Aplikace umožní detekovat obličeje ve vstupních datech
FR4	Aplikace umožní na detekovaných obličejích rozpoznat klíčové části (oči, nos, pusu, obočí)
FR5	Aplikace umožní určit náladu detekovaného obličeje (rozzlobený, znechucený, vystrašený, šťastný, neutrální, smutný a překvapený)
FR6	Aplikace umožní odhadnutí věku detekovaného obličeje
FR7	Aplikace umožní rozpoznání pohlaví detekovaného obličeje
FR8	Aplikace vizuálně zobrazí výstup analýzy obličeje

Tabulka 2: Funkční požadavky. Zdroj vlastní.

6.1.2 Nefunkční

ID	DEFINICE
NR1	Aplikace využívá pro strojové učení knihovnu TensorFlow.js
NR2	Aplikace bude dostupná na webové platformě
NR3	Aplikace bude využívat jazyk JavaScript
NR4	Aplikace nabídne uživatelsky přívětivé prostředí
NR5	Aplikace bude responzivní, čímž umožní použitelnost i na mobilních zařízeních

Tabulka 3: Nefunkční požadavky. Zdroj vlastní.

6.2 Diagram případů užití

Jedná se o jeden z nepoužívanějších typů diagramu v UML (Unified Modeling Language). Používá se v rané části vývoje, kde slouží ke grafickému znázornění interakcí aktérů se systémem. Diagram funguje na principu tzv. blackboxu, kdy znázorňuje, jaké funkcionality bude systém nabízet, bez ohledu na to, jakým způsobem budou implementovány. [53]

Případ užití

Slouží pro definici funkcionality, kterou bude systém nabízet. V diagramu se zpravidla znázorňuje elipsou s názvem případu užití uvnitř. [53]

Aktér

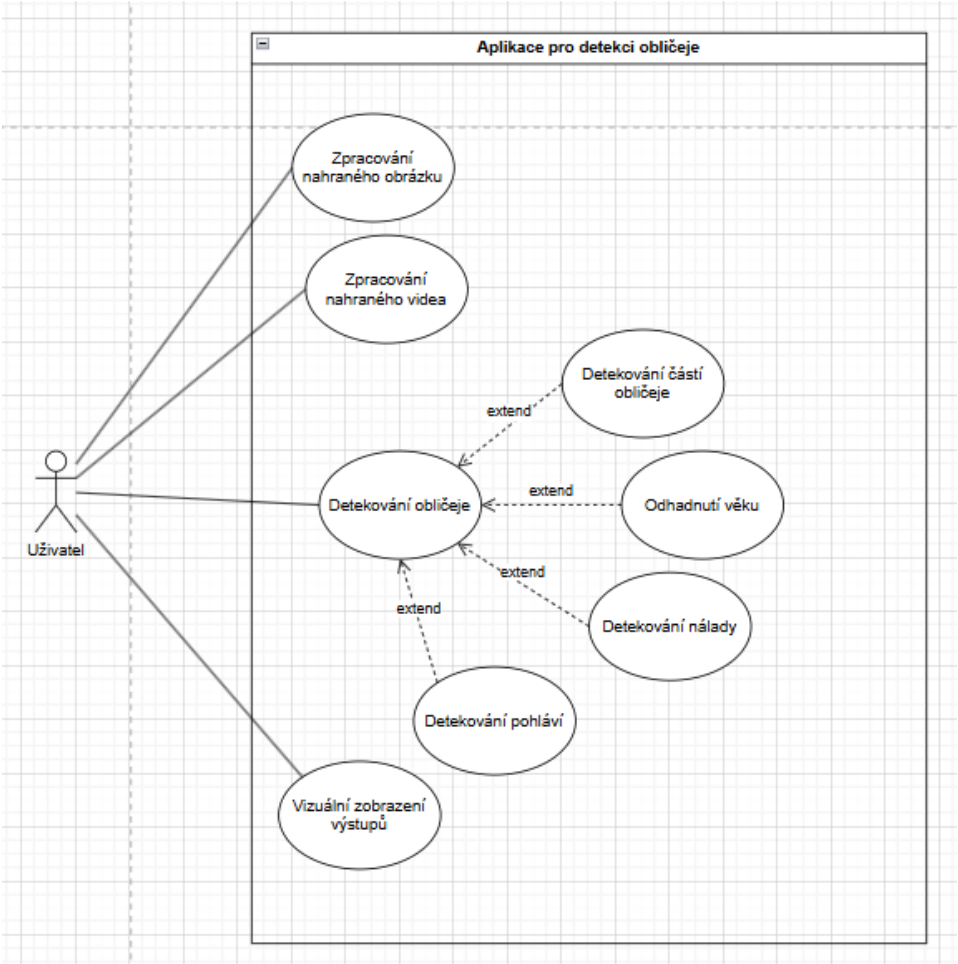
Je role, která inicializuje nebo je inicializována případem užití. Aktérem může být jak nějaká osoba (běžný uživatel, administrátor), tak i jiný systém. V diagramu je aktér znázorněn panáčkem s názvem. [53]

Hranice systému

Odděluje systém a jeho interní případy užití od aktérů, kteří jsou pro systém externí. Tento prvek digramu je volitelný a znázorňuje se obdélníkem. [53]

Vztahy

Jsou znázorněny čárami nebo směrově šipkami. V diagramu je znázorněna asociace čárou mezi aktérem a případem užití a říká, že daný případ užití může být tímto aktérem prováděn. Vazba exclude mezi případy užití značí, že případ užití, od kterého vede šipka, může volitelně rozšiřovat případ užití, ke kterému šipka směřuje. [54]



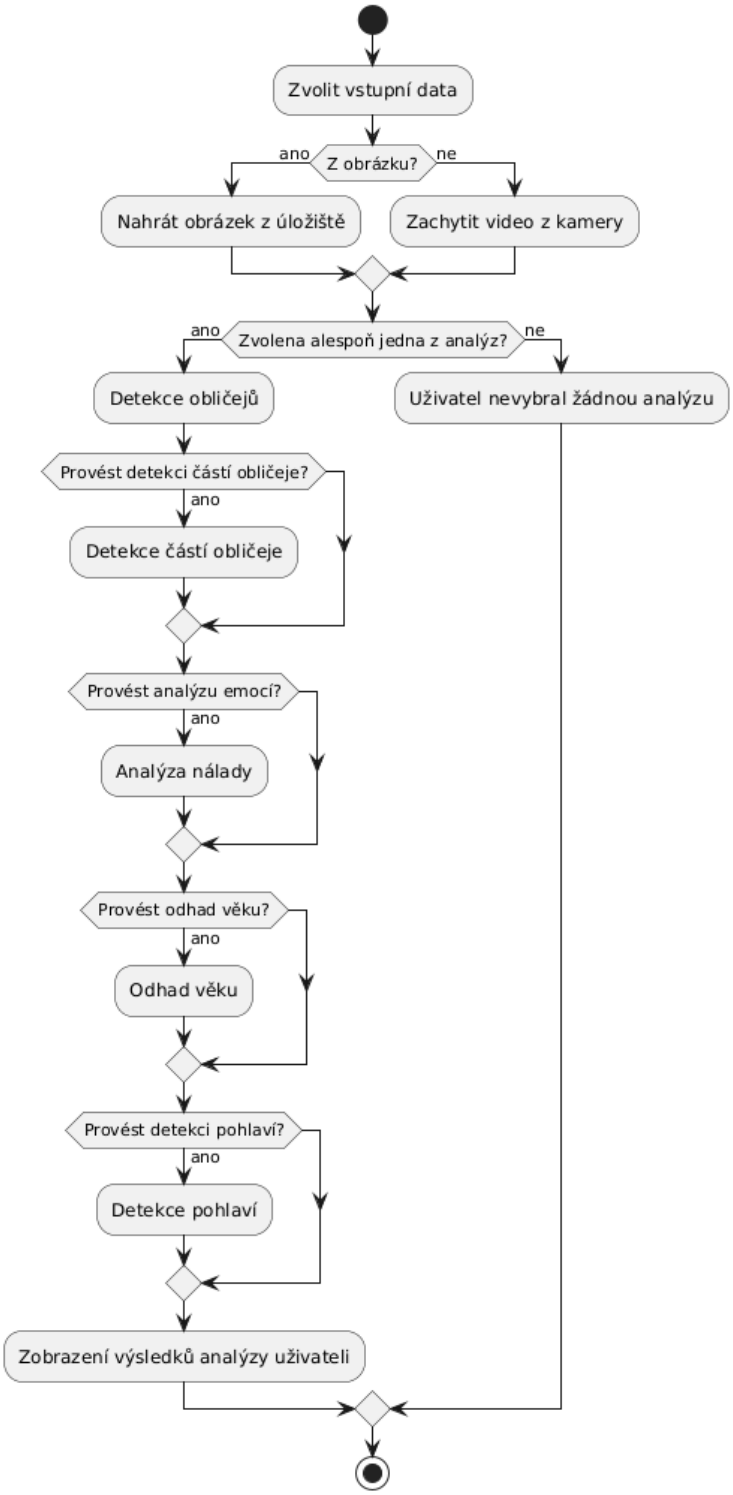
Obrázek 14: Use Case diagram. Zdroj vlastní.

6.3 Diagram aktivit

Dalším z často využívaných diagramů nejen pro modelování chování systému, ale například i pro modelování business procesů, je diagram aktivit. Ten vizuálně zobrazuje postupnou návaznost jednotlivých činností, jejich větvení a paralelní průběh. Hodí se jak na začátku vývoje pro plánování hlavních pracovních toků, kdy umožňuje lepší pochopení systému, tak i později pro upřesnění chování systému. [55]

Diagram níže se skládá z

- počátečního uzlu (černý plný kruh)
- aktivit (obdélník se zaoblenými rohy)
- toku, neboli hran (šipka)
- rozhodovacími uzly (kosočtverec)
- koncovým uzlem (terčík s černým středem). [55]

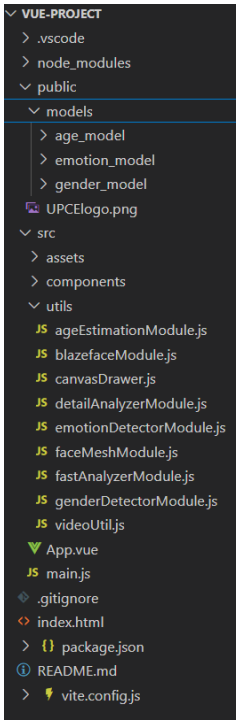


Obrázek 15: Diagram aktivit. Zdroj vlastní.

7 IMPLEMENTACE

7.1 Struktura projektu aplikace

Strukturu zachycuje obrázek níže. Zdrojové kódy jsou rozděleny do dvou adresářů, kde adresář *components* obsahuje *vue* komponenty sloužící jako stavební bloky uživatelského rozhraní. Adresář *utils* obsahuje soubory s pomocnými zdrojovými kódy pro obsluhu jednotlivých modelů, zpracovávání video vstupu z kamery a vykreslování na canvas.



Obrázek 16: Struktura projektu aplikace. Zdroj vlastní.

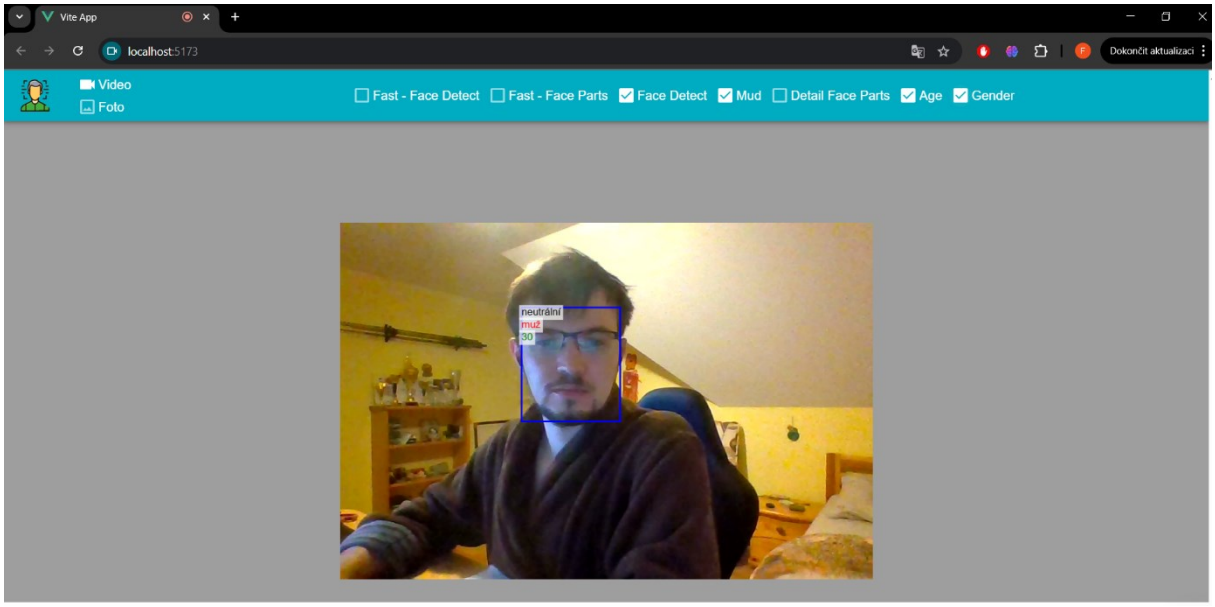
7.2 Uživatelské prostředí

Aplikace běží čistě na straně uživatele a je tak tvořena pouze frontendovou částí. Uživatelské rozhraní je implementováno jako SPA aplikace pomocí open-source framevorku Vue. V rámci tohoto frameworku je využíván komponentový přístup, kdy komponenta je rozdělena na 3 části.

- Template – HTML kód, který definuje strukturu komponenty
- Script – Implementace logiky komponenty
- Style – Definice vzhledu komponenty pomocí CSS stylů

Díky využití frameworku Vuetify je možné používat předdefinované komponenty uživatelského rozhraní a jednoduše tak vytvářet přehledné a stylově sjednocené uživatelské prostředí na základě Material Design od Google.

Uživatelské prostředí je uživatelsky přívětivé a intuitivní. Uživatel zde má v levém horním rohu možnost zvolit typ vstupu (video vstup z kamery zařízení, nebo nahrát obrázek z úložiště). V prostřední části se poté nachází výběr možných analýz z předložených obrazových dat.



Obrázek 17: Ukázka uživatelského prostředí. Zdroj vlastní.

7.3 Strojové učení

Pro detekci obličeje a jeho částí byly využity předtrénované modely určené přímo pro tyto účely, které jsou dostupné v knihovně TensorFlow.js. Tyto modely jsou představeny v kapitole 4.6.

Při potřebě modelu pro specifický účel, pro který knihovna řešení nenabízí, je nutné si obstarat model vlastní a importovat ho do aplikace. Zde se hodí také zmínit velké množství dostupných předtrénovaných modelů, jež je možné získat na webu Kaggle. Zde jsou dostupné nejen hotové modely, ale také především velké množství datasetů pro tvorbu vlastních modelů.

Funkce aplikace	**Použitý model**
Fast – Face Detect	MediaPipe FaceDetection
Fast – Face Parts	MediaPipe FaceDetection
Face Detect	MediaPipe FaceMesh
Mud	Vlastní model na základě vlastní topologie
Detail Face Parts	MediaPipe FaceMesh
Age	Vlastní model na základě upravené topologie MobileNet
Gender	Model vytvořený nástrojem Teachable Machine

Tabulka 4: Použité modely. Zdroj vlastní.

7.3.1 Model pro detekci emocí

Model pro rozpoznání emocí řeší klasifikační problém, a to udělit obličeji na předloženém obrázku příslušnost jedné ze sedmi emocí. Vytvořen je v jazyce Python pomocí API Keras, které je součástí knihovny TensorFlow.

Trénování modelu proběhlo na veřejně dostupném datasetu obličejů získaném z Kaggle pod názvem *Facial Emotion Expressions (FER) Balanced*. Dataset je již rozdělen na trénovací a testovací část. Obsahuje obrázky o rozměru 48x48 pixelů v odstínech šedi, které jsou rozděleny do 7 podadresářů, dle kategorií nálad (rozzlobený, znechucený, vystrašený, šťastný, smutný, překvapený a neutrální). Trénovací část datasetu obsahuje celkem přibližně 29 tisíc obrázků, testovací část pak přibližně 7 tisíc obrázků. Pro testování je tedy vyčleněno přibližně 20 % z celkového počtu obrázků. Oproti běžně využívanému datasetu *FER2013* Dataset poskytuje vyváženost klasifikačních tříd a výsledný model by tak neměl zvýhodňovat některé třídy. [56]

Model je navržen jako konvoluční neuronová síť. Význam jednotlivých vrstev je popsán v kapitole výše o konvolučních neuronových sítích. Pro řešení této úlohy byla využita vlastní topologie. V rámci experimentování byly vyzkoušeny různé změny v topologii. Většina těchto změn ovšem nepřispěla k výrazně vyšší přesnosti modelu.

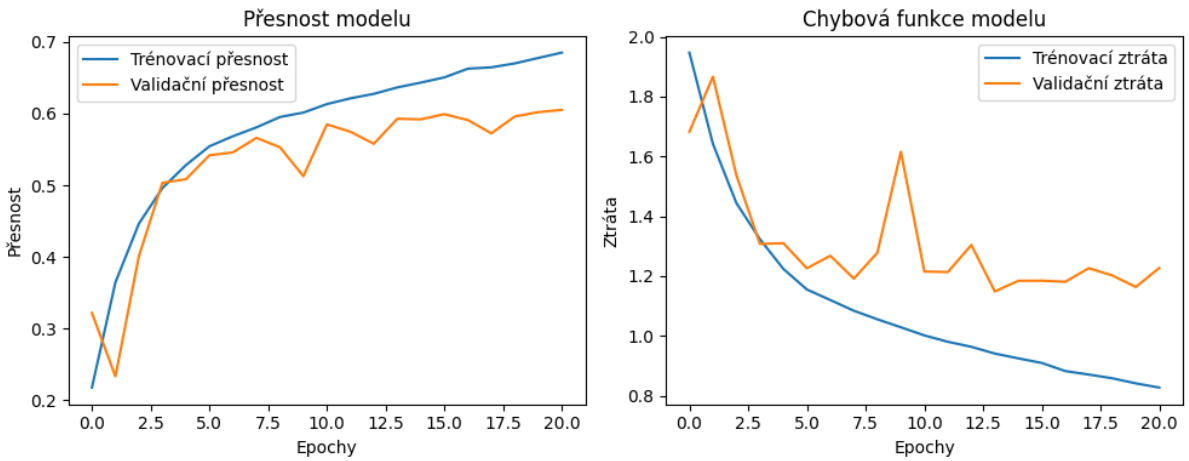
Výsledný model tak má topologii, jež je zachycena v následujícím kódu.

```
# Model Topology
model = Sequential()
# 1. Layer
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 1)))
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
# 2. Layer
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))
# 3. Layer
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
```

Zdrojový kód 1: Topologie modelu pro detekci emocí. Zdroj vlastní.

Trénink modelu se zastavil po 21 epochách díky použití metody „early stop“. Nejlepších výsledků model dosáhl po 14 epoše. Výsledná přesnost tak je přibližně 64 % na trénovacích datech a cca 59 % na validačních datech. V následujících epochách je v záznamu průběhu trénování modelu vidět, že již dochází spíše k přeučování modelu. Ačkoliv chybovost na trénovacích datech klesá, chybovost na validačních datech spíše stagnuje.



Obrázek 18: Průběh trénování modelu pro detekci emocí. Zdroj vlastní.

Na první pohled působí hodnota přesnosti poněkud nízká. Je ale nutné zmínit, že i pro člověka samotného je určení emoce pouze na základě výrazu obličeje často nelehké a subjektivní. I odborné články uvádějí přesnost člověka na datasetu *FER2013* přibližně 65 % [57]. Je také vhodné zmínit, že při náhodném výběru emoce by přesnost dosahovala přibližně pouze 14 %. Dalším aspektem, ke kterému je nutno přihlédnout je použití modelu ve webovém prostředí, kde je kladen požadavek na rychlost zpracování. To neumožňuje využít některou ze známých, přesnějších topologii – například přesnost *VGGNet* je přes 70 % [57], ale výpočetní náročnost by byla mnohem větší. Z výše zmíněných důvodů je přesnost modelu považována za dostatečnou.

Model je možné následně přetransformovat z formátu H5 do modelu, který je kompatibilní s TensorFlow.js. Pro tyto účely se využívá TensorFlow.js converter, jehož výstupem je topologie modelu ve formátu JSON a soubory s váhami modelu. To je možné následujícím příkazem. [58]

```
tensorflowjs_converter --input_format=keras /tmp/model.h5 /tmp/tfjs_model
```

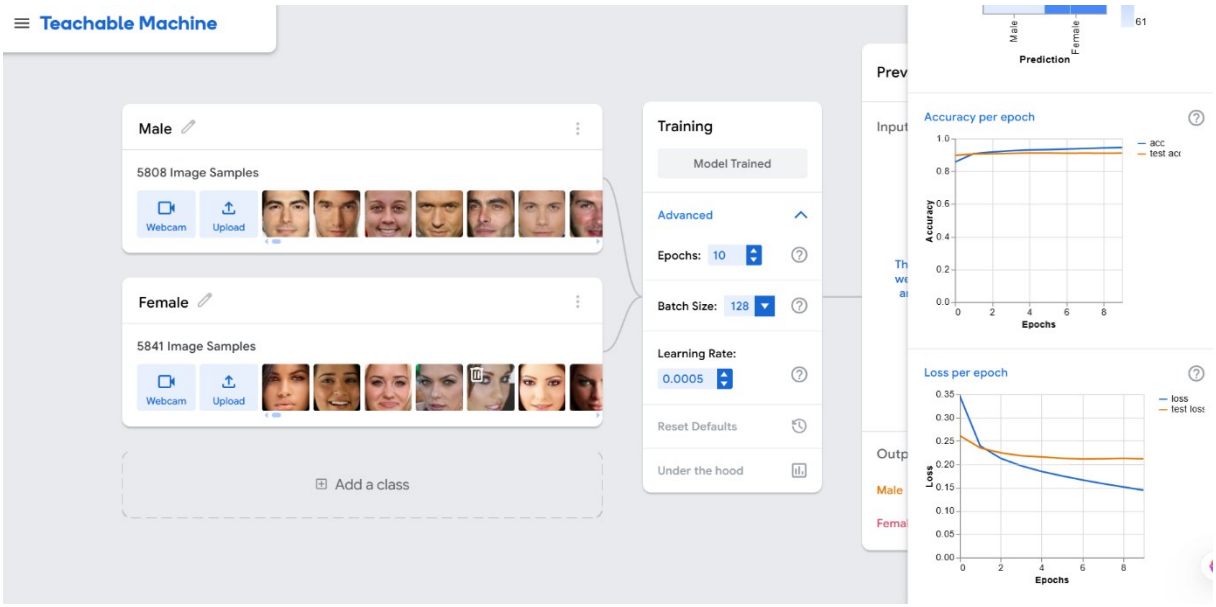
Zdrojový kód 2: Příkaz pro převod modelu. Zdroj [58].

7.3.2 Model pro detekci pohlaví

Tento model řeší klasifikační problém, jehož cílem je rozhodnout, zda obličej na obrázku je muž nebo žena. Při tvorbě tohoto modelu bylo využito nástroje Google Teachable Machine.

Nástroj nabízí možnost trénování vlastních modelů strojového učení, a to bez psaní kódu. Ovládání probíhá pomocí jednoduchého GUI. Podporovány jsou obrazová a audio data. Tvorba samotného modelu je zde velmi jednoduchá. Nejprve je nutné vybrat, že model bude určen pro obrazová data. Následně definovat příslušné třídy, které budou v obrázkách detekovány a

nahrát do těchto tříd obrázky pro trénování. Následně je možné nastavit parametry trénování – počet epoch, velikost dávky a rychlost učení. V průběhu trénování je možné sledovat graf ztrátové funkce, přesnost, matici záměn a přesnost pro dané třídy. Po dokončení trénování je možnost model exportovat a stáhnout, kde mezi možnými formáty je i TensorFlow.js. Nástroj vytváří modely na základě lehké topologie *MobileNet*. [59]



Obrázek 19: Ukázka nástroje Google Teachable Machine. Zdroj vlastní.

Model je natrénován na datasetu dostupném na Kaggle pod názvem *Gender Classification Dataset*. Dataset obsahuje dva adresáře – s trénovacími a validačními daty. Ty obsahují podadresáře s obrázky mužských a ženských obličejů. Trénovací množina obrázků čítá cca 23 000 obrázků pro každé pohlaví. Validační množina má cca 6000 obrázků pro každé pohlaví. [60]

Dataset poskytuje různorodé obrázky, a to z pohledu různého:

- Úhlu obličejů
- Nasvětlení a pozadí
- Emoce
- Barvy pleti
- Věku
- Účesu

Jelikož Google Teachable Machine poskytuje omezené výkonové možnosti, model byl natrénován pouze pomocí validační množiny tohoto datasetu, kterou si nástroj následně rozdělil

na novou trénovací a validační množinu. Při vkládání rozsáhlého trénovacího datasetu již docházelo k zamrzání aplikace.

Ačkoliv nástroj nenabízí příliš robustní řešení a moc pokročilých možností tvorby a trénování modelů, pro tuto jednodušší úlohu dosáhl vcelku uspokojivého výsledku. Následující tabulka zachycuje přesnost pro jednotlivé třídy na testovacích datech.

CLASS	ACCURACY	SAMPLES (test)
Male	0,93	872
Female	0,89	877

Tabulka 5: Výsledná přesnost modelu pro detekci pohlaví. Zdroj vlastní.

7.3.3 Model pro odhad věku

Tento model řeší klasifikační problém, jehož cílem je odhadnout věk obličeje na obrázku.

Základem tohoto modelu je dataset *UTKFace* dostupný na Kaggle. Dataset obsahuje obrázky obličejů lidí ve věkovém rozpětí od 0 do 116 let. Informaci o věku osoby uvádí první číslo v názvu každého obrázku. Celkem má dataset cca 23700 obrázků. Obrázky v datasetu jsou opět různorodé z hlediska výrazu, kvality, osvětlení a úhlu pohledu osoby. Snímky mají jednotné rozlišení 200x200 pixelů. [61]

Model je postaven na známé topologii *MobileNetV2* vyvinuté společností Google. Tato síť je navržena pro efektivní výpočty na zařízeních s omezeným výpočetním výkonem. Její výhodou je především nízký počet parametrů, čímž přináší snížení výpočetní náročnosti. [62]

Pro vytvoření a natrénování modelu bylo opět využito jazyku Python a API Keras. V kódu níže je vidět načtení základního modelu, který zde slouží jako extraktor rysů z obrázku. Vstupní data jsou ve formátu 224x224 pixelů s 3 barevnými kanály, což odpovídá standardnímu vstupu modelů předtrénovaných na datasetu *ImageNet*. Původní klasifikační vrstva je odstraněna (*include_top=False*), aby bylo možné model přizpůsobit regresnímu úkolu (odhad věku). Výstup z *MobileNetV2* je dále zpracován vrstvou *GlobalAveragePooling2D*, která zredukuje prostorové rozměry výstupu. Následuje plně propojená (*Dense*) vrstva se 128 neurony a aktivační funkce ReLU, která umožňuje modelu zachytit nelineární vztahy. Poslední výstupní vrstva obsahuje jediný neuron bez aktivační funkce, čímž poskytuje spojitou hodnotu – v tomto případě odhad věku. Model je tak sestaven jako spojení vstupní vrstvy základního modelu s přednastavenými váhami z datasetu *ImageNet* a nově přidané výstupní části pro regresní úlohu. [62]

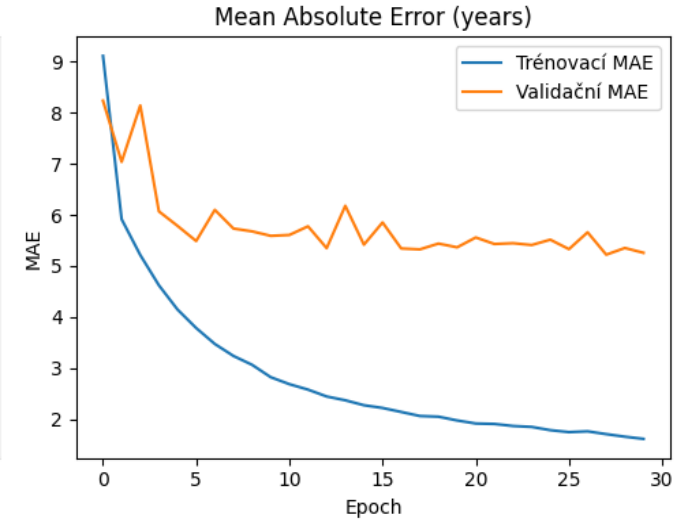
```
# Model creation
base_model = MobileNetV2(
     input_shape=(224, 224, 3),
     include_top=False,
     weights="imagenet")
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dense(1)(x)  # výstup: odhad věku
model = Model(inputs=base_model.input, outputs=x)
```

Trénink modelu se zastavil po 30 epochách. Nejlepších výsledků model dosáhl po 28. epoše. Pro měření přesnosti modelu je využito metriky MAE (Mean Absolute Error), která udává průměrnou absolutní odchylku mezi skutečnými a predikovanými hodnotami. MAE je také použito pro chybovou funkci.

Trénováním bylo dosaženo:

MAE na trénovacích datech: 1,61 let

MAE na validačních datech: 5,26 let



Obrázek 20: Průběh trénování modelu pro odhad věku. Zdroj vlastní.

7.3.4 Nasazení modelů v aplikaci

Následující úryvek kódu z modulu *emotionDetectorModule* zachycuje načtení modelu pro detekci emocí v aplikaci ve funkci *init*. Zde je také obsaženo tzv. „zahřátí“ modelu, které zajistí, že model další predikci zpracuje rychleji. Metoda *detectEmotion* již přijímá v parametru obrázek – reference na DOM element typu obrázek, který přizpůsobí na požadovaný vstup modelu. Model vytvoří predikci, kde výstupem je pravděpodobnost pro jednotlivé emoce, respektive indexu emoce. Zde je vybrán index s nejvyšší pravděpodobností a převeden na

textovou hodnotu emoce. Za zmínku také stojí funkce *tidy* a *dispose*, které slouží pro uvolnění paměti při alokování tensoru.

Použití modelů pro odhad věků a detekci pohlaví je velmi podobné, proto zde nebude představeno.

```
async function init() {
    model = await tf.
        loadLayersModel('/models/emotion_model/model.json');
    // Zahřátí modelu - následná predikce bude rychlejší
    const dummyInput = tf.zeros([1, 48, 48, 1]);
    model.predict(dummyInput).dispose();
    dummyInput.dispose();
}

async function detectEmotion(img) {
    const emotion = tf.tidy(() => {
        const image = tf.browser.fromPixels(img)
            .resizeNearestNeighbor([48, 48])
            .mean(2)
            .expandDims(0)
            .expandDims(-1)
            .toFloat()
            .div(tf.scalar(255));

        const prediction = model.predict(image);
        const emotionIndex = prediction.argMax(-1).dataSync()[0];
        const emotions = ['rozzlobený', 'znechucený', 'vystrašený',
            'šťastný', 'neutrální', 'smutný', 'surprise'];
        return emotions[emotionIndex];
    });
    return emotion;
}
```

Zdrojový kód 3: Inicializace a vytváření predikce vlastním modelem pro detekci emocí obličeje. Zdroj vlastní.

Použití knihovních modelů je podobné. Zde bude představeno použití pouze modelu *MediaPipe Facemesh*, který je použit v modulu aplikace pod názvem *faceMeshModule*. V aplikaci složí pro detailnější detekci obličejů a jejich klíčových částí. Detekované obličeje jsou následně v modulu *detailAnalyzerModule* převáděny na samostatné obrázky, které slouží jako vstup pro modely detekující věk, pohlaví a emoce. Použití ostatních knihovních modelů je velmi podobné. Opět je nejdříve nutné provést počáteční inicializaci modelu. Poté je možné provádět predikce (funkce *detectFaces*). Zde již není nutné provádět úpravy obrázku, jako tomu bylo při použití vlastního modelu. Díky tomu, že stačí předat pouze referenci DOM element obrázku,

není nutné ručně vytvářet tensory a následně se starat o uvolňování paměti, jako tomu bylo v předchozím případě.

```
async function init() {
    const detectorConfig = {
        runtime: 'tfjs',
        maxFaces: 5
    }
    detector = await faceLandmarksDetection.
    createDetector(faceLandmarksDetection.
        SupportedModels.MediaPipeFaceMesh, detectorConfig);
}

async function detectFaces(image){
    return await detector.estimateFaces(image);
}
```

Zdrojový kód 4: Inicializace a vytváření predikce knihovním modelem Mediapipe Facemash. Zdroj vlastní.

Následující kód ukazuje vybrané funkce modulu *detailAnalyzerModule*. Ve funkci *markFaceAge* je pro každý obličej provedena pomocí funkce *getImagePartWithFace* jeho separace do samostatného obrazku, respektive canvasu, který je předán modulu pro detekci pohlaví – *ageEstimationModule*. Vykreslování na canvas obstarává modul s názvem *canvasDrawer*. Tento modul slouží především pro vykreslování jednotlivých tvarů, či textu na konkrétní pozici na výstupní canvas předkládající výsledky analýz uživateli.

```
async function markFaceAge() {
    faces.forEach(async face => {
        const canvasWithFace = getImagePartWithFace(face);
        const age = await ageEstimationModule.estimateAge(canvasWithFace);
        canvasDrawer.drawText(face.box.xMin, face.box.yMin,
             age, "green", face.box.width, 2);
    })
}

// Funkce pro vyvtvoření nového obrazu z části s obličejem z původního
// Zahrnuje úpravu obličejové části na čtvrcový formát pro modely ML
function getImagePartWithFace(face) {
    console.log(face);
    const size = Math.max(face.box.width, face.box.height);
    // Canvas pro změnu velikosti obrázku
    /* pokud by se použil jen img, fce drawImage pracuje s atributem
    naturalWidth místo width, dojde ke špatnému mapování pixelu */
    const can = document.createElement('canvas');
    const ctx = can.getContext('2d');
    can.width = img.width;
    can.height = img.height;
    ctx.drawImage(img, 0, 0, img.width, img.height);
    // Výsledný canvas pro přenesení obličejové části obrazu
    const can2 = document.createElement('canvas');
    const ctx2 = can2.getContext('2d');
    can2.width = size;
    can2.height = size;
    let xMin = face.box.xMin
    let yMin = face.box.yMin
    if(face.box.width > face.box.height){
        yMin = yMin - (face.box.width - face.box.height) / 2
    }else{
        xMin = xMin - (face.box.height - face.box.width) / 2
    }
    ctx2.drawImage(
        can,
        xMin, yMin, size, size, // Odkud brát z originálního obrázku
        0, 0, size, size // Kam vykreslit na canvas
    );
    return can2;
}
```

Zdrojový kód 5: Funkce pro detekci věku a funkce pro separaci obličejů do samostatných obrázků. Zdroj vlastní.

7.4 Nasazení aplikace

Nasazení aplikace bylo provedeno pomocí platformy Netlify, která umožňuje jednoduché a rychlé nasazení moderních aplikací. Po dokončení vývoje aplikace je nutné spustit příkaz *npm run build*, který vytvoří adresář s buildem aplikace (adresář *dist*) a tento adresář následně umístit do Netlify. Odkaz na spuštění aplikace je uveden v příloze.

8 TESTOVÁNÍ

Testování proběhlo na několika zařízeních. Cílem bylo vyhodnotit, zda aplikace i na méně výkonném hardwaru bude poskytovat výsledky v přijatelném čase, a to především při použití videa jako vstupu, které představuje vyšší zátěž. Dalším cílem je také ověřit responzivitu aplikace na mobilních zařízeních.

Použitá zařízení:

- Herní Notebook HP Omen (Intel Core i5 - 4 jádra, NVIDIA GTX 1650, 16 GB RAM) – zástupce výkonnějšího zařízení
- Notebook HP Pavilion – zástupce slabšího desktopového zařízení (Intel Core i3 – 2 jádra, integrovaná grafická karta, 4 GB RAM) - zástupce slabšího zařízení
- Smartphone Samsung Galaxy S23 – zástupce výkonnějšího mobilního zařízení
- Smartphone Samsung Galaxy A33 – zástupce slabšího mobilního zařízení

Obsah testování:

- Na každém zařízení bylo otestováno postupné nahrání několika fotografií z úložiště, nad kterými byly provedeny všechny dostupné analýzy, přičemž fotografie byly v různých velikostech.
- Na každém zařízení byl spuštěn režim video vstupu z kamery zařízení a jednotlivé analýzy byly po jedné spouštěny.
- Na každém zařízení byl spuštěn režim video vstupu z kamery zařízení a bylo spouštěno několik analýz současně.
- Spuštění a ověření funkčnosti aplikace na různých internetových prohlížečích.

Výsledek:

Aplikace byla úspěšně spuštěna na všech testovaných zařízeních. I na méně výkonném hardwaru bylo možné provést všechny plánované scénáře. Na slabších zařízeních bylo pozorováno delší počáteční načítání modelů strojového učení, což však nebránilo jejich plnohodnotnému použití. U fotografického vstupu probíhala analýza na všech zařízeních bez problému, pomalejší dostupnost výsledků analýz u slabších zařízení byla z pohledu uživatelského zážitku zanedbatelná. Při použití video vstupu na výkonnějším zařízení, a to jak mobilním, tak i desktopovém je i při provádění všech analýz výstup plynulý. Slabší zařízení zvládají dobře spouštění samostatných analýz, ovšem při spouštění více analýz je už patrné, že

výstupy modelů nejsou vždy dostupné včas. Responzivita aplikace byla na testovaných zařízeních taktéž bez problémů. V rámci testování na různých prohlížečích byl zjištěn drobný problém pouze u prohlížeče Firefox, který nepodporuje WebGPU. Zpomalení je znát především v režimu vstupu z kamery. Na ostatních testovaných prohlížečích – Google Chrome, Microsoft Edge i prohlížeč Samsung běží aplikace bez problémů. Pro spouštění aplikace je tak doporučeno používat prohlížeč s podporou WebGPU.

ZÁVĚR

Výsledná aplikace slouží jako důkaz toho, že i v prostředí webového prohlížeče lze efektivně využít moderní nástroje pro rozpoznávání obrazu. Volba využití knihovny TensorFlow.js se ukázala jako správná. Je možné provádět detekci a analýzu obličejů přímo na straně klienta, bez nutnosti serverového zpracování, což přináší výhody zejména z hlediska ochrany soukromí uživatelů, rychlejší odezvy aplikace a také lehčí architektury aplikace. Sílu zde také ukazuje moderní backend WebGPU, který citelně urychluje výpočty strojového učení.

Knihovna poskytuje možnosti pro základní práci s obličejem – jeho detekce a rozpoznání klíčových rysů. Zde je nutné podotknout, že řešení je zaměřeno spíše pro použití na vstup získaný z kamery zařízení, kde obličeje jsou blízko kamery. Pro tyto vstupy modely poskytují poměrně přesné výstupy, pokud ovšem je obličej ve větší vzdálenosti od kamery (cca 2 m a více), model již selhává a není schopen obličej rozpoznat. Dalším drobným nedostatkem je detekce více obličejů. Pokud jich je více blízko sebe, model často chybuje.

Práce rovněž ukazuje, jak lze využít různé přístupy k trénování modelů – od použití dostupných nástrojů, až po návrh a nasazení vlastního modelu. Vytvořené modely dosahují uspokojivých výsledků a v aplikaci jsou úspěšně integrovány pro reálné použití, kde poskytují predikce v odhadu věku a detekci pohlaví a emoce uživatele.

Aplikace v současné podobě má především demonstrační záměr. Použitelná je pro výukové účely nebo různé zábavné experimenty. V modifikované verzi by ji ovšem bylo možné použít například v odvětvích UX, marketingu nebo v interaktivních hrách.

V teoretické části se povedlo čtenáře obeznámit s problematikou strojového učení, konvolučních neuronových sítí a v neposlední řadě také představit knihovnu TensorFlow.js.

Jisté nedostatky lze spatřovat ve vytvořených modelech. Zde by se do budoucna jistě nabízela možnost vylepšit tyto modely, aby jednak dosahovaly lepší přesnosti, ale i rychlejších predikcí. To by bylo možné jak změnou samotné topologie, tak i hledáním kvalitnějších datasetů pro trénink. Aplikaci by bylo dále možné rozšířit například o detekci dalších atributů (přítomnost brýlí, vousů nebo roušky).

POUŽITÁ LITERATURA

[1] KUMAR, Ashu, KAUR, Amandeep a KUMAR, Munish. Face detection techniques: a review. In: *Springer Nature Link.* [Online] 4. 7. 2018. [cit. 2025-05-02]. Dostupné z: https://link.springer.com/article/10.1007/s10462-018-9650-2.

[2] YANG, Shuo, et. al. WIDER FACE: A Face Detection Benchmark. In: *The Chinese University of Hong Kong.* [Online] ©2016. [cit. 2025-05-02]. Dostupné z: https://openaccess.thecvf.com/content_cvpr_2016/papers/Yang_WIDER_FACE_A_CVPR_2016_paper.pdf.

[3] ML Kit. Face detection. In: *ML Kit.* [Online] ©2025. [cit. 2025-05-02]. Dostupné z: https://developers.google.com/ml-kit/vision/face-detection.

[4] HJELMAS, Erik . Face Detection: A Survey. In: *ScienceDirect.* [Online] 17. 4. 2001. [cit. 2025-05-02]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S107731420190921X.

[5] SOLTANY, Milad. Face-Detection. In: *GitHub.* [Online] 17. 5. 2021. [cit. 2025-05-02]. Dostupné z: https://github.com/miladsoltany/Face-Detection?tab=readme-ov-file.

[6] KAUSHIK, Himanshu. Machine Learning vs Human Learning: The Battle for Future Dominance. In: *Medium.* [Online] 21. 10. 2023. [cit. 2025-05-02]. Dostupné z: https://medium.com/@himanshubangalore/machine-learning-vs-human-learning-the-battle-for-future-dominance-fa7a1c99cd0c.

[7] BROWNLEE, Jason. Difference Between Algorithm and Model in Machine Learning. In: *Machine Learning Mastery.* [Online] 19. 8. 2020. [cit. 2025-05-02]. Dostupné z: https://machinelearningmastery.com/difference-between-algorithm-and-model-in-machine-learning/.

[8] Nixus. Machine Learning Algorithms. In: *Nixus.* [Online] ©2025. [cit. 2025-05-02]. Dostupné z: https://nixustechnologies.com/machine-learning-algorithms/.

[9] KUMAR, Ajitesh. Most Common Machine Learning Tasks. In: *Analytics Yogi.* [Online] 4. 12. 2022. [cit. 2025-05-02]. Dostupné z: https://vitalflux.com/7-common-machine-learning-tasks-related-methods/.

[10] ZHOU, Zhi-Hua. *Machine Learning.* Shaowu LIU (překladatel). Singapore : Springer Nature, 2021. ISBN 978-981-15-1967-3.

[11] Pecan Team. Data Preparation for Machine Learning: The Ultimate Guide to Doing It Right. In: *Pecan.* [Online] 21. 11. 2023. [cit. 2025-05-02]. Dostupné z: https://www.pecan.ai/blog/data-preparation-for-machine-learning/.

[12] BIBA, Jacob. 4 Types of Machine Learning to Know. In: *Built In.* [Online] 18. 7. 2024. [cit. 2025-05-02]. Dostupné z: https://builtin.com/machine-learning/types-of-machine-learning.

[13] WIJAYA, Cornellius Yudha. Exploring Unsupervised Learning Metrics. In: *KDnuggets.* [Online] 13. 4. 2023. [cit. 2025-05-02]. Dostupné z: https://www.kdnuggets.com/2023/04/exploring-unsupervised-learning-metrics.html.

[14] IBM Technology. What is Semi-Supervised Learning?. In: *YouTube.* [Online] 3. 2. 2025. [cit. 2025-05-02]. Dostupné z: https://youtu.be/C3Lr6Waw66g?si=XFmpmRuiH5JUAvYk.

[15] BERGMANN, Dave, STRYKER, Cole. What is a loss function?. In: *IBM.* [Online] 12. 7. 2024. [cit. 2025-05-02]. Dostupné z: https://www.ibm.com/think/topics/loss-function.

[16] sabrepc. Epochs, Batch Size, Iterations - How are They Important to Training AI and Deep Learning Models. In: *sabrepc.* [Online] 9. 8. 2024. [cit. 2025-05-02]. Dostupné z: https://www.sabrepc.com/blog/Deep-Learning-and-AI/Epochs-Batch-Size-Iterations.

[17] MUCCI, Tim. Overfitting vs. underfitting: Finding the balance. In: *IBM.* [Online] 11. 12. 2024. [cit. 2025-05-02]. Dostupné z: https://www.ibm.com/think/topics/overfitting-vs-underfitting.

[18] HOLBROOK, Ryan, COOK, Alexis. Overfitting and Underfitting. In: *Kaggle.* [Online] 20. 4. 2023. [cit. 2025-05-02]. Dostupné z: https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting.

[19] HOLDSWORTH, Jim, SCAPICCHIO, Mark. What is deep learning? In: *IBM.* [Online] 17. 6. 2024. [cit. 2025-05-02]. Dostupné z: https://www.ibm.com/think/topics/deep-learning.

[20] ZANDL, Patrick. Neuronové sítě. In: *Marigold.cz.* [Online] 6. 7. 2024. [cit. 2025-05-02]. Dostupné z: https://www.marigold.cz/ai/neuronove-site/.

[21] GeeksforGeeks. Multi-Layer Perceptron Learning in Tensorflow. In: *GeeksforGeeks.* [Online] 5. 2. 2025. [cit. 2025-05-02]. Dostupné z: https://www.geeksforgeeks.org/multi-layer-perceptron-learning-in-tensorflow/.

[22] O'SHEA, Keiron, NASH, Ryan. An Introduction to Convolutional Neural Networks. In: *Cornell Iniversity.* [Online] 26. 11. 2015. [cit. 2025-05-02]. Dostupné z: https://arxiv.org/abs/1511.08458.

[23] SAHA, Sumit. A Guide to Convolutional Neural Networks — the ELI5 way. In: *SaturnCloud.* [Online] 15. 12. 2018. [cit. 2025-05-02]. Dostupné z: https://saturncloud.io/blog/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way/.

[24] DOLEŽEL, Petr. INUI2, NNUI2 - Přednáška (Konvoluční umělá neuronová síť). In: *Youtube.* [Online] 27. 4. 2020. [cit. 2025-05-02]. Dostupné z: https://youtu.be/-2vEi-Aa0FA?si=jWV5LkydYewM_KG_.

[25] VISHNUAM. Activation Functions in Convolutional Neural Networks (CNN). In: *Medium.* [Online] 6. 10. 2024. [cit. 2025-05-02]. Dostupné z: https://medium.com/@vishnuam/activation-functions-in-convolutional-neural-networks-cnn-b1114a1b9b4d.

[26] Team SuperDataScience. Convolutional Neural Networks (CNN): Step 3 - Flattening. In: *SuperDataScience.* [Online] 18. 8. 2018. [cit. 2025-05-02]. Dostupné z: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening.

[27] COTA, Sasirekha. Deep Learning Basics — Part 8 — Convolutional Neural Network (CNN). In: *Medium.* [Online] 12. 12. 2023. [cit. 2025-05-02]. Dostupné z: https://medium.com/@sasirekharameshkumar/deep-learning-basics-part-8-convolutional-neural-network-cnn-4cff567bad46.

[28] BISWAS, Avishek. The History of Convolutional Neural Networks for Image Classification (1989- Today). In: *towards data science.* [Online] 28. 6. 2024. [cit. 2025-05-02]. Dostupné z: https://towardsdatascience.com/the-history-of-convolutional-neural-networks-for-image-classification-1989-today-5ea8a5c5fe20/.

[29] SMILKOV, Daniel, et al. TENSORFLOW.JS: MACHINE LEARNING FOR THE WEB AND BEYOND. In: *MLSys Proceedings.* [Online] ©2019. [cit. 2025-05-02]. Dostupné z: https://proceedings.mlsys.org/paper_files/paper/2019/file/acd593d2db87a799a8d3da5a860c028e-Paper.pdf.

[30] ZHYGAILO, Alexander. How and why we decided to use TF.js for a digital health product prototype. In: *MEV.* [Online] 29. 10. 2024. [cit. 2025-05-02]. Dostupné z: https://mev.com/blog/how-and-why-we-decided-to-use-tf-js-for-a-digital-health-product-prototype.

[31] TensorFlow. Tensors and operations. In: *TensorFlow.* [Online] 1. 11. 2022. [cit. 2025-05-02]. Dostupné z: https://www.tensorflow.org/js/guide/tensors_operations.

[32] Tensorflow. Platform and environment. In: *Tensorflow.* [Online] 1. 11. 2022. [cit. 2025-05-02]. Dostupné z: https://www.tensorflow.org/js/guide/platform_environment.

[33] SOULANILLE, Matthew. tfjs-backend-webgpu. In: *GitHub.* [Online] 23. 4. 2025. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs/tree/51577688f1c695db72ed75176875a2d3c8c3ea29/tfjs-backend-webgpu.

[34] admin. WebGPU computations performance in comparison to WebGL. In: *Pixels Commander.* [Online] 30. 11. 2021. [cit. 2025-05-02]. Dostupné z: https://pixelscommander.com/javascript/webgpu-computations-performance-in-comparison-to-webgl/.

[35] TensorFlow.js Community. How Modiface utilized TensorFlow.js in production for AR makeup try on in the browser. In: *TensorFlow Blog.* [Online] 24. 2. 2020. [cit. 2025-05-02]. Dostupné z: https://blog.tensorflow.org/2020/02/how-modiface-utilized-tensorflowjs-in-ar-makeup-in-browser.html?_gl=1*mzzzym*_ga*MTI2NDY0NjEwLjE3NDA0OTIwNTE.*_ga_W0YLR4190T*MTc0MjY4Mzk1OC4xNi4xLjE3NDI2ODg1NjAuMC4wLjA..

[36] TensorFlow.js Community. InSpace: A new video conferencing platform that uses TensorFlow.js for toxicity filters in chat. In: *TensorFlow Blog.* [Online] 21. 12. 2020. [cit. 2025-05-02]. Dostupné z: https://blog.tensorflow.org/2020/12/inspace-new-video-conferencing-platform-uses-tensorflowjs-for-toxicity-filters-in-chat.html.

[37] TensorFlow.js Community. How Adobe used Web ML with TensorFlow.js to enhance Photoshop for web. In: *TensorFlow Blog.* [Online] 30. 3. 2023. [cit. 2025-05-02]. Dostupné z: https://blog.tensorflow.org/2023/03/how-adobe-used-web-ml-with-tensorflowjs-to-enhance-photoshop-for-web.html.

[38] TensorFlow.js Community. How LinkedIn Personalized Performance for Millions of Members using TensorFlow.js. In: *TensorFlow Blog.* [Online] 29. 3. 2022. [cit. 2025-05-02]. Dostupné z: https://blog.tensorflow.org/2022/03/how-linkedin-personalized-performance.html.

[39] TensorFlow.js. How DermAssist uses TensorFlow.js for on-device image quality checks. In: *TensorFlow Blog.* [Online] 11. 10. 2021. [cit. 2025-05-02]. Dostupné z: https://blog.tensorflow.org/2021/10/how-DermAssist-uses-TensorFlowJS.html.

[40] BAZAREVSKY, Valentin, KARTYNNIK, Yury, ABLAVATSKI, Artsiom. MODEL CARD Mediapipe BlazeFace. In: *Google Drive.* [Online] 9. 6. 2021. [cit. 2025-05-02]. Dostupné z: https://drive.google.com/file/d/1d4-xJP9PVzOvMBDgIjz6NhvpnlG9_i0S/preview.

[41] GRISHCHENKO, Ivan, ABLAVATSKI, Artsiom, KARTYNNIK, Yury. MODEL CARD MediaPipe Attention Mesh. In: *Google Drive.* [Online] 28. 10. 2020. [cit. 2025-05-02]. Dostupné z: https://drive.google.com/file/d/1tV7EJb3XgMS7FwOErTgLU1ZocYyNmwlf/preview.

[42] TensorFlow.js Team. MobileNet. In: *GitHub.* [Online] 24. 10. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/mobilenet.

[43] TensorFlow.js Team. Object Detection (coco-ssd). In: *GitHub.* [Online] 24. 10. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/coco-ssd.

[44] Tensorflow.js Team. Semantic Segmentation in the Browser: DeepLab v3 Model. In: *GitHub.* [Online] 29. 4. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/deeplab.

[45] TensorFlow.js Team. Pose Detection. In: *GitHub.* [Online] 21. 5. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/pose-detection.

[46] TensorFlow.js Team. Body Segmentation. In: *GitHub.* [Online] 24. 10. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/body-segmentation.

[47] TensorFlow.js Team. Hand Pose Detection. In: *GitHub.* [Online] 21. 5. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/hand-pose-detection.

[48] TensorFlow.js Team. Depth Estimation. In: *GitHub.* [Online] 24. 10. 2024. [cit. 2025-05-02]. Dostupné z: https://github.com/tensorflow/tfjs-models/tree/master/depth-estimation.

[49] Brain.js. Brain.js: GPU accelerated Neural networks in JavaScript. In: *Brain.js.* [Online] [cit. 2025-05-02]. Dostupné z: https://brain.js.org/#/getting-started.

[50] ConvNetJS. Intro. In: *ConvNetJS.* [Online] [cit. 2025-05-02]. Dostupné z: https://cs.stanford.edu/people/karpathy/convnetjs/index.html.

[51] ml5.js. About. In: *ml5.js.* [Online] 2018. [cit. 2025-05-02]. Dostupné z: https://ml5js.org/about/.

[52] WANG, Yulong. ONNX.js. In: *GitHub.* [Online] 14. 12. 2021. [cit. 2025-05-02]. Dostupné z: https://github.com/microsoft/onnxjs?tab=readme-ov-file.

[53] HARTINGER, David. Lekce 2 - UML - Use Case Diagram. In: *itnetwork.cz.* [Online] ©2025. [cit. 2025-05-02]. Dostupné z: https://www.itnetwork.cz/navrh/uml/uml-use-case-diagram.

[54] Europe, Sparx System. Use Case Diagram. In: *Sparx System Europe.* [Online] ©2025. [cit. 2025-05-02]. Dostupné z: https://www.sparxsystems.eu/languages/uml/diagrams/usecasediagram/.

[55] creately. Activity Diagram Tutorial | Easy Guide with Examples. In: *creately.* [Online] 29. 11. 2022. [cit. 2025-05-02]. Dostupné z: https://creately.com/guides/activity-diagram-tutorial/.

[56] SAWANT13, Ashish. Facial Emotion Expressions (FER) Balanced. In: *Kaggle.* [Online] 27. 3. 2025. [cit. 2025-05-02]. Dostupné z: https://www.kaggle.com/datasets/ashishsawant13/facial-emotion-expressions-fer-balanced/.

[57] KHAIREDDIN, Yousif, CHEN, Zhuofa. Facial Emotion Recognition: State of the Art Performance on FER2013. In: *Boston University.* [Online] 8. 5. 2021. [cit. 2025-05-02] https://arxiv.org/abs/2105.03588.

[58] TensorFlow. Model conversion. In: *TensorFlow.* [Online] 12. 10. 2023. [cit. 2025-05-02]. Dostupné z: https://www.tensorflow.org/js/guide/conversion.

[59] TECHZIZOU. Build a custom Image classification Android app using Teachable Machine (Old version TF App). In: *Medium.* [Online] 19. 10. 2022. [cit. 2025-05-02]. Dostupné z: https://medium.com/geekculture/build-a-custom-image-classification-android-app-using-teachable-machine-f60b197eaa90.

[60] CHAUHAN, Ashutosh. Gender Classification Dataset. In: *Kaggle.* [Online] 15. 12. 2019. [cit. 2025-05-02]. Dostupné z: https://www.kaggle.com/datasets/cashutosh/gender-classification-dataset.

[61] SUBEDI, Sanjaya. UTKFace. In: *Kaggle.* [Online] 16. 8. 2018. [cit. 2025-05-02] https://www.kaggle.com/datasets/jangedoo/utkface-new/data.

[62] TensorFlow. MobileNetV2. In: *TensorFlow.* [Online] 7. 6. 2024. [cit. 2025-05-02]. Dostupné z: https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV2.

PŘÍLOHY

PŘÍLOHA A – ZDROJOVÉ KÓDY A APLIKACE

Odkaz na spuštění aplikace: https://dip-face-app.netlify.app/

Součástí elektronické verze této práce je příloha obsahující:

- projekt se zdrojovými kódy aplikace
- adresář se zdrojovými kódy k vytvářeným modelům strojového učení.