VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

DETEKCE VÝSKYTU OBJEKTŮ VE VIDEOZÁZNAMU

DETECTING THE OCCURRENCE OF OBJECTS IN A VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAN ŠAMÁNEK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. TOMÁŠ GOLDMANN,**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Šamánek Jan**
Program: Informační technologie
Název: **Detekce výskytu objektů ve videozáznamu**
Detecting the Occurrence of Objects in a Video
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s problematikou detekce objektů ve videu. Především se zaměřte na moderní řešení umožňující analýzu videa.
2. Prostudujte a sumarizujte dostupné algoritmy, které se používají pro detekci a klasifikaci objektů ve videu.
3. Navrhněte řešení, které bude umožňovat detekci minimálně 5 různých objektů. Řešení navrhněte tak, aby bylo možné jednoduše měnit klasifikační algoritmus. Dále navrhněte algoritmus pro monitorování vybrané oblasti ve videu. V případě, že se v této oblasti vyskytne vybraný objekt, dojde k uložení času výskytu.
4. Implementujte navržené řešení v libovolné programovacím jazyce. Pro ovládání vytvořte jednoduché uživatelské rozhraní.
5. Proveďte experimenty zaměřené na určení výkonnosti použitého detektoru.

Literatura:

- CIPOLLA, Roberto, Sebastiano BATTIATO a Giovanni Maria FARINELLA. Computer vision: detection, recognition and reconstruction. Berlin: Springer, 2010, 350 s. : il., fot. ISBN 978-3-642-12847-9.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Goldmann Tomáš, Ing.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 31. října 2019

Abstrakt

Tato bakalářská práce se zabývá detekcí objektů ve videozáznamu, především pomocí konvolučních neuronových sítí a implementace jednoduchého uživatelského rozhraní, které dovolí uživateli vybírání mezi klasifikátory a jejich využití pro analýzu videa a jejich trénování na vlastním datasetu. První část je dedikovaná popisu strojového učení a neuronových sítí. Poté následuje část pro popis segmentace a klasifikace obrazu pomocí algoritmů strojového učení a předzpracování dat pro trénování modelů. Poslední je praktická část, která popisuje návrh vytvořeného modelu, uživatelského rozhraní a dosažených výsledků.

Abstract

This bachelor thesis deals with detection of objects in videos by using primarily convolution neural networks and creating simple user interface, which allows user to choose classification model and use it to analyze video or train given model on own dataset. First part is dedicated to description of machine learning and neural networks. After that follows the section about image description and image classification using machine learning algorithms and data augmentation. Last part deals with describtion of own design of neural network and user interface and describing achieved results.

Klíčová slova

umělá Inteligence, strojové učení, neuronové sítě, konvoluční neuronové sítě, obraz, práce s obrazem, klasifikace obrazu, Python, Opencv, Tensorflow.

Keywords

artificial Intelligence, machine learning, neural networks, convolutional neural networks, image, image segmentation, image classification, Python, Opencv, Tensorflow.

Citace

ŠAMÁNEK, Jan. *Detekce výskytu objektů ve videozáznamu.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Tomáš Goldmann,

Detekce výskytu objektů ve videozáznamu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Tomáše Goldmanna. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Jan Šamánek
29. května 2020

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Tomáši Goldmannovi za cenné připomínky a rady, které vedli k vypracování této bakalářské práce. Následné bych chtěl také poděkovat mé přítelkyni a rodině za morální podporu a stylistickou a gramatickou korekturu.

Také bych chtěl v neposlední řadě poděkovat výpočetnímu centru MetaCentrum za možnost využití jejich výpočetních zdrojů.

Obsah

Kapitola 1

Úvod

V posledních letech je čím dál tím více kladený důraz na analýzu videa a detekování objektů ať už ve videozáznamu nebo v reálném čase pomocí agilních přístupů, které je možné měnit a trénovat. Takovéto technologie mají obrovský potenciál od analýzy, statistik, bezpečnosti až po vědecké účely.

Tato práce se zaměřuje na agilní a lehce měnitelné přístupy k analýze videí, jejich porovnání a přesnost. Zejména se pak tato práce zaměřuje na možnost klasifikace pomocí neuronových sítí, což je nástroj využívaný ve strojovém učení na vytváření, nejen klasifikačních, ale také regresních a spoustu dalších typů systémů. Práce poskytuje základní popis strojového učení a neuronových sítí, kde je možné se dočíst jaká je myšlenka a funkcionalita učících se systémů, jaké jsou typy těchto systémů a k čemu se mohou využít. Následně je zde seznámení s moderními přístupy ke klasifikátorům, umožnujícím i klasifikaci obrazu a videa. Kromě výše zmíněných neuronových sítí je zde možné se seznámit s algoritmy shlukování, Random forests nebo GMM (*gaussian mixture model*). Práce se ovšem především zaměřuje na konvoluční neuronové sítě a to včetně jejich modifikací jako jsou R-CNN (*region proposals with convolutional neural network*) nebo YOLO network (*you only look once network*). V té samé části jsou rozepsány algoritmy pro předzpracování obrazu neboli *data augmentation*, které se v poslední době čím dál tím více využívá pro zlepšení trénovacího datasetu, dosažení větší diverzity a pozitivního vlivu na přesnost trénovaného klasifikátoru.

Jako další je zde pak rozepsán návrh vlastního modelu klasifikátoru a návrh uživatelského rozhraní, které je možné využít pro práci s vlastními klasifikátory.

V rámci této práce je i lehké výkonnostní a rychlostní porovnání se známými modely neuronových sítí jako je například VGGNet16, které má za účel zjistit jak dobrý vlastní vytvořený model je a jak velký dopad má vlastně velikost sítě na přesnost a rychlost analýzy. Kromě toho je zde i rozepsán vliv *data augmentation* na trénování a porovnání výsledků přesnosti trénovaných modelů, na kterých *data augmentation* bylo, nebo naopak nebylo použito.

Na závěr je uvedené lehké shrnutí práce a dosažených výsledků společně s návrhy na možná zlepšení.

Kapitola 2

Strojové učení a neuronové sítě

První částí této práce je popis samotné podstaty a myšlenky takzvaných *Machine Learning* (ML) systémů, neboli systémů strojového učení. Dále jsou zde pak popsány typy strojového učení a problémy, na které se dají ML systémy využívat.

Další velkou částí je pak seznámení se s neuronovými sítěmi, které jsou, v poslední době, čím dál tím více používaný jako nástroj pro implementaci těchto agilních, učících se systémů.

2.1 Strojové učení

Tato sekce vychází ze zdroje [26]. Jako první ze všeho musíme pochopit co to je strojové učení, k čemu se využívá a proč se zasazuje čím dál tím více do reálných systémů. Podle zdroje [8] můžeme strojové učení definovat jako podoblast informatiky a umělé inteligence. Hlavní definicí strojového učení je vytváření počítačového systému, který na základě trénování mění svůj vnitřní stav tak, aby co nejlépe reagoval na změny okolního prostředí. Tyto systémy se v této době zasazují do stále více oblastí jako je například rozpoznávání textu, hraní her, skládání písní nebo identifikace objektů v obraze.

2.1.1 Typy učení

Strojové učení můžeme rozdělit na učení s učitelem, bez učitele a posilované nebo také motivované učení. Speciálním případem je poté kombinace učení s a bez učitele.

Učení s učitelem

Jde o případy učení nebo také trénování, kdy máme velkou množinu vstupních dat x a jejich požadované výstupy y. Pomocí trénování systému se se vytvoří funkce $y = f(x)$. Učení se dosahuje pomocí předložení systému trénovací množinu X poskládanou z předem definovaných dvojic x, y kde x je prvek testovacích dat a y je požadovaný výsledek pro dané x. Systému předložíme x a následně porovnáme výsledek s y. Případnou chyba se poté zmenšuje pomocí algoritmu Back-propagation, uvedeného v sekci 2.2.5.

Tyto typy systémů na rozdíl od ostatních typů mají přesně určené dynamiky. Neboli kdy se bude systém trénovat a kdy je trénovací proces u konce. Jedná se takzvaně o dávkový algoritmus, neboli systém požaduje veškerá trénovací data před nasazením do provozu. To znamená, že systém je nutné první natrénovat na dedikovaných datech a poté se na testovacích a skutečných datech už vnitřní stav systému nemění.

Problém tohoto typu systému je v jeho omezení. Systém může být jen natolik dobrý jako je trénovací množina X, na které se trénuje. To znamená, že teoreticky vždycky bude existovat případ, na který nebude systém natrénován.

Tento typ systému se nejčastěji používá pro řešení problémů regrese a klasifikace.

Problém učení s učitelem je hlavně v získání trénovací množiny. Často jsou potřeba tisíce až desetitisíce příkladů v trénovací množině X s důrazem na rozmanitost a normalizaci trénovací množiny, aby se dalo dosáhnout co největší přesnosti systému.

Učení bez učitele

Jedná se o podobný systém jako v případě učení s učitelem. Rozdílem je, že trénovací množina nemá žádné předem známe výsledky. Jde o případ, kdy je systém nechán ať nalezne sám své nejlepší řešení problému. Tomu se děje pomocí hledání skrytých vzorů a pravidel mezi vstupními daty. Systém tedy hledá mezi sebou korelující prvky, které následně shlukuje do sebe. Jedná se především o problémy typu asociace a shlukové analýzy. Příkladem algoritmu pro shlukovou analýzu může být například K-means, který je více rozepsaný v sekci 3.2.2.

Posilované učení

Podle zdroje [45] je posilované učení nejpřirozenější způsob učení pro živé bytosti. Celý systém posilovaného učení funguje na jednoduchém pravidlu. Správná funkce systému je odměněna a nesprávná funkce je potrestána. Samozřejmě odměna a trest jsou v tomto kontextu realizovány pouze číselným ohodnocením.

Jedná se o jiný typ systému, než jsou výše zmíněné. Neexistuje zde žádná trénovací množina a proces trénování a testování je možné sloučit do jedné. Systémy s tímto typem učení jsou většinou nasazovány na problémy, kde chceme, aby byl výsledný systém lepší než člověk nebo jakýkoliv jiný systém. Výhodou je možnost stálého učení i při nasazení.

Systém je zpravidla definovaný agentem, prostorem, ve kterém se agent pohybuje a pravidly:

S	množina stavů prostoru,
A	množina akcí,
δ	transformační funkce,
r	odměna,
π	strategie chování,
V^{π}	hodnotová funkce.

Posilované učení funguje ve 2 fázích. První je fáze učení, kdy se určuje správná strategie chování π a ve druhé fázi už se tato strategie využívá pro řešení daného problému. Existuje speciální případ, kdy jsou tyto dvě fáze prolnuty do sebe. V takovém případě se jedná o inkrementální typ systému a nazývá se takzvaně učení za chodu. Z dlouhodobého hlediska je pak tento způsob výhodnější, avšak je zde problém veliké nepřesnosti počátečních výsledků.

Ve výsledku máme agenta, který sleduje situaci kolem sebe a rozhoduje se kam se pohnout v předem definovaném systému. Pokud se pohne správnou cestou, jeho pohyb je ohodnocen kladnou zpětnou vazbou. Naopak pokud se pohne nesprávně, přijde zpětná vazba záporná. Takto dokážeme ovládat chování agenta.

Příkladem systému, využívající posilované učení, může být například laboratorní myš v labyrintu, která se snaží dostat k sýru. Myš zastává roli agenta, labyrint roli prostoru,

ve kterém se agent pohybuje a sýr slouží jako odměna. Pokud dáme myš na jeden konec labyrintu a kousek sýru na druhý konec, myš dříve či později nalezne cestu k sýru. Dále je možné nastavit labyrint o 2 cestách, které vedou k sýru. Jedna cesta se ovšem určí jako nesprávná a pokud jí myš půjde, bude jí odebrán sýr a myš bude ponechána o hladu. Hlad zastává roli trestu. Po pár pokusech se myš naučí, že jediná správná cesta k odměně je vybrat cestu, která nebyla určena jako špatná. Takto je možné ovládat pohyb agenta v prostoru a jedná se o základní myšlenku posilovaného učení.

Tento způsob učení má velkou škálu použití jako například hraní her, řízení aut nebo řízení robotů.

2.2 Neuronové sítě

Tato sekce vychází ze zdroje [13]. Neuronové sítě jsou čím dál víc nasazovanější nástroj pro strojové učení a dají se chápat jakožto síť nebo topologie vrstev elementárních částic zvaných neurony, které jsou společně propojeny. Myšlenka umělého neuronu zde byla už od první poloviny minulého století. Důvodem byla snaha přiblížit počítačové myšlení blíže k tomu lidskému. Jako inspirace pro tyto neurony slouží skutečné neurony v nervové soustavě lidí a zvířat. Formálně jsou pak tyto umělé neurony nazývány jako perceptrony.

S aplikací neuronových sítí byl v minulosti problém, a to zejména kvůli nedostatečnému výkonu výpočetní techniky. Důvodem tohoto problému bylo, že i přesto, že jsou umělé neurony velice jednoduché, jedna síť jich může obsahovat veliké množství a také protože neurony v síti zpravidla fungují paralelně.

2.2.1 Perceptron

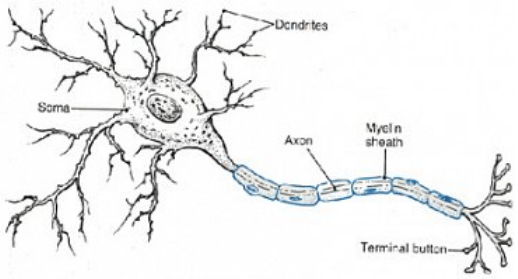
Model umělého neuronu podle zdroje [13] je skládán z n vstupních číselných hodnot, hlavičky perceptronu, a právě jedné výstupní číselné hodnoty. Každá vstupní hodnota má svoji váhu a hodnotu. Hlavička neuronu obsahuje aktivační funkci. Výsledkem je poté hodnota, která pokračuje buď na výstup neuronové sítě nebo do další skryté vrstvy neuronů.

Celková funkčnost perceptronu je popsána rovnicí (2.2.1.1) a znamená, že se veškeré vstupy neuronu vynásobí vlastní vahou, následně se sečtou a tento výsledek se vloží do aktivační funkce neuronu. Hodnota aktivační funkce je pak vyslána výstupní větví jako výsledek.

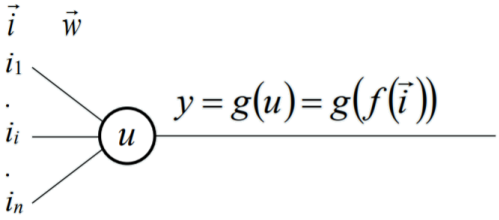
$$g(x) = \delta(\sum_{i=1}^{m}(w_i x_i) - b) \tag{2.2.1.1}$$

kde

$g(x)$	výsledná hodnota perceptronu,
m	počet vstupů perceptronu,
w	váha jednotlivých vstupů,
x	hodnota jednotlivých vstupů,
b	bias,
δ	aktivační funkce.



Obrázek 2.1: Znázornění biologického neuronu [9].



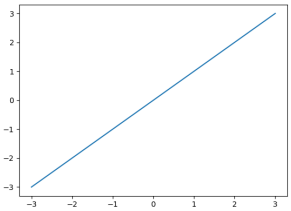
Obrázek 2.2: Model perceptronu [13].

2.2.2 Aktivační funkce

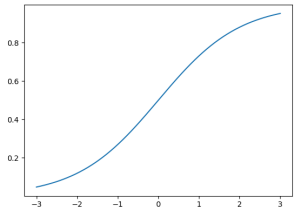
Jak už bylo zmíněno v kapitole 2.2.1, každý perceptron má svoji aktivační funkci. Jedná se zpravidla o křivku, která má $f(x)$ definované na určitém rozsahu jako například $f(x) \in <-1, 1>$. Perceptron následně sečte všechny jeho vstupy vynásobené jednotlivými vahami každého vstupu, odečte případnou hodnotu bias a výsledek vloží jako vstupní x aktivační funkci. Výsledná hodnota pro dané x je pak následně výstupem perceptronu.

Důvod využívání aktivační funkce je právě namapování hodnoty perceptronu do určitého intervalu. Křivka na tomto intervalu může být buď lineární anebo ji může tvořit polynom. Každý typ aktivační funkce se hodí na jiný typ problémů.

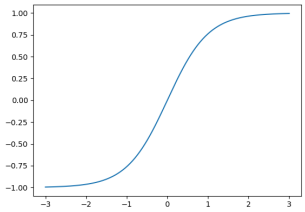
Jako příklady aktivačních funkcí je možné si uvést:



Obrázek 2.3: Lineární funkce $y = x$.



Obrázek 2.4: Sigmoida $y = \frac{1}{1+e^{-x}}$.



Obrázek 2.5: Hyperbolický tangens $y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

2.2.3 Využití neuronových sítí

Při vytváření neuronových sítí je potřeba se zaměřit na několik základních kroků. První musíme prostudovat problém, který má síť řešit a podle toho vybrat model. Každý model neuronové sítě má svou specializaci a hodí se na jiný typ úloh. Následně je potřeba zjistit klíčové body daného problému, které potřebujeme pro tvorbu našeho modelu. Například u problému klasifikování objektů v obraze je potřeba zjistit například velikost obrazu, zda je obraz barevný, výstupní třídy, které má model klasifikovat a další detaily. Poslední už jsou samotné problémy při implementaci neuronové sítě. Ty mohou být odlišné v závislosti na modelu a problému. Jaké nastavit prvotní váhy, kolik vložit vrstev, kolik neuronů budou mít skryté vrstvy a další. Přitom se musí brát ohled na požadovanou přesnost našeho modelu, ale i na časovou náročnost trénování modelu nebo na rychlost evaluace při případné analýze v reálném čase. Nejčastější typy úloh, na které se využívají neuronové sítě jsou regrese, klasifikace a shluková analýza.

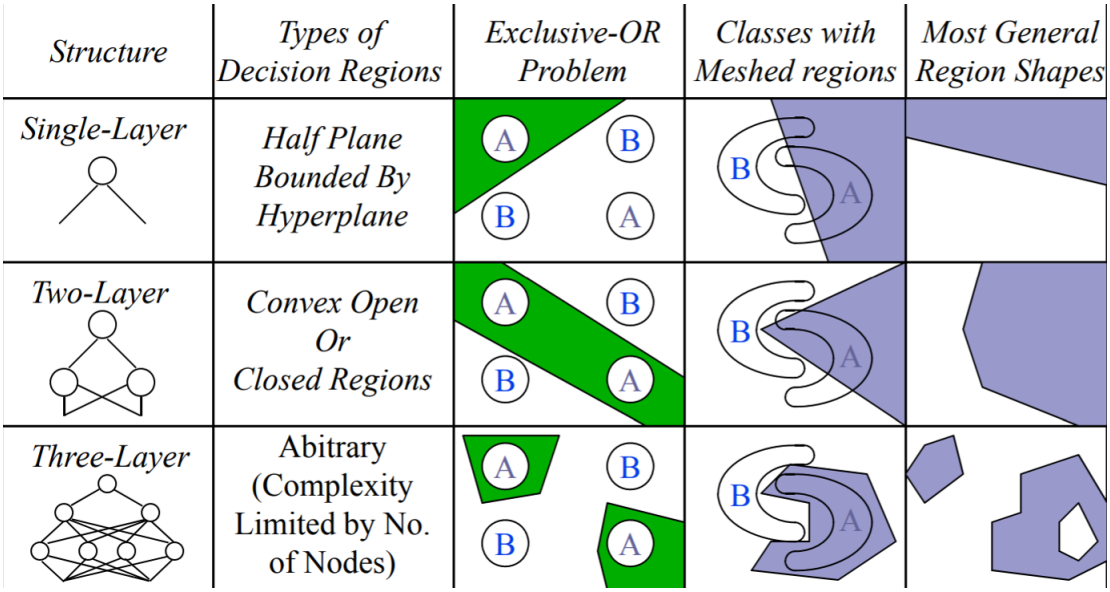
Regrese

Regresi si můžeme představit podle zdroje [1] jako proložení souboru bodů křivkou. Jednoduše řečeno, jde o pokus předvídat, jak bude určitý vývoj dále pokračovat při zadání kritéria. Jedná se tedy o problém aproximace bodů.

U neuronových sítí je úroveň křivky, která body prokládá určena podle počtu vrstev neuronové sítě. Jestliže síť nemá žádnou skrytou vrstvu, pak se jedná o takzvaný lineární klasifikátor a křivka bude mít tvar přímky. S každou další skrytou vrstvou se pak zvyšuje úroveň polynomu, který dané body prokládá.

Klasifikace

Sekce klasifikace vychází ze zdroje [50]. Úkolem je zařadit objekt do jedné z předem definovaných tříd. To znamená, že objekt je podobnější objektům v dané třídě, než objektům všech ostatních tříd. Charakteristika určení klasifikace se nazývá klasifikační pravidlo. Na to abychom byli schopni klasifikovat objekt, musíme jej umět popsat. Na to se používají metody příznakové a strukturální.



Obrázek 2.6: Neural Networks – An Introduction Dr. Andrew Hunter.

Shluková analýza

Tato sekce vychází z [18]. Jedná se zejména o metody učení bez učitele. Jde o nalezení podmnožin v množině vstupních dat. Matematická reprezentace takového úkolu je pak:

Nechť χ je libovolná množina n objektů. Rozklad $\Omega = C_1, C_2, ..., C_n$ množiny χ kde C jsou disjunktní neprázdné podmnožiny, které dohromady tvoří χ. Matematická reprezentace je poté:

$$C_i \cap C_j = \varnothing \tag{2.2.3.1}$$

a

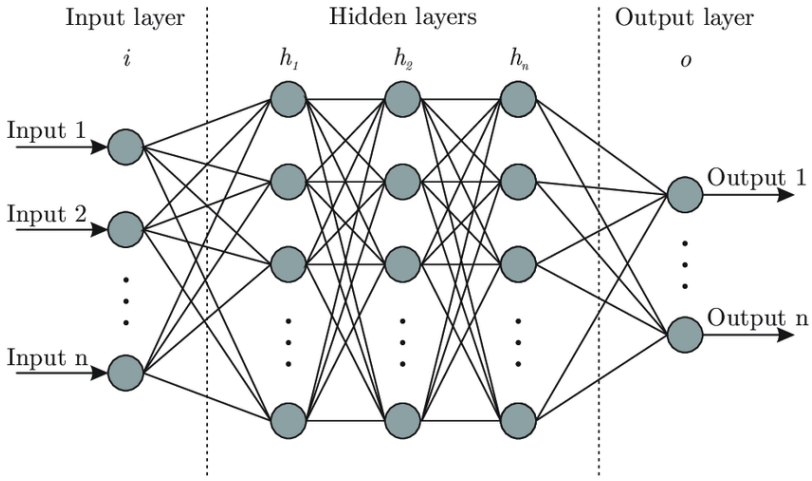
$$C_1 \cup C_2 \cup ... \cup C_m = \Lambda \tag{2.2.3.2}$$

Zjednodušeně řečeno, jde o typ úkolů, kdy dáme systému množinu dat a on je rozdělí do předem stanoveného počtu podmnožin, tak aby prvky každé podmnožiny byly co nejvíce podobné dalším prvkům stejné podmnožiny, ale zároveň co nejméně podobné prvkům ostatních podmnožin.

2.2.4 Typy modelů neuronových sítí

Každá neuronová síť se skládá z n vrstev neuronů, kde $n > 1$. To znamená, že síť má svou první vstupní vrstvu neuronů, která přijímá vstupní data a poslední výstupní vrstvu, která určuje výsledek, ke kterému síť dospěla. Mezi těmito vrstvami může být pak dalších m vrstev, které se nazývají skryté a slouží pro aplikace nelineární transformace vstupních dat. Vizualizace takového modelu je na obrázku 2.7.

Neuronových sítí je velké množství a dají se v základu rozdělit podle více pohledů jako například podle počtu vrstev, typu učení, nebo podle směru toku informací, které proudí neuronovou sítí. Příkladem se mohou uvést jednovrstvý a vícevrstvý perceptron, konvoluční neuronová síť, rekurentní neuronová síť nebo modulární sítě.



Obrázek 2.7: Model klasické neuronové sítě.

Jednovrstvý a vícevrstvý perceptron

Tato podsekce vychází z [25]. Jednovrstvý perceptron je často uváděn jakožto základní model neuronové sítě a slouží jako inspirace a základ dalším, komplexnějším modelům. Jedná se o jednoduchý model, vyobrazený na obrázku 2.7, seskládaný z vrstev neuronů, které jsou mezi sebou propojeny. Platí pravidlo, že každý neuron jedné vrstvy je propojen se všemi neurony vrstvy sousední. Neurony v rámci jedné vrstvy spolu propojeny nejsou. Rozdíl mezi jednovrstvým a vícevrstvým perceptronem je v počtu skrytých vrstev. Pokud model nemá žádnou skrytou vrstvu, tudíž má pouze vstupní a výstupní vrstvu neuronů, pak se jedná o jednovrstvý perceptron.

Je důležité si uvědomit jaké má vlastně jednovrstvý perceptron schopnosti. Pokud si představíme samostatný perceptron s binárním výstupem 0/1, pak můžeme říct, že takový perceptron je schopný rozlišit 2 třídy. Dělící linie takového perceptronu pak bude přímka, jejíž normálový vektor bude udáván nastavením vnitřních vah perceptronů. V případě, kdy

máme m perceptronů na výstupní vrstvě sítě, můžeme protnout prostor m takovými přímkami. To ovšem nezvyšuje schopnosti neuronové sítě. Chybí jí totiž schopnost kombinovat výstupy jednotlivých perceptronů a provádět nelineární transformace vstupních dat. Tento problém řeší vícevrstvý perceptron. Vyobrazení tohoto problému je na obrázku 2.6

Konvoluční neuronová síť

Speciální model neuronové sítě navržený hlavně pro práci s obrazem. Konvoluční neuronová síť vychází ze zdroje [39] a skládá se ze 2 základních částí. Extrakce dat a klasifikátoru. Extrakční část má za úkol dostat ze vstupních dat podstatné informace, se kterými může dále pracovat klasifikátor, který je zařadí do jedné z definovaných výstupních tříd. Obraz se dá definovat jako číselná matice také známá jako tenzor a právě s těmi konvoluční síť pracuje. Název konvoluční vyplývá z matematické operace konvoluce, která se používá při práci se signály. Jedná se o matematickou operaci mezi dvěma vektory téhož argumentu.

Diskrétní konvoluce

$$x(i) * h[i] = \sum_k x[k]h[i-k] \tag{2.2.4.1}$$

2D konvoluce

$$X[i,j] * H[i,j] = \sum_k \sum_l X[k,l]H[i-k, j-l] \tag{2.2.4.2}$$

kde

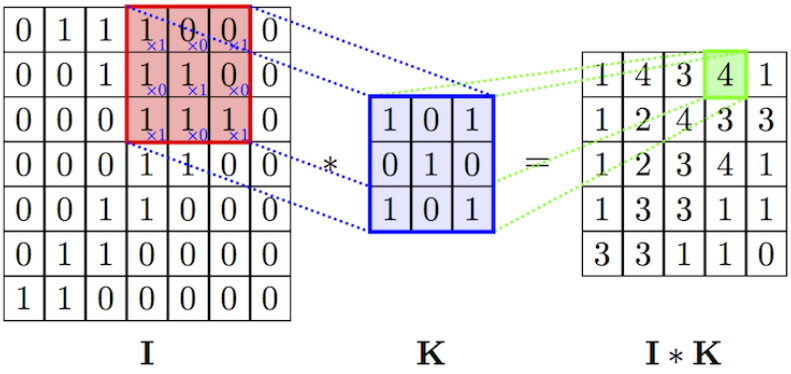
x, h, X, H	vstupní složka (signál),
i, j, k, l	indexy.

Konvoluce se hojně využívá v počítačovém vidění, což je podoblast informatiky, která se zabývá vytvářením počítačových systémů, které jsou schopny získávat informace z obrazu. Podrobnější popis práce s obrazem a počítačovém vidění lze nalézt v sekci 3.1 Zpracování obrazu.

Celá konvoluční neuronová síť se skládá z několika vrstev, kde každá vrstva má svůj smysl. Základní vrstvy jsou:

- Konvoluční vrstva *(Convolution layer)*

  Startovací vrstva sítě. Tato vrstva aplikuje množinu kernelů na vstupní data a provádí mezi nimi konvoluci. Výstup konvoluční vrstvy následně tvoří takzvaně příznakové mapy. V případě práce s obrazem je úkolem konvoluční vrstvy získat lokální informace o rastrovém obraze.

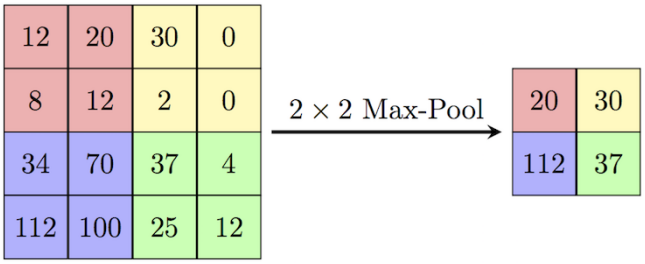


Obrázek 2.8: Příklad 2D konvoluce [28]. Hodnoty ve stejné části se společně vynásobí a výsledky násobení se společně sečtou.

- Poolovací vrstva *(Pooling layer)*

  Vrstvy konvoluční a poolovací se ze začátku většinou střídají. Dohromady pak tvoří část sítě starající se o extrakci dat. poolovací vrstva přijímá vygenerované aktivační mapy předešlých konvolučních vrstev a agreguje okolní pixely do jedné hodnoty. Díky agregaci poolovací vrstva snižuje rozměr tenzoru. Zmenšení závisí na velikosti poolovacího pole.

  Využívají se 2 typy poolovacích vrstev. *Max* a *Average*. Tyto vrstvy se liší podle způsobu jakým agregují hodnoty. Jedna vybere maximální hodnotu z prostoru a druhá počítá průměr.



Obrázek 2.9: Příklad 2D *MaxPooling* [2]. Z *pool* pole vybere maximální hodnotu a tu předá jako výslednou.

- Vyrovnávací vrstva *(Flatten layer)*

  Tato vrstva se dává na konci části extrakce dat. Přijímá výstup předchozí vrstvy a její cíl je převést vstupní vektor, reprezentovaný jako matice, na vektor, se kterým pracuje klasifikátor. Existuje více typů vrstev pro tento účel. Jednou z nich je například *Global-pooling* vrstva, která smrskne matici aktivační mapy do velikosti 1× 1. Další takovou vrstvou může být jednoduše *Flatten* vrstva, která spojí řádky matice do vektoru.

- Klasifikační vrstva *(Dense layer)*

  Spíše známá jako *Dense layer*. Zde můžeme říct, že končí část extrakce dat a začíná klasifikátor. V tuto chvíli jsme získali potřebné informace ze vstupních dat a je potřeba klasifikovat výsledek. Klasifikačních vrstev může být více za sebou a platí pro ně

stejná pravidla jako pro vícevrstvý perceptron. Vrstva si bere jako vstup výsledný tenzor z vyrovnávací vrstvy nebo výsledek předešlé klasifikační vrstvy. Tyto vrstvy se pak nadále dají dělit podle jejich aktivačních funkcí. Nejčastěji používané aktivační funkce v dnešní době je *ReLU* uvnitř sítě a *soft-max* pro konečnou vrstvu.

Rekurentní neuronová síť

Tato sekce vychází z přednášek strojového učení a rozpoznávání [24]. Výjimečnost tohoto typu neuronových sítí spočívá ve směru toku dat sítí. Oproti klasickým dopředným neuronovým sítím totiž rekurentní síť šíří data i z výstupu zpět na vstup. Princip spočívá v samovolné modifikaci neuronů uvnitř sítě pomocí zpětné vazby, kterou si síť sama poskytuje ze svých výsledků. To znamená, že rekurentní síť je schopná oproti dopředným sítím zpracovávat množinu vstupních vektorů a s každým dalším vstupním vektorem z množiny modifikuje svůj vnitřní stav, který následně vrací zpět na vstup. To znamená, že výsledek každého nového rámce se rekurzivně spočítá sloučením vstupních dat pro daný rámec s výsledkem předchozího rámce.

Síť pak buď může konvergovat ke správnému výsledku, který vyplývá z celé množiny vstupních vektorů nebo si může držet historii výsledků a po zpracování celé vstupní množiny tuto historii zobrazí.

Tento typ sítí se hojně využívá například při rozpoznávání řeči, kdy síť například poslouchá postupně každou slabiku slova a podle toho modifikuje svůj vnitřní stav a počítá pravděpodobnosti, která slabika přijde jako další. Používá se učení s učitelem i bez učitele.

Modulární neuronová síť

Jak bylo výše zmíněno v sekci 2.2 umělé neuronové sítě vznikli díky kopírování funkcionality opravdových neuronů v nervové soustavě lidí a zvířat. Výše zmíněné typy ovšem fungují jako jedna ucelená topologie, která řeší jeden typ problémů. Podle studie uvedené v [23], ale funguje mozek spíše jako soustava více menších sítí, které můžeme nazývat moduly. Každý modul řeší určitý typ problému a všechny moduly takto můžou spolupracovat a řešit komplexnější problémy, než by jedna velká síť mohla zvládnout. Právě touto myšlenkou jsou inspirovány modulární neuronové sítě, které jsou seskládány z více samostatných sítí. Tyto sítě pak spolu mohou soutěžit nebo spolupracovat a dohromady vytváří funkční systém pro řešení daného problému.

Podle dizertační práce [6] lze na modularitu neuronových sítí nahlížet dvěma pohledy. Z hlediska teorie sítí, což znamená, že se problém rozdělí do více částí, na kterých pak každá podsíť pracuje samostatně. A z hlediska obvodového a systémového, což je pohled, který už řeší jakým způsobem by se jednotlivé podsítě měli poskládat, aby bylo dosaženo maximální efektivity a přesnosti.

2.2.5 Proces učení

Učení systému si je možné představit jako proces, při kterém se ovlivňuje vnitřní stav systému, aby co nejlépe prováděl daný úkol. Tohoto je možné dosáhnout buď na začátku, když je systém sestavený a ještě se nenasazuje do provozu nebo je možné i takzvané učení za pochodu, kdy se síť učí s každým dalším vstupem. Učení za pochodu ovšem není možné vždycky. Jde o speciální případy učení bez učitele nebo posilovaného učení, kdy je žádoucí, aby se systém nepřestal adaptovat třeba i na nové typy problémů.

U neuronových sítí učení zpravidla funguje v rámci generací. Pro představu je možné si vzít příklad učení s učitelem, kdy na začátku je síti předložena trénovací množina a pomocí algoritmu *Back-propagation* se síť adaptuje na trénovací množinu a provádí nad ní takzvanou generalizaci. Cílem je provést generalizaci nad prvky v trénovací množině a přitom je nutné dát pozor na nedotrénování, ale zároveň i přetrénování.

Dynamiky neuronové sítě

Z materiálů uvedených ve zdroji [25] jsou dynamiky základní fáze pro ustanovení a provoz umělé neuronové sítě. Jedná se o organizační, adaptační a aktivní dynamiku.

1. Organizační dynamika

   Tato dynamika se stará o uspořádání topologie sítě. Ve většině případů je konstantní, ale nemusí být. Tato dynamika nemá smysl u jednovrstvých perceptronů, ale spíše u komplexnějších topologií, kde můžeme měnit strukturu sítě za účelem nalezení optimální topologie. Například přidávání a odebírání neuronu v jednotlivých vrstvách. Příkladem systému s měnící se topologií je například Kohonenova mapa uvedená ve zdroji [47].

2. Adaptační dynamika

   Představuje proces učení sítě neboli proces modifikace vnitřních vah podle funkce, které následně vstupují do aktivní dynamiky. Je založena na učení s učitelem a funguje na postupném předkládání prvků z trénovací množiny, kde prvek je dvojicí $[x, y]$ a x jsou vstupní data pro síť, jako například nahrávka nebo obraz a y je požadované označení, kterého má síť dosáhnout. S adaptační dynamikou je často spojován algoritmus *Back-propagation*, který je více rozepsán níže.

3. Aktivní dynamika

   V této chvíli se vnitřní stav sítě nachází v ustáleném stavu. Jde o případ, kdy se síť neučí, ale už pouze předvídá výsledky vstupních dat. V aktivní dynamice se tedy nastaví vstupní uzly neboli se předloží vstupní data a po ustálení aktivní dynamiky pro daný vstup se nachází ustálený výstup na konci sítě neboli výsledek, ke kterému síť dospěla.

Back-propagation

Algoritmus *Back-propagation* (tato část vychází ze zdrojů [37], [36] a [25]) je optimalizační algoritmus, který ovlivňuje váhy vstupů neuronů tak, aby se minimalizoval výsledek chybové funkce na výstupu. Ve vnitřním systému neuronové sítě se může měnit *bias* (lineární posun aktivační funkce) neuronu a váhy vstupů neuronu. Vzorec na výpočet hodnoty výstupu neuronu je definován v sekci 2.2.1.

Back-propagation vezme prvek z trénovací množiny a předloží jej na vstup sítě. Výsledkem je pak vektor číselných hodnot. Tento vektor můžeme porovnat se „ztrátovou funkcí“, která reprezentuje požadovaný vektor, kterého by měla síť dosáhnout pro daný trénovací prvek. Porovnáním jednotlivých hodnot vektorů mezi sebou je možné pro každý neuron dané vrstvy zjistit, zda je potřeba, aby jeho hodnota pro daný prvek vzrostla nebo klesla. Je ovšem nutné si pamatovat, že vektor změn z tohoto předložení je pouze pro jeden neuron. Tento proces se proto zopakuje pro všechny neurony v dané vrstvě. Tímto vznikne množina vektorů, které se sečtou. Výsledkem je pak jeden vektor o délce počtu neuronů v dané

vrstvě. Tento celý proces se opakuje pro všechny prvky v trénovací množině. Výsledkem je poté list vektorů. Ze všech vektorů se vytvoří průměrný vektor, který se může nazvat jako negativní gradient, protože udává nejrychlejší směr, kterým se minimalizuje hodnota chybové funkce.

Pro matematické rovnice je uvažován zjednodušený model sítě, která má 2 vrstvy a každá vrstva má pouze jeden neuron.

Výpočet vstupu aktivační funkce neuronu:

$$z^L = \omega^L a^{L-1} + b^L \tag{2.2.5.1}$$

kde

a^x	výstupní hodnota neuronu v x-té vrstvě,
ω	váha vstupu neuronu,
b	bias neuronu,
L	index označující vrstvu sítě (v našem případě L = 1).

Pak výstupní hodnota neuronu je:

$$a^{(L)} = \delta(z^{(L)}) \tag{2.2.5.2}$$

kde

δ	aktivační funkce neuronu.

A ztrátová funkce se vypočítá podle metody *mean square error* jako:

$$C_0 = (a^{(L)} - y)^2 \tag{2.2.5.3}$$

kde

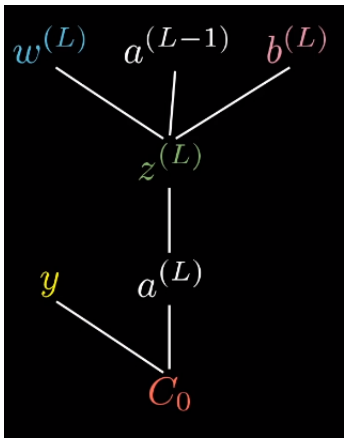
C_0	ztrátová hodnota pro jeden neuron jedné vrstvy,
y	požadovaný výstup neuronu.

Vznikne stromová struktura výpočtu. Dále je potřeba zjistit, jak je C_0 citlivé na změnu $\omega^{(L)}$. Toho lze dosáhnout pomocí parciálních derivací. Musí se ovšem postupovat postupně, protože mezi $\omega^{(L)}$ a C_0 je ve stromě několik mezikroků. Z výše vypočtených hodnot je pak možné poskládat hlavní rovnici.

Jde o řetězové pravidlo, kdy váha ω ovlivňuje vstup aktivační funkce z, který pak ovlivňuje výstup neuronu a, který ovlivňuje *Cost function* C. Hierarchie je znázorněna na obrázku 2.10. Tato funkce platí pouze pro jeden prvek z trénovací množiny. Pro všechny prvky se musí dále všechny výsledky zprůměrovat.

Kromě metody *mean square error* (MSE) se v praxi využívá pro výpočet ztrátové funkce ještě metoda *cross-entropy loss error* [31].
Výpočet ztrátové funkce pomocí *mean square error* (MSE):

$$y = E = \frac{1}{P} \sum_{p=1}^{P} \sum_{j=1}^{J} (t_{pj} - y_{pj})^2 \tag{2.2.5.8}$$



Obrázek 2.10: Struktura výpočtu [36].

Rovnice pro výpočet změny váhy:

$$\frac{\partial C_0}{\partial \omega^{(L)}} = \frac{\partial z^{(L)}}{\partial \omega^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C_0}{\partial a^{(L)}} \tag{2.2.5.4}$$

jednotlivé derivace rovnice:

$$\frac{\partial C_0}{\partial a^{(L)}} = 2(a^{(L)} - y) \tag{2.2.5.5}$$

$$\frac{\partial a^{(L)}}{\partial z^{(L)}} = \delta'(z^{(L)}) \tag{2.2.5.6}$$

$$\frac{\partial z^{(L)}}{\partial \omega^{(L)}} = a^{(L-1)} \tag{2.2.5.7}$$

kde

J	velikost vstupního vektoru,
P	počet trénovacích vzorků,
$t_{p,j}$	očekávaná výstupní hodnota,
$y_{p,j}$	skutečná výstupní hodnota sítě,

Výpočet ztrátové funkce pomocí cross-entropy loss:

$$E = \frac{1}{P} \sum_{p=1}^{P} -log \left(\frac{e^{Sp}}{\sum_{j=0}^{J} e_{jp}^{y}} \right) \tag{2.2.5.9}$$

kde

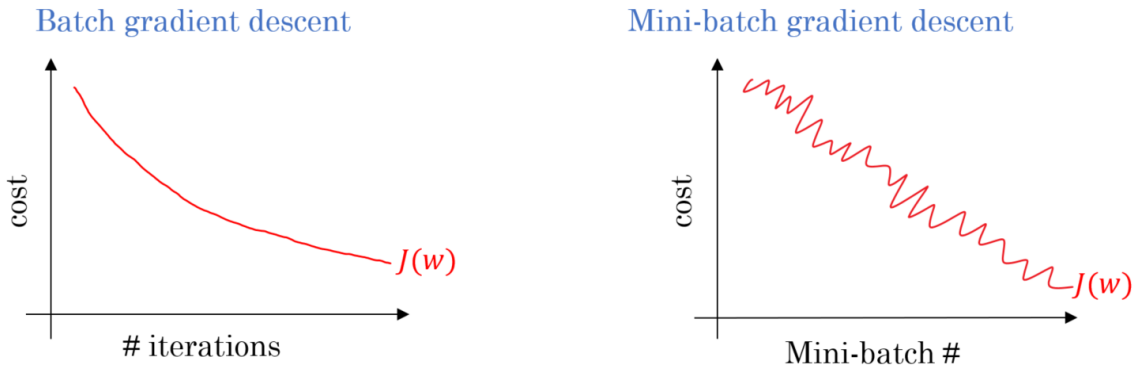
J	velikost vstupního vektoru,
P	počet trénovacích vzorků,
j	index ve vektoru o velikosti J,
s_p	požadovaná hodnota,
$y_{j,p}$	skutečná, predikovaná výstupní hodnota sítě.

Je potřeba mít na paměti, že i při nalezení minima ztrátové funkce se nemusí nutně jednat o nejlepší řešení. Důvodem jsou lokální minima, která se tváří jako globální minimum. Pro uniknutí lokálnímu minimu se může zavést určitý typ regularizace neboli zavedení šumu, který bude s hodnotami hýbat mimo gradient.

Algoritmus 1: Algoritmus *Back-propagation* [10].

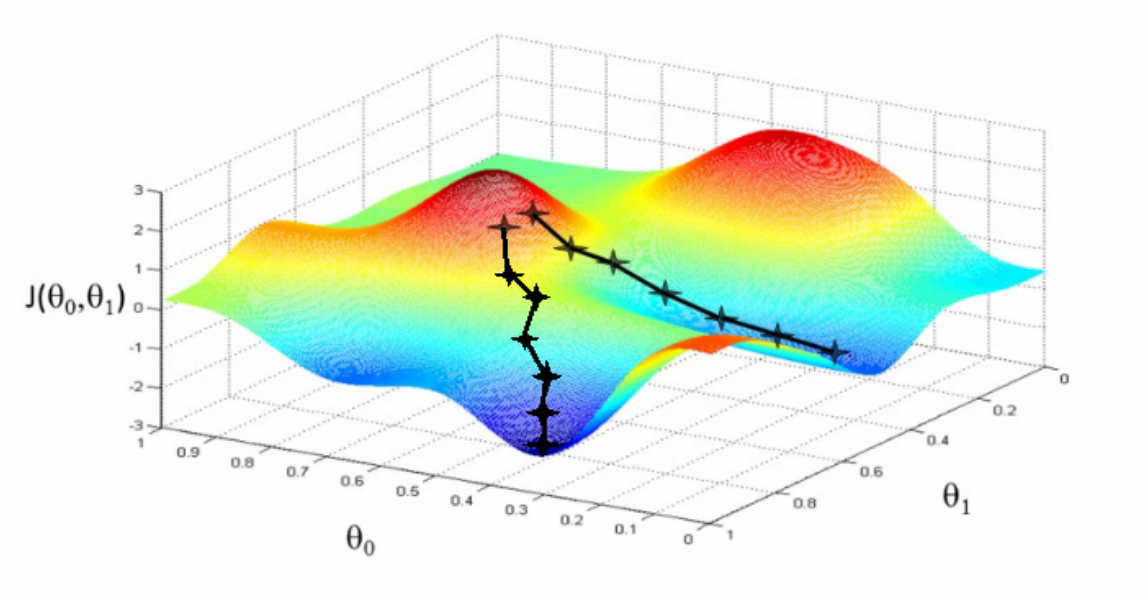
```
1 Prvotní inicializace vah
2 while není dostatečná přesnost do
3     Vyber vzorek x_p z trénovací množiny
4     Urči chybu modelu podle y_p
5     Zpropaguj chybu do předešlých vrstev
6     Uprav váhy
7 end
```

Spočítání celého gradientu je velice náročná činnost jak na čas, tak na výkon a výsledkem je spočítání dalšího kroku pomocí derivace sumy celého datasetu. Tento typ optimalizace se jmenuje *Gradient descent* neboli gradientní sestup a v praxi se příliš nevyužívá. Kromě gradientního sestupu je pak možné využít ještě *Stochastic gradient descent* nebo *Mini-batch descent*. Tyto přístupy počítají s tím, že derivace sumy je to samé jako suma derivací a tudíž nepočítají jeden krok podle všech trénovacích dat, nýbrž *Stochastic gradient descent* počítá „mini-kroky“ podle každého jednotlivého prvku z trénovací množiny a *Mini-batch descent* rozděluje dataset na menší množiny prvků, ze kterých následně vypočítává krok. V případě *Mini-batch descent* jsou podmnožiny většinou seskládány náhodně z původní množiny a při trénování na více generacích se mění. Oproti metodě *Gradient descent* je zde nevýhoda menší přesnosti gradientního kroku, která je ovšem vykompenzovaná signifikantním zrychlením trénovacího procesu. Znázornění křivky ztrátové funkce při učení pomocí *Gradient descent* a *Mini-batch descent* je na obrázku 2.11.



Obrázek 2.11: Rozdíl sestupu celého a částečného gradientu [12].

Na obrázku 2.12 lze vidět chybovou funkci a interpolace dvou množin bodů. Takhle se dá znázornit hledání minima chybové funkce. Počáteční bod obou množin značí první inicializaci vah sítě a každý další bod je úprava vnitřního stavu. Také je možné si všimnout, že se obě křivky vydaly jiným směrem. To je z důvodu, že i prvotní inicializace vah může mít velký dopad na výsledný stav. Zatím co jedna křivka našla pouze lokální minimum, druhá díky jiné počáteční pozici dokázala najít globální minimum funkce.



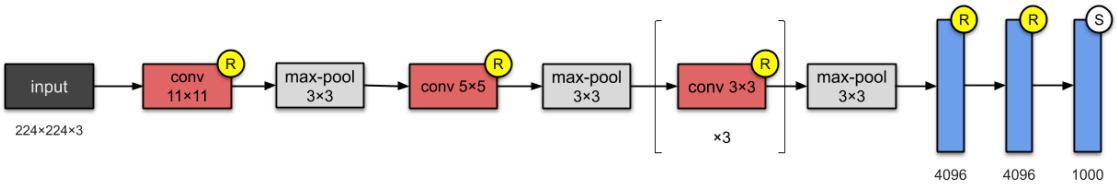
Obrázek 2.12: Hledání minima ztrátové funkce [48].

2.2.6 Vybrané modely neuronových sítí

V rámci bakalářské práce je uvedeno i lehké srovnání existujících modelů, které lze využít pro analýzu videa. Přesněji jsou zde zmíněny modely sítí AlexNet, VGGNet16 a Inception ResNet V2.

- AlexNet

  AlexNet vznikla v roce 2012 a jejím autorem byl Alex Krizhevsky. AlexNet je považována za revoluční síť, která změnila pohled na to, jak se mají vytvářet konvoluční neuronové sítě. Byla jednou z prvních sítí, která používala ReLU aktivační funkci na místo tanh a tato změna měla za následek dosažení hodnoty ztrátové funkce pod 0.25 na datasetu CIFAR-10 přibližně 6-krát rychleji než při použití staré aktivační funkce. Dalším převratným příspěvkem AlexNet bylo využití více GPU jednotek a možnost trénování větších modelů s kratším časem. Poslední zmíněnou inovací podle zdroje [49] bylo představení překrývání u *pool* vrstev. Zjistilo se, že překrývání jednotlivých kernelů má za následek snížení ztrátové hodnoty a modely bylo těžší přetrénovat.

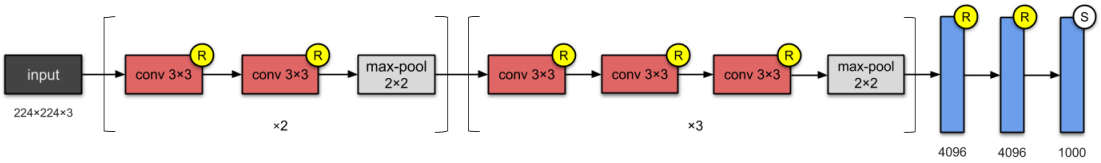


Obrázek 2.13: Architektura AlexNet modelu [17].

- VGGNet16

  Důvodem vzniku VGGNet bylo, podle zdroje [4], redukovat počet parametrů v síti a zrychlit tím trénovací čas. Hlavní myšlenkou bylo nahradit všechny velké kernely,

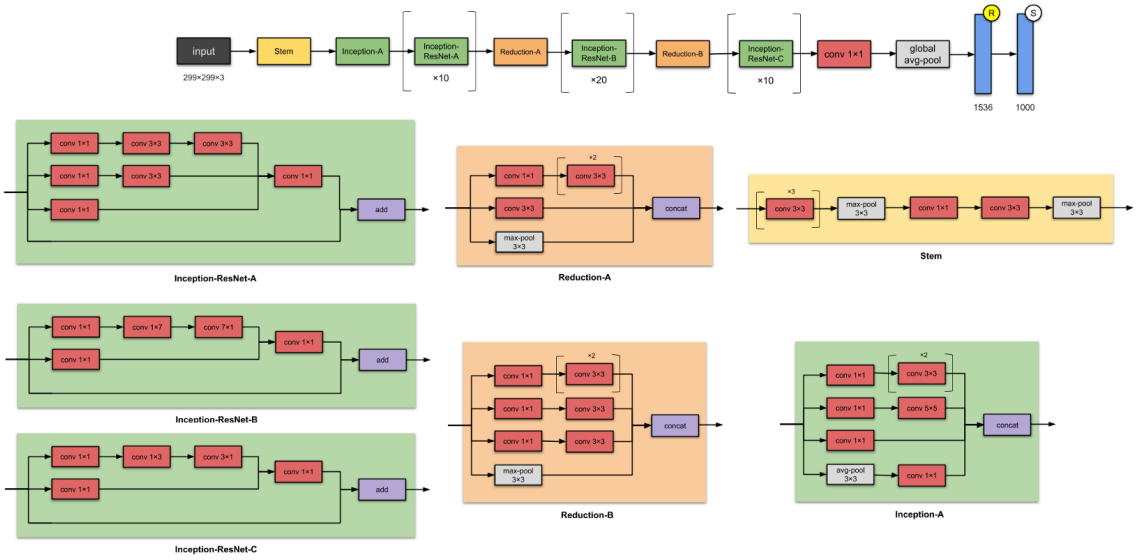
které využívá AlexNet jako například 11×11 fixní velikostí 3×3 a *pool* šířkou 2×2. Důvodem bylo zjištění, že každá konvoluční vrstva využívající velké kernely může být nahrazena posloupností více konvolučních vrstev využívajících menší kernely.



Obrázek 2.14: Architektura VGGNet16 modelu [17].

- Inception ResNet v2

  Vychází ze zdroje [41]. Inception ResNet v2 patří do rodiny ResNet a tento model je následníkem modelu Inception-v3. Vzniknul v roce 2016 a jako jediný model zmíněný v této bakalářské práce obsahuje paralelní větve vrstev. Tento model je velice komplexní a popisuje se ve stavebních modulech. Oproti Inception v3 uvedl několik změn jako přidání více Inception modulů nebo vytvoření a přidání nového typu modulu Inception-A.



Obrázek 2.15: Architektura Inception ResNet V2 modelu [17].

Kapitola 3

Zpracování a klasifikace obrazu

Základní postup klasifikace obrazu je zpravidla stejný. Je potřeba rozdělit obraz na menší části, mezi kterými se poté hledají podobnosti, podle kterých se klasifikuje. Na úvod je v této části práce popsán obraz a jak ho vidí počítač. Následně jsou zde rozepsány některé algoritmy, které se využívají při takzvaném *data augmentation* a také jsou zde zmíněny algoritmy, pomocí kterých segmentují obraz konvoluční neuronové sítě. V druhé části je poté zaměření na samotnou klasifikaci, tedy na algoritmy, které jsou schopné, pomocí dat ze segmentačních algoritmů, klasifikovat objekt v obraze. Je zde zmíněno a rozepsáno několik *Machine learning* algoritmů včetně moderních přístupů jaký nabízí *Region proposal networks* modely.

3.1 Zpracování obrazu

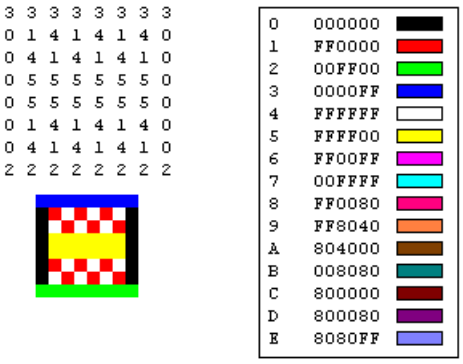
Další kapitolou této bakalářské práce je popis obrazu, práce s obrazem a možnosti extrakce dat z obrazu.

Tato sekce je zaměřená na obraz, na pohled počítače na obraz a na možnosti extrakce informací z obrazu, ale také na rozšiřování datasetů pomocí *data augmentation*. Jako první je zde jednoduchý popis obrazu. Jak vidí obraz počítač, a hlavně jak se s obrazem dá pracovat, pokud je potřeba si vytvořit vlastní dataset. Jsou zde popsané jednoduché metody, jak převést obraz do odstínů šedi, aplikace filtrů na obraz nebo modifikace obrazu pomocí transformačních matic. Také jsou zde zmíněny a rozepsány konvoluční kernely, které využívají konvoluční sítě pro extrakci dat z obrazu.

V základu existují 2 typy obrazu. Rastrový a vektorový. Rozdíl mezi těmito typy je v jejich uložení. Zatím co rastrový obrázek je uložen jako matice čísel, vektorový obraz se dá spíše představit jako posloupnost instrukcí pro počítač, jak má obraz nakreslit. Hlavní rozdíl využití těchto přístupů je při změně velikosti. Rastrový obraz při zvětšování ztrácí ostrost, zatímco vektorový obraz si ostrost zachovává při jakékoliv změně velikosti.

3.1.1 Barevné modely

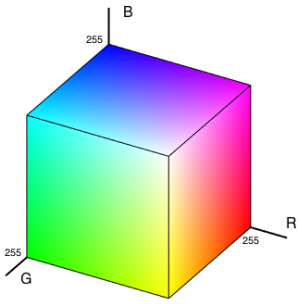
Barvy v obraze jsou většinou tvořeny třemi složkami R, G, B (*Red, Green, Blue*), kde každá složka je reprezentována hodnotou $h \in < 0, 255 >$. Smícháním těchto složek poté vzniká barva, která se zobrazuje na monitoru počítače. Barevný obraz je tedy tvořen 3-dimenzionální maticí o velikosti šířky a výšky obrazu a počtu barevných kanálů.



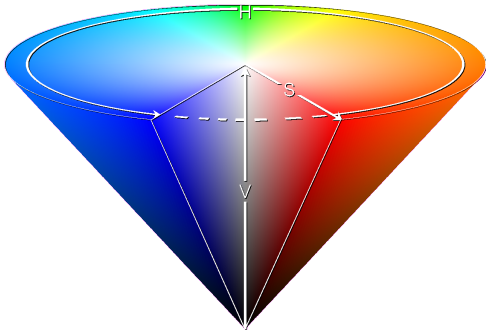
Obrázek 3.1: Způsob uložení rastrového obrázku v RGB [3].

Na obrázku 3.1 je znázorněná 2D matice obsahující barevné čtverce. Každý čtverec má pak hodnotu, kterou lze převést na hexadecimální číslo z pravého sloupce. Toto číslo se dá rozdělit po dvou znacích na 3 části neboli R, G, B.

Za zmínku pak ještě stojí barevný model H, S, V (*hue*, *saturation*, *value*). V překladu pak odstín, saturace a hodnota nebo možná lépe řečeno jas. Tento model má tvar kužele a byl vytvořen za účelem namapování základních barev do tvaru, který je intuitivnější pro uživatele. Odstín nastavuje barvu objektu a dá se představit jako otáčení kuželem podle osy y. Saturace, neboli sytost barvy, nastavuje čistotu barvy a přimíchání šedé. Jestliže výška HSV kužele značí odstíny šedé barvy, pak saturace se dá představit jako pohyb bodu barvy po kolmici k výšce. Poslední je jas. Jas nastavuje hodnotu bílé barvy a dá se realizovat jako pohyb bodu barvy v kuželu po ose y. Pomocí těchto 3 hodnot je možné nastavit kteroukoliv barvu ve škále kuželu.



Obrázek 3.2: Vizualizace RGB kostky.



Obrázek 3.3: Vizualizace HSV kuželu.

3.1.2 Práce s obrazem

Sekce práce s obrazem vychází z učebních textů [27]. V této sekci je rozebrána práce s obrazem. Konkrétně pak jak převádět obraz do tónů šedi, pokud chceme vytvářet černobílé snímky, provádění transformací obrazu pomocí matic a filtrování obrazu.

Práce s barvami

V předchozí sekci o základech obrazu je zmíněno, že barvy v obraze jsou tvořeny většinou 3 základními složkami, neboli barva jednoho pixelu je reprezentována pomocí 3 čísel. V počítačové grafice se na každou složku většinou využívá 8 bitů. Když tuto hodnotu převedeme z binární podoby do desítkové tak nám vyjde, že každá složka může mít až 256 odstínů. Dohromady pak všechny složky dávají necelých 17 milionů barev. V reálném světě je paleta barev teoreticky nekonečná. To znamená, že při pořizování fotografií nebo videí dochází ke kompresi, a tudíž ke ztrátě dat. Dalo by se tedy říct, že uložená fotografie je už pouze aproximací skutečného světa.

Při převodu obrazu na černobílý už není potřeba ukládat barevné složky, ale bude stačit pouze jedna hodnota, která bude určovat odstín šedi.

Konverze na odstíny šedi

Pro konverzi barvy na stupně šedi jsou zde uvedeny 3 rovnice, které využívají různých způsobů výpočtu [11]. Pro případ, kdy by bylo potřeba vytvořit přímo černobílý snímek, neboli snímek obsahující pouze černou a bílou barvu, se využívají metody ditheringu jako například prahování nebo distribuce chyby [27].

Prvním způsobem konverze je pro každý pixel vybrat průměrnou hodnotu jeho nejsilnější a nejslabší složky.

$$y = \frac{1}{2}(max(R, G, B) + min(R, G, B)) \tag{3.1.2.1}$$

kde

y	hodnota odstínu šedi pixelu,
R, G, B	barevné složky,
$min(), max()$	funkce vybírající maximální nebo minimální prvek z množiny.

Další metodou je jednoduchý průměr všech složek.

$$y = \frac{1}{3}(R + G + B) \tag{3.1.2.2}$$

kde

y	hodnota odstínu šedi pixelu,
R, G, B	barevné složky.

Poslední metodou je výpočet podle vztahu intenzity barevných složek.

$$y = 0,299R + 0,587G + 0,114B \tag{3.1.2.3}$$

kde

y	hodnota odstínu šedi pixelu,
R, G, B	barevné složky.

Geometrické transformace

Občas je také nutné s obrazem pracovat v jiných geometrických závislostech. K tomu využíváme speciální matice aplikované na obraz. Těmito maticemi můžeme obraz například zkosit, natočit nebo zmenšit.

V rámci této bakalářské práce jsou zmíněny pouze metody pro práci s 2D obrazem. V základu se práce těchto matic dělí na 3 typy: rotace, translace a zkosení obrazu. Tyto typy se pak nadále mohou spojovat do sebe. Kromě uvedených způsobů transformace obrazu jsou také metody aplikování matic na jednotlivé barevné složky obrazu. Tím se může například zvýraznit určitá složka nebo úplně změnit barvy v obraze.

- Rotační matice

  Rotační matice se využívá pro rotaci obrázků. Je tedy možné otočit objekt například vzhůru nohama.

$$\left[x', y', 1\right] = \left[x, y, 1\right] * \begin{bmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1.2.4}$$

  kde

  α úhel otočení obrazu.

- Translační matice

  Translační matice dokáže pohybovat s obrazem. Je tedy možné při fixní velikosti obrazu vysunout část obrazu nebo akorát posunout objekt do jiné části. Příklad translační matice může vypadat takto:

$$T = \begin{bmatrix} 1 & 0 & P_x \\ 0 & 1 & P_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1.2.5}$$

  kde

  P_x posun na ose x,
  P_y posun na ose y.

- Matice zkosení

  Matice zkosení se využívá pro zkosení obrazu. Příklad takové matice může vypadat takto:

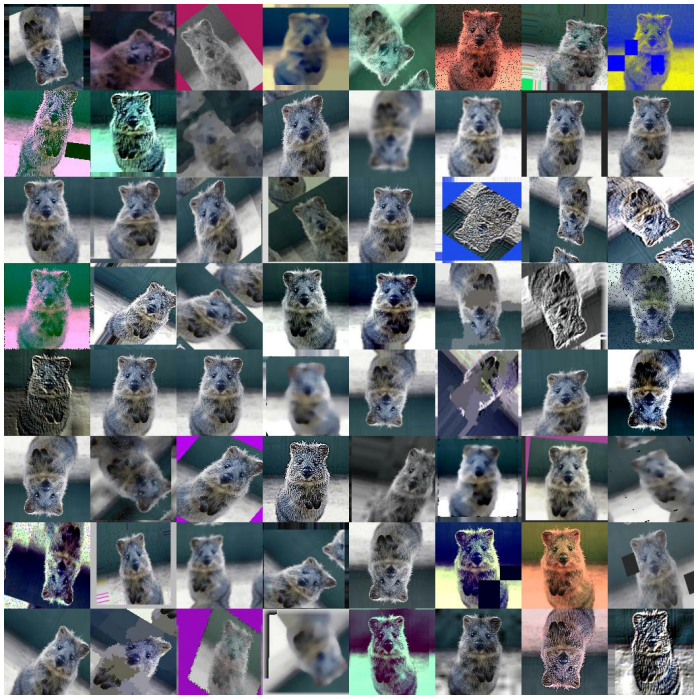
$$S_H = \begin{bmatrix} 1 & S_{hy} & 0 \\ S_{hx} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1.2.6}$$

  kde

  S_{hx} zkosení na ose x,
  S_{hy} zkosení na ose y.

Filtry a předzpracování dat

Filtrů je veliké množství a mají velkou škálu využití. Pokud je řeč o Sobelových filtrech, aplikují se tyto filtry na obraz z důvodů jeho modifikace a zvýraznění, nebo naopak zahlazení některých částí jako jsou například ostré hrany. Tyto filtry se používají pro takzvané *data augmentation* neboli předzpracování dat. Předzpracování dat je proces rozšiřování datasetu. Trénovací data (v případě Sobelových filtrů a konvolučních sítí se jedná o obrázky) jsou náhodně modifikována za účelem zvýšení diverzity datasetu a vyšší cílové přesnosti klasifikátoru. Metody využívané pro *data augmentation* jsou například: změna jasu, přiblížení/oddálení, prohození podle os x,y nebo aplikace výše zmíněných filtrů. Výsledky po aplikaci jednotlivých kernelů pak mohou vypadat jako na obrázku 3.4.



Obrázek 3.4: Modifikace obrazu pomocí aplikace filtrů. Tento proces se nazývá *data augmentation*.

Příklady konvolučních filtrů

Jak už bylo řečeno, kernelů existuje velké množství. Jako příklad základních kernelů by se dali uvést detekce hran, rozmazání obrazu, nebo naopak zaostření obrazu.

Pro práci s obrazem využívají kernely konvoluci, což je operace aplikovaná na dva vektory stejné velikosti. Prvky obou vektorů se společně vynásobí a výsledky se sečtou. Výsledkem je poté jediná hodnota vložená doprostřed prostoru, na který byl kernel aplikovaný.

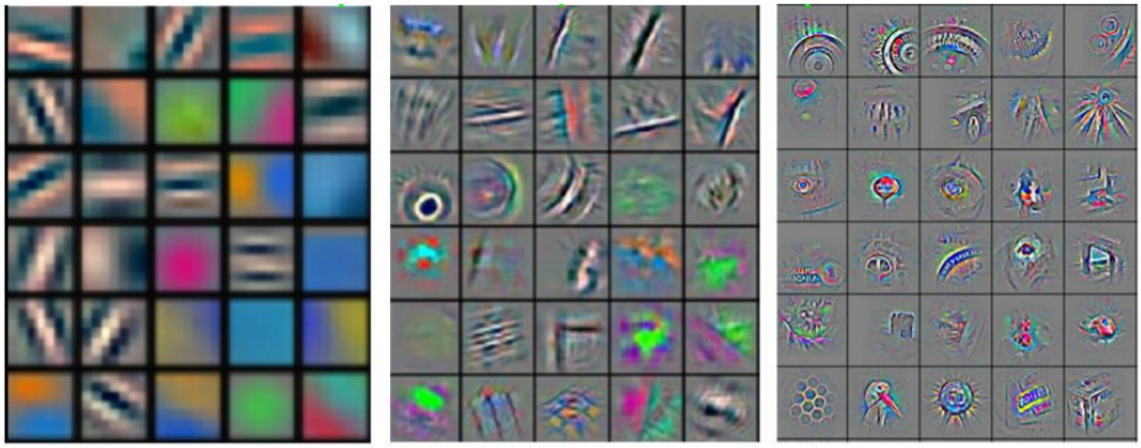
Obrázek 3.5: Ukázka aplikace Sobelových filtrů na obraz. Levý obraz je modifikací pomocí filtru na zachycení horizontálních hran, prostřední obraz je modifikován filtrem pro zachycení vertikálních hran a pravý obraz byl modifikován filtrem pro zahlazení ostrých frekvencí neboli rozostření obrazu.

Kernely v konvolučních sítích

Kernely používané v konvolučních neuronových sítích pracují obdobně. Aplikují se na snímek a modifikují ho. Nyní ovšem nejde jen o jednoduchou změnu obrazu, ale o extrakci dat. Prvotní filtry v síti jsou jednoduché a dokážou extrahovat pouze jednoduché příznaky jako jsou čáry, body a podobně. Ovšem při opakované aplikaci kernelů na modifikované obrazy je možné zachytávat stále komplexnější tvary, tak jak je tomu ukázáno na obrázku 3.6.

3.2 Klasifikace obrazu

V této sekci je zaměření na druhou část klasifikačního algoritmu a to je samotná klasifikace objektů v obraze. Jsou zde uvedeny nejen algoritmy využívající neuronové sítě, ale také jiné, použitelné metody jako je například GMM nebo *Random Forests*. Dále jsou zde rozepsány moderní metody pro klasifikaci a lokalizaci objektů v obraze a také jaké typy neuronových sítí se využívají v případě, kdy je potřeba zvýšit rychlost analýzy videa nebo je potřeba zpracovávat více videí dohromady. V neposlední řadě je v této části rozepsaná IVA neboli Inteligentní video analýza, která je sice vysoce abstraktní a sama neodhaluje



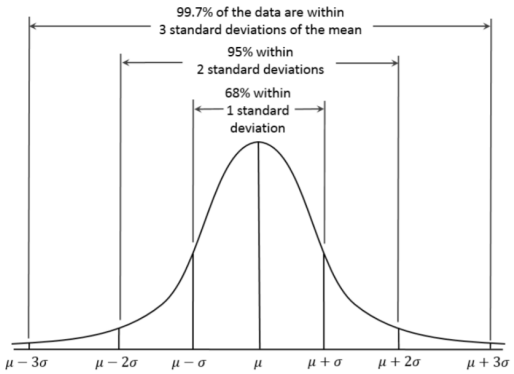
Obrázek 3.6: Mapy příznaků po aplikaci kernelů [16]. Na levé části se nachází prvotní příznakové mapy, které extrahují jednoduché příznaky, uprostřed se nachází komplexnější tvary a vpravo už je možné identifikovat některé snímky.

žádné konkrétní algoritmy pro analýzu videa. Jedná se ovšem o moderní způsob navrhování systémů, které mají za úkol analyzovat videa.

3.2.1 Gaussovské Modely (GMM)

Tato sekce vychází z přednášek strojového učení a klasifikace [24]. Jedná se o způsob trénování klasifikátoru a stejně jako u učení neuronových sítí 2.2.5 jde o iterační algoritmus. Samotný název napovídá, že budeme mísit více modelů Gaussovského rozložení do jednoho prostoru. Jednoduše řečeno se dá konstatovat, že 1 model Gaussovského rozložení bude utvářet 1 klasifikační třídu, ale v praxi se může vyskytnout i případ, kdy data jedné třídy nemají Gaussovské rozložení, a proto můžeme mít i více modelů pro jednu třídu.

Důvodem aproximace pomocí Gaussovského nebo také Normálního rozložení je, že se jedná o nejčastější rozložení prvků, které se vyskytuje v přírodě. Příkladem je možné uvést například výšku člověka ve světě nebo hodnocení zkoušky. S největší pravděpodobností budou mít tyto statistiky Gaussovské rozložení.



Obrázek 3.7: Gaussovská křivka [24].

Funkce Gaussovského rozložení

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}) \qquad (3.2.1.1)$$

kde

σ střední hodnota,
μ rozptyl rozložení.

3.2.2 Metody shlukování

Stejně jako GMM nebo neuronové sítě jsou metody shlukování trénovatelným algoritmem. Shlukování si můžeme představit jako rozdělování objektů do tříd podle jejich podobnosti. Jeden z hlavních problémů shlukování je vědět podle čeho máme shlukovat. Každý objekt můžeme rozlišovat podle více kritérií a takto můžeme dosáhnout zcela jiných výsledků [5].

Podle zdrojů [22], [20], [21] se pak tyto algoritmy dále dělí na hierarchické a ne-hierarchické. Obojí je možné dále rozdělovat. Hlavním rozdílem těchto dvou přístupů je, že při rozkládání množiny vytváří nebo naopak nevytváří hierarchické struktury podmnožin. K rozkladu je zpravidla určeno dělící kritérium. Takovýchto algoritmů je velké množství, ale jako příklad se dají uvést:

K-means

Jedná se o Ne-hierarchickou metodu založenou na těžištích jednotlivých shluků. Písmeno „K“ v názvu znamená proměnou, která značí počet průměrů *means*. Metoda potřebuje jako inicializační hodnoty množinu bodů, kterou má shlukovat, počet shluků a počáteční body shluků. Ty mohou být vybrány třeba náhodně z množiny vstupních dat. Metoda je iterační, což znamená, že opakuje svůj postup, dokud nedojde k ustálenému výsledku nebo aspoň výsledku, který je uspokojivý. Samotný algoritmus je vcelku jednoduchý.

Mezi těžišti shluků se spočítá vzdálenost. V polovině této vzdálenosti se vytvoří pomyslná kolmice, která rozdělí prostor na části. V takto rozděleném prostoru se roztřídí všechny ostatní body do daných shluků. V nově vytvořených shlucích se přepočítají těžiště a postup se opakuje od začátku. V případě, kdy mezi 2 iteracemi už není změna žádného shluku, algoritmus končí a ustálené prostory definované shluky se berou jako výsledek. Matematická reprezentace algoritmu K-means je poté: [29]

$$J = \sum_{i=1}^{k} \sum_{j=1}^{n} (||x_i - \vartheta_j||)^2 = 1 \tag{3.2.2.1}$$

kde

$||x_i - \vartheta_j||$ Euklidovská vzdálenost mezi body x_i a ϑ_j.

Mezi výhody patří:

- jednoduchost,
- jistota konvergence k výsledku,
- jednoduchá adaptace podle nových bodů.

Nevýhody pak mohou být:

- možnost odlišných výsledků v závislosti na inicializaci,
- veliký vliv okrajových bodů shluku na těžiště.

Fuzzy C-means

Tuto metodu je možné brát jako rozšíření metody *K-means* 3.2.2 o fuzzy logiku. To znamená, že pro každou část každého shluku se počítá pravděpodobnost příslušnosti do daného

shluku. Jednoduše řečeno to znamená, že body, které jsou na okraji shlukového prostoru mají menší pravděpodobnost, že do shluku patří než body, které se nachází uprostřed. Důsledkem pak můžou být body, které patří do více shluků. Výhodou tohoto rozšíření je možnost lepší klasifikace objektů, které nepatří do žádného shluku neboli do žádné třídy.

k-Nearest Neighbour

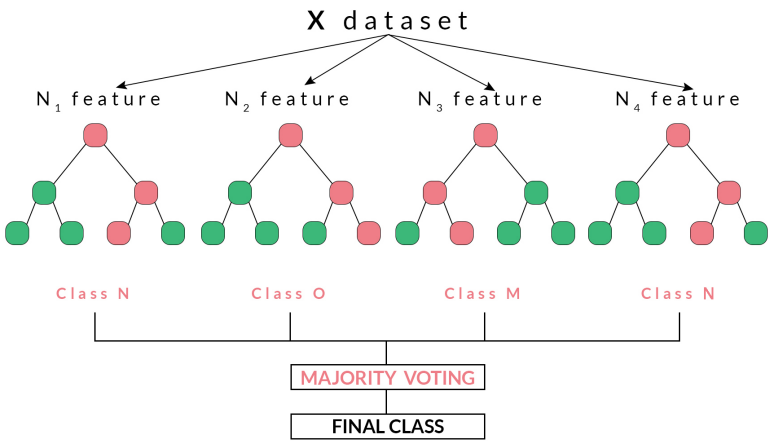
Také často známé jako k-NN algoritmus. Stejně jako u algoritmu K-means 3.2.2 se v názvu vyskytuje proměnná „K“. V tomto případě jde o k nejbližších sousedů na jejíž klasifikační třídu, nebo shluk se algoritmus dívá a podle největšího počtu přiřadí novému objektu jeho třídu. To tedy znamená, pokud například $k = 5$ pak u nově vloženého objektu se algoritmus kouká na 5 nejbližších sousedů a zaznamená si jejich klasifikační třídu. Pokud pak například 3 sousedi mají třídu A a 2 sousedi mají třídu B, pak nově vložený objekt dostane klasifikační třídu A.

3.2.3 Random forests

Random forests vychází z článku Leo Breimana [7]. Algoritmus je založený na metodě *bagging*. Jde o učící se algoritmus užívaný pro regresní a klasifikační problémy. Při trénování tohoto algoritmu se vytváří B stromy a každý jednotlivý strom je trénován bootstrap vektorem z trénovacích dat. Tohle má za příčinu lepší stabilitu systému.

Trénovací proces funguje na postupném rozkladu prostoru vstupních proměnných. Ze vstupního vektoru o velikosti p se vybere m prvků kde $(m < p)$ a s těmi se následně počítá. Podle hodnot prvků se vytváří postupně strom. Náhodný výběr proměnných snižuje nechtěnou korelaci mezi jednotlivými stromy. Takto postupně je vytvořeno L stromů. Vytvořený bootstrap vektor, který se vytváří na začátku, umožňuje duplicitní vložení prvků z datasetu, což také znamená, že některé prvky se do vektoru nedostanou. Tyto prvky se dají dále využít jako testovací data pro dané stromy.

Po natrénování se vkládají testovací data na vstup každého vytvořeného stromu a vede se statistika výstupů. Pro regresní problémy, kde má systém predikovat výsledek se počítá průměr výsledků a pro klasifikační problémy se vybere nejčastěji vybraná třída, která se určí jako finální.



Obrázek 3.8: Algoritmus Random forests [42].

Algoritmus 2: Algoritmus *Random-forests*. Proces trénování [7].

Input: Trénovací data
Output: Množina vytvořených stromů

```
while Není dostatek stromů do
    Vytvoř boostrap vektor o velikosti trénovacích dat
    for prvek in bootstrap vektor do
        Vezmi náhodný počet příznaků z prvku
        Rozděl část stromu podle příznaků
    end for
    Ulož vytvořený B-strom
end
```

3.2.4 Inteligentní video analýza (IVA)

Dále zmiňováno jako IVA. Vychází ze zdroje [46]. IVA se dá definovat jako sada algoritmů, které slouží pro automatickou analýzu videa. Jedná se převážně o zpracovávání videozáznamů. Data, které vychází z IVA jsou ukládány pomocí metadat. Tyto data obsahují nezbytné informace o objektech, které se ve videu vyskytly.

IVA má velké zastoupení v dnešním světě, zejména pak při dohlížení a ochraně majetku nebo také detekování chybných a kritických událostí. Příkladem z reálného světa je možné uvést kamerový systém v obchodech a firmách nebo také kamerový systém na ulicích. Algoritmů, které IVA využívá je velké množství, ale v základu se mohou rozdělit na adaptivní a neadaptivní.

Neadaptivní metoda je jednodušší a užívá se hlavně v prostoru se statickým pozadím (kamera se nehýbe) a nejedná se o scénu s mnoha pohybujícími se objekty. Funguje na principu výpočtu změny pozice objektu mezi snímky.

Adaptivní metoda je složitější, a tedy potřebuje i větší výpočetní výkon. Je ovšem schopna práce i s pohybující se kamerou. Principem je práce s modelem pozadí, který se počítá s každým novým snímkem. Adaptivní metody jsou spolehlivější a robustnější a mají větší využití.

IVA se dá rozdělit na centralizovanou a decentralizovanou. Centralizovaná IVA zpracovává veškerá data na serveru, zatímco decentralizovaná IVA může provádět dané úkony přímo v zařízení, které pořizuje videozáznam.

- Centralizovaná IVA

  Jak už bylo řečeno, centralizovaná IVA provádí všechny úkony na serveru. Výhodou tohoto přístupu pak může být větší výpočetní výkon a díky tomu možnost rychlejší nebo komplexnější práce s videozáznamem. Server může provádět analýzu videa i v reálném čase, proto je ovšem potřeba aby zařízení, která vysílají obraz měla pořád možnost komunikace se serverem. To může způsobit několik potíží. Hlavním problémem může být nedostatečně velké přenosové pásmo a také omezení výpočetního výkonu serveru při přijímání většího počtu vysílacích zařízení. To znamená, že pokud připojíme více kamer k serveru, server nemusí mít dostatečný výkon na provádění analýzy nad všemi kamerami anebo nemusí být ani schopen přijmout všechny data z kamer. Z bezpečnostního hlediska pak může být problémem samotná centralizace,

kdy stačí aby útočník napadl server a důsledkem je ztráta veškeré informace o všech připojených zařízeních. Tyto problémy částečně řeší decentralizace celého systému.

- Dentralizovaná IVA

  Jde o přístup, kdy analýzu obrazu provádí samotné zařízení, které obraz pořizuje. To sebou samozřejmě může nést omezení výpočetního výkonu. Výhody takového přístupu jsou pak větší bezpečnost, a i menší náchylnost k poruše. Další nespornou výhodou je absence nutnosti posílat video pro analýzu na server v reálném čase. U decentralizovaného systému není nutná přítomnost serveru. Jedná se o takzvaný *edge-based* přenos. Koncové zařízení jednoduše provádí analýzu obrazu a data z této analýzy jsou následně zakódována do videa pomocí metadat. V případě, kdy analýza zjistí kritickou situaci (pohyb člověka v zakázané zóně, chybné chování pozorovaného stroje, ...) může koncové zařízení informovat o této události server, pokud server existuje, nebo samo vykonat akci jako například signalizovat tuto situaci na výstupu kamery.

3.2.5 Neuronové sítě pro lokalizaci a klasifikaci objektů

Spíše známé jako *Region proposals networks* (RPN) je moderní řešení analýzy snímků a videí, které vychází z práce s konvolučními neuronovými sítěmi. Popis RPN vychází ze zdroje [19]. Základní myšlenka tohoto přístupu rozšiřuje možnosti konvolučních sítí tím, že model není pouze schopen říct co se na obrázku nachází, ale také kde. Veliká výhoda tohoto přístupu je pak také možnost klasifikace více objektů v jednom snímku.

Asi nejjednodušším přístupem je rozdělit obraz na předem stanovené části a tyto části následně procházet pomocí takzvaného *sliding window*, což je definované okno, které se posouvá v obraze a každá, takto vybraná část obrazu se analyzuje samostatně. Tento přístup je ovšem zbytečně výpočetně namáhavý z důvodu, že síť musí analyzovat i ty části obrazu, kde se zcela určitě nenachází žádný objekt. Navíc vyprodukované čtverce nebudou označovat objekt ve snímku, ale pouze okno *sliding window*, ve kterém se objekt nachází.

Kvůli tomuto problému vznikl přístup, který se nazývá *Region-Convolution neural network* (R-CNN), který se snaží vybrat pouze některé části obrazu pro klasifikaci a ušetřit tedy výpočetní čas analýzy.

Vybrané architektury Region proposals networks

- R-CNN

  R-CNN vychází ze zdroje [15]. R-CNN využívá takzvaného segmentačního algoritmu, který dokáže snímek rozdělit do více podkategorií, reprezentující objekty ve videu. Výsledek takového segmentačního algoritmu pak může vypadat jako na obrázku 3.9.

  Kolem těchto objektů se následně vytvoří rámce, které se následně analyzují. Pro představu se může jednat i o 2000 rámců a klasifikací v jediném snímku. Výhodou tohoto přístupu je, že klasifikační algoritmus nebude pouze zachycovat objekty čtvercových podob, ale je možné zachytit objekty veškerých tvarů a velikostí. Nevýhodou pak byla samotná rychlost algoritmu, která je stále nedostačující.

- Fast R-CNN

Obrázek 3.9: Výstup segmentačního algoritmu pro detekci objektů ve snímku.

Vylepšením předchozí metody se stala, v roce 2015 metoda *Fast R-CNN*, uvedená ve zdroji [14]. Předchozí metoda R-CNN využila segmentačního algoritmu pro nalezení potencionálních objektů, orámovala je a následně pro každý rámec použila konvoluční neuronovou síť pro identifikaci objektu. Metoda *Fast R-CNN* tento proces, dalo by se říct, otočila. Po aplikaci segmentačního algoritmu se na celý obraz aplikuje konvoluční neuronová síť, výsledkem je poté modifikovaný obraz. Do tohoto obrazu se vepíší rámce ze segmentačního algoritmu a části v jednotlivých rámcích jsou pak předkládány klasifikátoru pro označení.

Podle výsledků v článku [34] se ukázalo, že evaluační proces *Fast R-CNN* byl až 25krát rychlejší a dosahoval stejných a vyšších přesností.

- Faster R-CNN

  Hlavní výhodou *Faster-R-CNN* je možnost vypustit segmentační algoritmus, který byl potřeba v předchozích metodách. *Faster R-CNN* využívá pro segmentaci obrazu a vytváření rámců samotnou konvoluční neuronovou síť.

  Možnost vynechání segmentačního algoritmu měla velký přínos pro rychlost evaluace snímku a podle článku [34] dokázala metoda *Faster R-CNN* se srovnatelnou přesností urychlit proces evaluace 10krát oproti *Fast R-CNN* a 250krát oproti prvotní *R-CNN*.

- Yolo network

  Tato podsekce vychází ze zdroje [33]. Předchozí metody jako R-CNN nebo *Fast R-CNN* mají jedno společné, a to, že musí daný obraz zpracovat 2krát. Jednou pro vytvoření rámců kolem objektů a podruhé pro klasifikaci daných rámců. *YOLO* v názvu *YOLO network* je akronym pro *You only look once* neboli „Podíváš se pouze jednou“.

  Myšlenka za funkcionalitou *YOLO network* je rozdělit obraz na několik pod-částí a tyto části klasifikovat samostatně. V případě, kdy algoritmus nalezne v dané pod-

části objekt, vypočítá jeho střed, šířku a výšku. Pro každou část obrazu pak tedy bude výstupní vektor ve složení:

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ \vec{c} \end{bmatrix} \tag{3.2.5.1}$$

kde

p_c	značení, zda se v pod-části nachází objekt,
b_x	souřadnice x středu objektu,
b_y	souřadnice y středu objektu,
b_h	výška rámce,
b_w	šířka rámce,
$\vec{c}$	pravděpodobnosti jednotlivých tříd.

To znamená v případě, kdy existují 3 klasifikační třídy a obraz se rozdělí na 5×5 pod-částí, pak výstupní tenzor sítě bude 3-dimenzionální t=(5+3)×5×5.

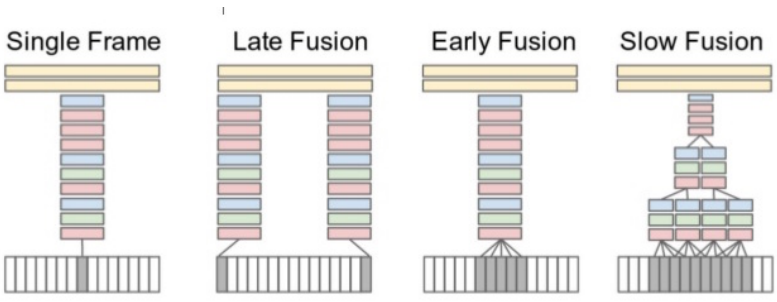
3.2.6 Typy modelů sítí pro analýzu videa

Co se týká hlubokých neuronových sítí určených pro analýzu videozáznamu, je možné zvolit hned několik přístupů [35]. Tyto přístupy se liší jednak v přesnosti klasifikace, ale také v rychlosti analýzy. V základu se dají tyto metody rozdělit na *Single stream network* a *Two stream network*.

Single stream network

Single stream network jsou typy sítí, které se specializují na agregaci informace z více sousedních snímků ve videu nebo také agregaci snímků z více kamer.

Single stream networks se dají rozdělit celkově do 4 kategorií, kde každá agreguje informace ze snímků jiným způsobem. Vizualizace těchto sítí je zobrazena na obrázku 3.10.



Obrázek 3.10: Typy *Single stream network*.

V článku Vídeňské technické univerzity [38] porovnávali přístup *Early* a *Late Fusion* při detekci pádu člověka pomocí 4 kamer z různých úhlů pohledu. Pro agregaci informací ze 4

modelů, zpracovávajících obraz ve stejný čas využili fuzzy logiku a výsledky ukázali, že oba přístupy jsou srovnatelné a dosahují příznivých výsledků. Při rychlosti analýzy 5 snímků za sekundu byl schopen Late Fusion detekovat správně pád v 87,6 % a *Early Fusion* dokonce v 93,5 % případů.

- Single Frame

  Nejjednodušším způsobem je evaluace každého snímku samostatně. Jde o metodu, kdy se z videa postupně načítají jednotlivé snímky, které se poté vyhodnocují. Tato metoda se nazývá *Single Frame* a jedná se o nejjednodušší metodu. Problém této metody může být pomalá analýza, ale také takzvaný *prediction flickering effect*, což je efekt, který způsobuje náhlou změnu predikce třídy, trvající pouze pár snímků.

  Problém s *prediction flickering effect* se dá vyřešit, pokud budeme na video nahlížet jako na skupinu snímků, kde sousední snímky mají podobný obsah a kontext. Takto na video nahlíží klasifikace *Single Frame* s rozšířením o *Rolling prediction averaging.*

  Rolling prediction averaging je jednoduchým řešením *prediction flickering* efektu pomocí implementace fronty s výpočtem průměru predikovaných výsledků neboli metoda zvaná jako *Rolling prediction averaging* [35]. Predikce snímku vyprodukovaná modelem nebude brána jako finální výsledek, ale od vektoru predikcí se odečte vektor průměrů x předešlých predikcí a z výsledného vektoru se vybere nejpravděpodobnější třída. Toto řešení se hodí při analýze videí, které jsou zaměřené na jeden kontext a neočekávají se veliké změny v obraze. V případě potřeby vyhledávání objektů pouze v krátké části videozáznamu však může mít tato metoda i nepříznivý efekt.

  Rolling prediction averaging nemusí být implementováno frontou. Fronta je pouze nejjednodušší způsob implementace této myšlenky. V praxi se často společně s konvolučními modely využívají takzvané *Long Short-term memory* modely (LSTM), což jsou speciální rekurentní typy modelů dedikované pro zpracovávání sekvencí dat.

- Late Fusion

  Late Fusion přístup využívá více modelů se sdílenými parametry, kde každá síť vyhodnocuje samostatný snímek. Tento typ se může dá využít například při snímání stejného objektu více kamerami. Predikce jednotlivých snímků se poté zprůměrují a výsledkem je jediný vektor.

- Early Fusion

  Metoda *Early Fusion*, stejně jako *Late Fusion*, zpracovává více snímků současně. Ovšem Early Fusion nezpracovává každý snímek vlastní sítí, nýbrž spojuje větší množství snímků do sebe hned v první vrstvě sítě. Stejně jako u *Late Fusion* je možné využít tuto metodu při snímání objektu z více kamer.

- Slow Fusion

  Nejvíce komplexní metoda z výše uvedených kombinuje přístup *Early* a *Late Fusion.* Více modelů si vybere větší počet snímků z videozáznamu. Postupným stromovým efektem se pak výsledky těchto modelů spojují a postupují do vyšších úrovní, kde jsou

zpracovány dalšími modely. Výsledkem je poté vektor nejvýše postaveného modelu ve stromové struktuře.

Two Stream Networks

Single stream networks jsou užitečné a vcelku kvalitní metody, mají ovšem problémy s adaptací pohybu ve videu [35]. S řešením tohoto problému přišli autoři Karen Simonyan a Andrew Zisserman v roce 2014 při uvedení *Two Stream Networks* [40]. Inovace tohoto řešení spočívala v implementaci 2 rozdílných modelů sítí. Jeden model byl dedikován pro prostorový kontext a druhý model byl určen pro kontext pohybový. Data obou modelů při analýze se pak zpracovávali pomocí *Support Vector Machine* (SVM).

Problém nastává při trénování *Two stream network* systému. Z důvodu používání 2 odlišných modelů, kde každý model analyzuje jiný snímek, se často objevoval problém špatného označování snímků při trénování. Dalším problémem je vypočítání *optical flow* vektorů pro SVM a jejich uložení. Celkově se jednalo o takzvaný *end-to-end* trénovací proces, který bývá zpravidla velice náročný.

Kapitola 4

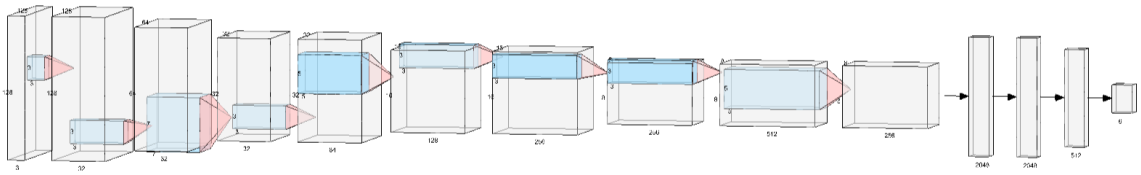
Návrh a implementace

Podstatou praktické části této bakalářské práce bylo navrhnout a implementovat klasifikátor, který bude moct být využit při analýze videozáznamů. Pro toto použití byl vybrán, navrhnut a implementován model konvoluční neuronové sítě. Také bylo potřeba navrhnout a implementovat uživatelské rozhraní, specializované pro analýzu videí pomocí vybraných klasifikátorů. Uživatelské rozhraní mělo umožňovat kromě analýzy videa také možnost změny klasifikačního algoritmu a možnost trénování modelů sítě na vlastním datasetu.

4.1 Návrh klasifikačního modelu

V rámci této bakalářské práce byly vybrány pouze typy modelů pro *Single Frame* přístup. a to konkrétně vlastní, implementovaný model, AlexNet, VGGNet-16 a Inception-resnet-v2.

Architektura vlastního modelu byla inspirovaná modelem VGGnet16, který je v sekci 2.2.6. Model byl vytvořen dvakrát, a to jednou s absencí Dropout vrstvy pro porovnání trénovacího procesu. Obrázek 4.1 vyobrazuje architekturu modelu.



Obrázek 4.1: Architektura vytvořeného modelu. Model byl inspirován architekturou VGG-net.

Model se skládá z celkových 24 vrstev, včetně *Pooling*, *Dropout* a *Batch-normalization* vrstev. Konvoluční vrstvy s velikostí kernelu větší, než 3×3 byly poskládány z více vrstev s velikostí kernelu 3×3. Jako *pool* vrstvy se využívá *MaxPooling* a jako konečná *pool* vrstva se využívá *GlobalAveragePooling*.

Dropout vrstva, přítomná v jednom z modelů, má podobnou architekturu jako *Dense* vrstva, ovšem její neurony jsou netrénovatelné a slouží hlavně k regularizaci a předcházení přetrénování modelu. Toho dosahuje pomocí náhodného „vypínání“ neuronů v předem daném poměru, v tomto případě 0,4.

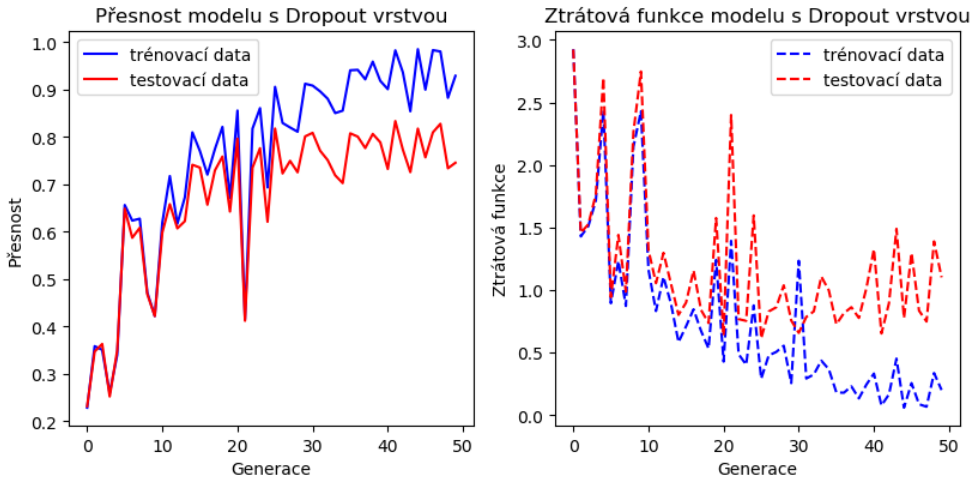
Část klasifikátoru je tvořen celkově 5 vrstvami. Ze začátku jsou vloženy 2 *Dense* vrstvy, každá o počtu 2048 neuronů, následuje menší *Dense* vrstva o velikosti 512 neuronů. Všechny tyto vrstvy využívají aktivační funkci ReLU.

Před poslední klasifikační vrstvou se nachází ještě *Batch-Normalization* vrstva, která se stará o normalizaci napříč „mini-batchemi“. Zajišťuje, aby aktivace měly průměr 0 a standardní odchylku 1. Vrstva si ukládá statistiky, a tedy obsahuje netrénovatelné parametry. Díky *Batch-Normalization* vrstvě se nemusí dbát takový důraz na prvotní inicializaci vah sítě a je možné nastavit větší *lerning-rate*.

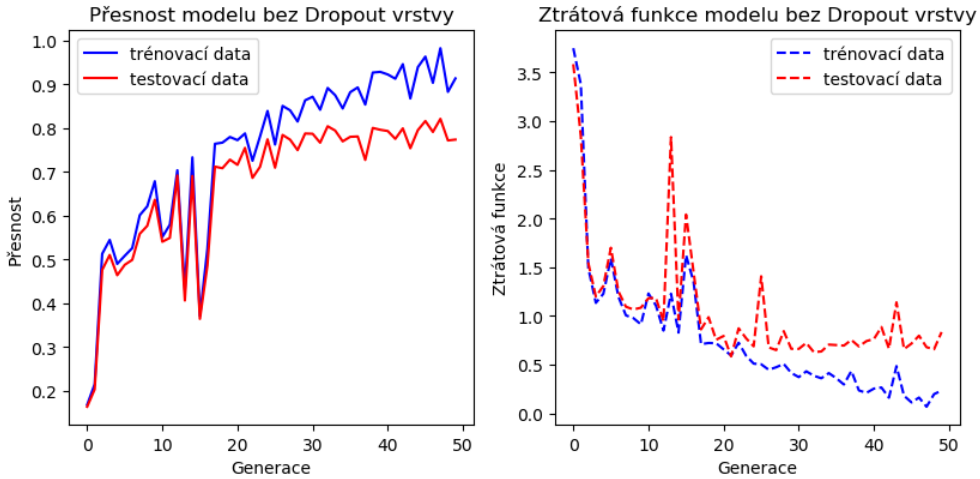
Poslední vrstvou klasifikátoru je *Dense* vrstva o velikosti 6 neuronů, které reprezentují klasifikační třídy modelu. Neurony v této vrstvě obsahují aktivační funkci soft-max, která převádí vstupní hodnotu na takzvaně posteriorní pravděpodobnost dané třídy.

Pro modely byl použit optimalizér „adam“, který používá adaptivní *learning rate*, akceleraci a momentum pro zvýšení rychlosti procesu učení.

Po navržení klasifikačních modelů následoval trénovací proces, během kterého se zároveň prováděla validace z důvodu ověření výkonnosti modelů. Na obrázcích 4.2 a 4.3 je možné vidět výsledky trénování vlastních modelů. Teoreticky by *Dropout* vrstva měla zajistit větší regularizaci a předejít přetrénování modelu a tím zajistit větší přesnost na testovacích datech. Ovšem podle těchto výsledků *Dropout* vrstva sice zajistila větší regularizaci, což lze vidět podle větší oscilace křivky pro model s vrstvou, ovšem výsledky po 50 generacích byli téměř totožné. S přesností na trénovacích datech byla, s rozdílem pouhých 1,6 % úspěšnější síť s *Dropout* vrstvou a na testovacích datech byl úspěšnější model bez *Dropout* vrstvy o 2,7 %.



Obrázek 4.2: Výsledek trénování modelu s Dropout vrstvou. Po 50 natrénovaných generacích byl výsledek na trénovacích datech 0,929 a na validačních datech 0,745.



Obrázek 4.3: Výsledek trénování modelu bez Dropout vrstvy. Po 50 natrénovaných generacích byl výsledek na trénovacích datech 0,912 a na validačních datech 0,773.

Trénování modelů

Pro trénování modelů byla využita platforma Google cloud platform [1], což je platforma řízená společností Google. Platforma poskytuje velkou škálu služeb pro své uživatele. Od Cloudových uložišť, přes možnost hostingu stránek a informačních systémů až po pronajmutí výpočetní jednotky pro náročné výpočty.

Další platformou pro trénování bylo Metacentrum[2], což je organizace provozovaná v České republice, poskytující studentům a akademickým pracovníkům možnost zdarma využívat výpočetní a úložné kapacity jejich serverů.

[1] https://cloud.google.com
[2] https://metavo.metacentrum.cz

Modely byli na platformách trénovány na počet 50 generací a experimenty s modely jsou uvedeny v sekci 5.

Modely sítí v této bakalářské práci byli trénovány na datasetu vytvořeném Anubhavem Maitym [3], který obsahuje 22 klasifikačních tříd a je zaměřený na rozpoznávání sportovních aktivit ve videu. Z tohoto datasetu se vybralo 6 tříd a na tyto třídy byly poté modely adaptovány. Klasifikační třídy jsou:

- boxing,
- football,
- formula1,
- ice-hockey,
- tennis,
- swimming.

4.2 Návrh uživatelského rozhraní

Součástí bakalářské práce bylo navrhnout a implementovat uživatelské rozhraní, které bude umožňovat vkládání a vybírání mezi klasifikačními modely, výběr části obrazu, kterou je třeba analyzovat a možnost trénování modelů na vlastním datasetu. Toto rozhraní bylo navrženo jako desktopová aplikace. Aplikace je zaměřená především na *Single Frame* modely a umožňuje využití *Rolling averaging prediction* pomocí implementované fronty. Aplikace kromě analýzy videozáznamu má možnost připojení na standardní kamerový vstup zařízení. Je tedy možné, pomocí importovaných modelů provádět analýzu videa v reálném čase.

4.2.1 Použité technologie

Pro implementaci jsem si vybral skriptovací jazyk Python verze 3.7.3. Hlavním důvodem byla přítomnost knihoven třetích stran, multiplatformnost a osobní zkušenost s jazykem. Kromě toho bylo ještě potřeba využít moduly, které se nenachází v základní verzi jazyku Python.

Knihovna OpenCV

Popis knihoven OpenCV je vytažen z oficiálních stránek a dokumentace [30]. OpenCV je volně dostupná open-source knihovna s BSD licencí zaměřující se na počítačové vidění. OpenCV má 2 funkční rozhraní. OpenCV 1.x, které je určeno pro programování v jazyce C a OpenCV 2.x, podporující objektové programování určeno pro jazyk C++.

OpenCV je modulární a obsahuje tyto moduly.

- core - Definující základní datové struktury, využívající se v ostatních modulech.
- imgproc - Pro práci s obrazem. Obsahuje filtry a operace pro stupňování, rotaci a další operace s obrazem.

[3] https://github.com/anubhavmaity/Sports-Type-Classifier/blob/master/readme_images/sports.png

- video - Pro práci s videem.
- calib3d - Obsahující algoritmy pro práci s 3D obrazem.
- features2d - Obsahuje detektory prvků v obraze jako například hrany, deskriptory a jejich porovnávače.
- objdetect - Algoritmy pro detekci objektů a instancí předdefinovaných tříd.
- highgui - Knihovny pro jednoduché uživatelské rozhraní.
- gpu - Pro akceleraci algoritmů na GPU.

Framework Tensorflow

Vychází z oficiálních stránek Tensorflow [43]. Tensorflow je volně dostupný *open-source* framework vytvořený a udržovaný společností Google. Tensorflow se především zaměřuje na strojové učení a stavbu modelů neuronových sítí. Je optimalizované na CPU, GPU a TPU *(tensor procecessing unit)* což je speciálně navržený čip pro náročné více-dimenzionální výpočty a neuronové sítě. Tensorflow poskytuje podporu pro velkou škálu programovacích jazyků jako jsou Java, Go, C++ nebo Python. Tensorflow obsahuje nepřeberné množství modulů starající se o datové typy, struktury, matematické modely, algoritmy, práci s neuronovými sítěmi a další.

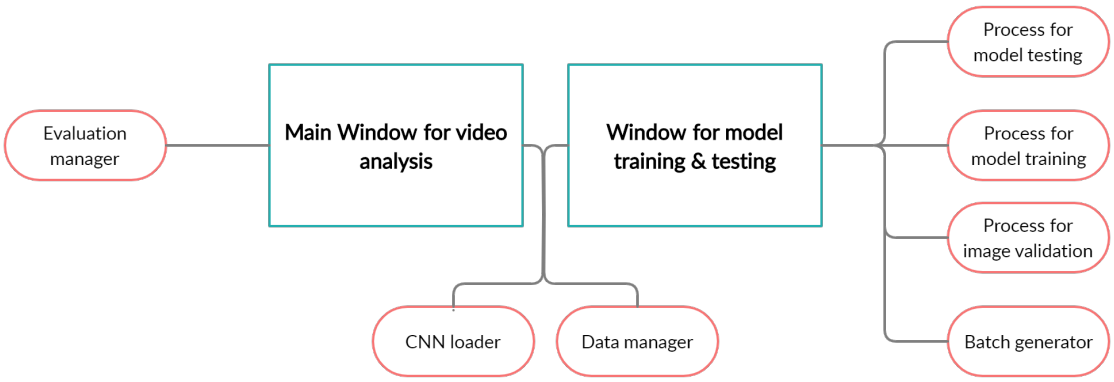
Framework PyQt5

Popis PyQt5 je vytažen z oficiální dokumentace Qt [32] a PyQt [44]. Qt je multiplatformní framework pro vývoj uživatelských rozhraní na počítače, mobily a vestavěné systémy. Je navržený jako framework rozšiřující programovací jazyk C++ a výsledný kód je před samotným přeložením zpracován pomocí tzv. MOC jednotky, která Qt kód převede na nativní C++ kód. Tento kód je tedy možné potom přeložit jakýmkoliv standardním překladačem jazyku C++.

PyQt5 je nástavba nad Qt frameworkem pro jazyk Python. Jedná se tedy o jednoduché volání C++ Qt v jazyce Python.

4.2.2 Návrh uživatelského rozhraní

Aplikace byla navržena pro práci se *Single Frame* modely, které jsou více rozepsány v sekci 3.2.6. Dále je aplikace určena pro „jednoduché“ modely konvolučních neuronových sítí, které mají jako vstupní tenzor RGB obraz a jako výstup je vektor značící klasifikační třídy. Nejedná se tedy o RPN modely. Aplikace slouží jako nástroj pro využití konvolučních neuronových sítí pro analýzu videa a také pro trénování těchto sítí na vlastním obrázkovém datasetu, a to i s použitím předzpracování dat. Jednoduchý návrh aplikace je vyobrazen na obrázku 4.4



Obrázek 4.4: Diagram návrhu aplikace.

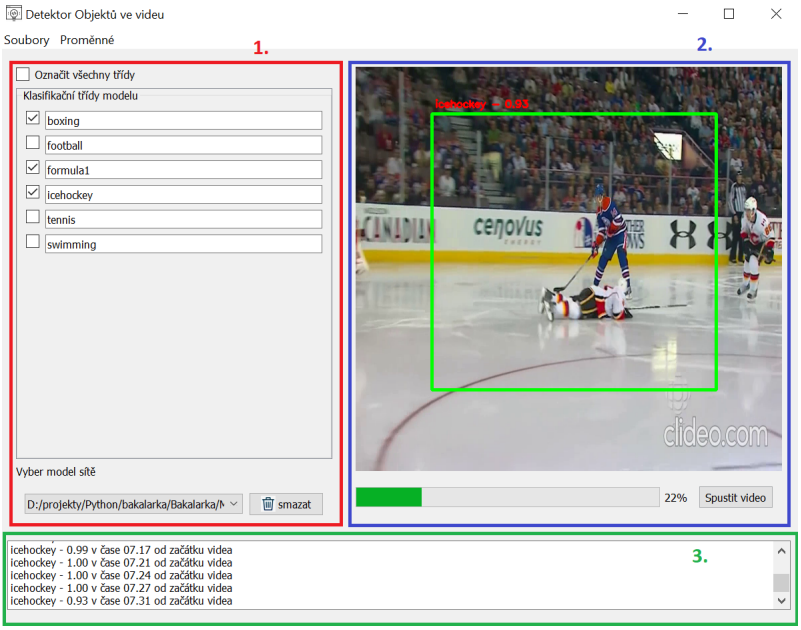
Videopřehrávač a prostředí pro analýzu videa

Hlavní okno aplikace slouží pro samotnou analýzu videa a pro práci s modely. Dá se rozdělit do 3 hlavních částí vyobrazených na obrázku 4.5. První část je tabulka pro pojmenovávání klasifikačních tříd modelu a vybírání hledaných tříd. Druhou částí je pak samotný videopřehrávač, kde se přehrávají modifikované snímky z videa. Poslední částí je tabulka záznamů nalezení objektu ve videu.

Hlavní okno pracuje s vlastně vytvořeným souborem `models_path.csv` kde si ukládá jednak adresy importovaných modelů a zároveň i jména jednotlivých tříd. Díky těmto záznamům jsou změny jako: importování modelu nebo změna názvu třídy implementovány jako perzistentní a uživatel je nemusí zadávat při každém spuštění aplikace.

Aplikace začíná analýzu videa při jeho vložení a skončí buď po přehrání celého videa anebo ukončením pomocí klávesy „q“.

Aplikace nemá žádný vložený filtr pro výběr formátu video-souboru. Kompatibilita tedy závisí pouze na podporovaných formátech knihovny OpenCV a přítomných kodecích na hostovaném zařízení.



Obrázek 4.5: Analýza videa v prostředí uživatelského rozhraní s vybraným modelem a hledanými klasifikačními třídami.

- Výběru modelu a klasifikačních tříd

  Na obrázku je označena červenou barvou jako část 1. Obsahuje především dvě komponenty. okno výběru cesty k modelu a tabulku, obsahující zaškrtávací a textové pole pro výběr hledaných tříd a změny názvů jednotlivých tříd. Pro importování modelu se dá využít horní lišta aplikace, konkrétně pak menu **Soubory**, anebo klávesová zkratka `ctrl+i` .

  Po importování modelu se zkontroluje validita vybraného souboru a duplicitní výskyt v souboru `models_path.csv`. Pokud je model validní a ještě nebyl importován, zjistí se počet výstupních tříd modelu a provede se zápis do souboru. Cesta k modelu se vloží do výběrového okna *ComboBoxu*. V případě, kdy model není validní se tento problém ohlásí uživateli a model není zapsán do souboru.

  Názvy tříd nového modelu jsou implicitně pojmenovány indexy. Změna názvu se poté provádí pomocí modifikace textu v textových polích. Změna názvu třídy je propsána do souboru ihned po dokončení modifikace.

- Videopřehrávač

  Videopřehrávač je na obrázku 4.5 označen modrým obdélníkem jako 2. část. Účel videopřehrávače je ulehčení práce s videem pro uživatele, díky tomu, že uživatel vidí v reálném čase, která část videa se právě analyzuje, může jednoduše vybírat část obrazu určenou k analýze a také může video pozastavit, případně přetočit.

  Výběr analyzované části videa je implementován pomocí klikatelné obrazovky videopřehrávače, kdy tažením myši je možné určit analyzovaný obdélník. V případě potřeby analýzy celého obrazu stačí vytvořit obdélník o nulovém obsahu neboli například pouze kliknout. Analyzovaná část je vyobrazena pomocí obdélníku, který se kreslí do snímků ve videu. Tento obdélník může být buď červený, v případě, kdy

nebyla nalezena žádná klasifikační třída, nebo zelený v případě nálezu. V případě nalezení klasifikační třídy se k obdélníku ještě také dopíše posteriorní pravděpodobnost a název nalezené třídy.

Pozastavení videa jde dvěma způsoby. Prvním způsobem je dvojklik na obrazovku videopřehrávače. Tento způsob má ovšem komplikaci automatického nastavení analýzy na celý obraz, právě z důvodu toho, že kliknutí vytvoří obdélník o nulovém obsahu. Druhým, jednodušším způsobem je použití dedikovaného tlačítka, umístěném v pravé dolní části videopřehrávače.

Přetáčení videa je implementováno klikatelným ukazatelem postupu pod obrazovkou videopřehrávače. Ukazatel má rozmezí od 0 do 100 % a případné kliknutí do prostoru ukazatele nastaví procento, které je poté převedené na odpovídající část ve videu.

- Tabulka záznamů nalezení objektů

  Zelená část s označením 3. je tabulka pro zaznamenávání nalezených objektů. Při analýze videa se zde vypisuje čas, třída a pravděpodobnost nalezeného objektu ve videu. V případě analýzy videozáznamu se vypisuje čas snímku v rámci videa, v případě *real-time* analýzy se vypisuje čas od začátku spuštění analýzy.

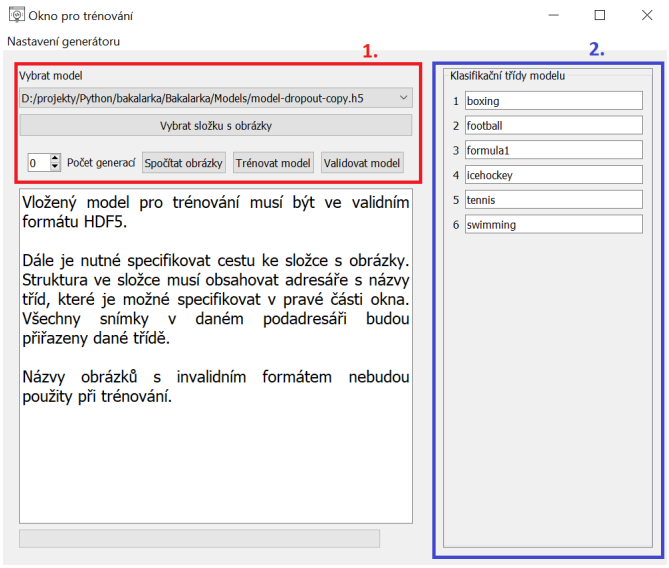
- Horní panel

  Poslední částí hlavního okna je horní lišta. Tato lišta je rozdělena na 2 části. **Soubory** a **Proměnné**. Část Soubory je určena pro vkládání videí, klasifikačních modelů, spuštění analýzy z kamery a otevření trénovacího okna aplikace. Část Proměnné slouží pro jednoduchou změnu systémových proměnných, využívaných při analýze videa. Tyto proměnné nastavují rychlost přehrávání videa, hranice nalezení třídy a velikost zásobníku pro *Rolling averaging prediction*.

Prostředí pro trénování a testování modelů

Okno pro trénování a testování je uvedené na obrázku 4.6 a slouží pro jednoduché nastavení a spuštění trénovacích a testovacích procesů na vybraném modelu a datasetu. Okno se může rozdělit do 2 hlavních částí. Část nastavení a výběru činnosti a část pro mapování souborů ke jménům tříd. Systém anotace snímků, určených pro trénování je v tomto případě implementován pomocí anotace ve jméně složky, ve které se snímky nachází. Popis tohoto způsobu anotace je popsán pod částí nastavení a znamená, že v uživatelem vybrané složce se musí nacházet podadresáře se jmény odpovídajícím názvům tříd, které lze specifikovat v pravé části trénovacího okna. V daném podadresáři se pak nachází všechny trénovací snímky pro danou třídu.

Je nutné podotknout že aplikace má pevně stanovenou strukturu trénovacího/testovacího datasetu a jelikož nepodporuje RPN modely, není schopna využívat takzvané *Region of interest* (ROI) datasety.



Obrázek 4.6: Okno pro trénování modelů sítí.

- Nastavení

  Sekce nastavení je na obrázku 4.6 označena červeně s názvem „1.“ a slouží pro vybrání klasifikačního modelu, složky s dedikovanými snímky a určení počtu generací pro trénování.

  Následně zde jsou 3 spouštěcí tlačítka pro výběr činnosti. Tlačítko „Spočítat obrázky“ projde vybranou složku a zkontroluje validitu snímků. Následně vypíše počet validních snímků pro trénování, pro každou třídu a celkový součet všech snímků. Tlačítko „Validovat model“ je určeno pro validaci modelu. V tomto případě se projdou všechny snímky, předloží se pro evaluaci vybranému klasifikátoru a spočítá se průměrná přesnost a ztrátová funkce přes všechny *batch*.

  Poslední tlačítko slouží pro samotný trénovací proces. Z vybrané složky se začnou nahrávat, pomocí generátoru, snímky ve velikosti *batch* a vybraný model se začne adaptovat na snímky ve složce.

- Nastavení klasifikačních tříd pro trénování

  V základu se berou jména tříd ze souboru `models_path.csv` , ale je možné tyto názvy změnit. Tyto změny jsou ovšem neperzistentní. To znamená, že po ukončení okna se změny neuloží. Sloupec slouží pro namapování snímků k jednotlivým třídám.

- Horní panel

  Stejně jako u hlavního okna se i zde nachází horní panel s částí **Nastavení generátoru**. V tomto menu lze specifikovat detaily pro generátor snímků ze složek. Kromě základní velikosti *batch* poskytuje aplikaci i možnost nastavení předzpracování dat. Konkrétně je možné nastavit rozsahy pro náhodné změny v:

  – jas,
  – vertikální posun,
  – horizontální posun,

– převrácení snímku podle osy y,
– rotace snímku,
– možnost prohození barevných kanálů.

4.3 Vytváření datasetu

V rámci této bakalářské práce došlo i na vyzkoušení vytvoření vlastního datasetu. Pro tyto účely byly vytvořeny 2 počítačové skripty v jazyce Python, které měli na starosti získávání dat.

4.3.1 Příprava dat pro trénování

I přesto, že vytvořené modely sítí pro tuto bakalářkou práci jsou určeny pro analýzu videozáznamů, trénování těchto modelů je nutné provádět na jednotlivých označených snímcích. Z tohoto důvodu se tvoří na internetu volně přístupné datasety s celou škálou použití a zaměření. Tyto datasety mohou mít ovšem několik komplikací.

- Často se jedná o velikostně velice náročné soubory, které je potřeba stáhnout a uložit.
- Předpřipravený dataset má už předem určené klasifikační třídy, které je velice obtížné změnit.
- V předpřipravených datasetech se mohou vyskytovat chyby, které můžou ovlivňovat přesnost trénování a jsou těžce odhalitelné.
- V případě velice známých, používaných datasetů se mohou vyskytnout problémy stejných nedostatků vytvořených systémů.

Pro získání dat byli využity vlastní skripty. Pro *data augmentation* v prostředí aplikace pak byl využit existující objekt `ImageDataGenerator` z Tensorflow.

4.3.2 Skripty pro vytváření datasetu

Skripty byli napsány v jazyce Python a dají se rozdělit na získávání dat z internetových galerií a z videí.

Skript pro stahování obrázků z internetu

Smyslem skriptu pro získávání obrázků z web-galerií je využít implementovaných vyhledávacích algoritmů na webových stránkách, kde stačí vložit do vyhledávače hledaný výraz a algoritmus vyhledá nejlepší výsledky. Skript vyžaduje nastavení několika parametrů pro správnou funkčnost.

- **Webové adresy** - Datový typ slovník, kde klíčem je webová adresa a hodnotou je klasifikační třída snímků, které se na adrese nachází.
- **Index počátečního obrázku** - Důvod k tomuto je přeskočit například logo stránky nebo jiné počáteční obrázky, které jsou nežádoucí.
- **Maximální počet obrázků** - Galerie mívají tendence po určitém počtu obrázků už nebýt tak přesné. Tento parametr dovoluje méně přesné nálezy nestahovat.

- **Složka pro ukládání snímků**.

Skript vytváří HTTP GET požadavky, které posílá serverům a stahuje si od nich zdrojový HTML kód stránky. V tomto kódu následně vyhledá všechny `img` elementy a jejich `src` atributy. Tyto atributy pak ve většině případů odkazují na webovou adresu, ze které se dají obrázky stáhnout. Každý jednotlivý obrázek je pak uložen do dané složky se jménem `třída_index.jpg`, kde třída je daná a index je určen podle posledního indexu dané třídy ve složce.

Hlavní zdroje byli internetová galerie freeimages.com[4] a Google images [5].

Skript pro extrakci snímků z videa

Skript pro získávání obrázků z videí využívá OpenCV pro práci s videem. Tomuto skriptu je nutno definovat jako vstupní parametry:

- **Video** - Video, ze kterého se berou snímky.
- **Klasifikační třída** - Klasifikační třídu pro dané video.
- **Počet snímků** - Požadovaný počet snímků z videa.
- **Složka pro ukládání snímků**.

Skript z nahraného videa zjistí počet snímků a pak podle pseudonáhodného generátoru čísel s rovnoměrným rozložením vybere snímky, které extrahuje a uloží. Formát uložení je ve stejném tvaru jako ukládá skript pro stahování z internetu.

[4] https://www.freeimages.com
[5] https://www.google.com

Kapitola 5

Experimenty

V rámci této sekce jsou popsány experimenty a porovnání, provedené na výše zmíněných modelech a na aplikaci. Tyto experimenty a porovnání se zaměřovali na přesnost modelů a rychlost evaluace videa v aplikaci. První experiment se zaměřoval na přesnost modelů na testovacích datech. Jako druhé je uvedeno porovnání rychlosti evaluace snímků v prostředí implementované aplikaci a jako poslední experiment je uvedeno porovnání dopadu využití předzpracování dat na trénování a přesnost modelu.

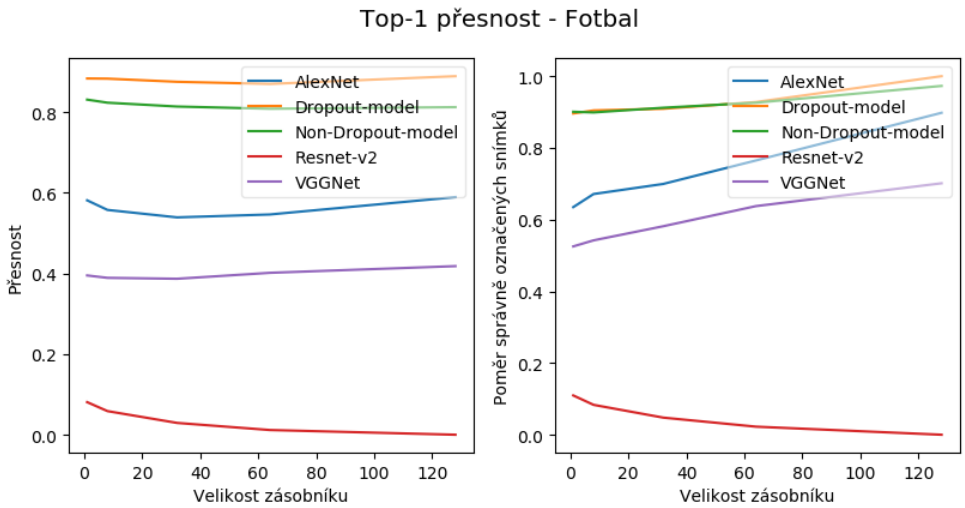
5.1 Porovnání přesnosti modelů

Prvním experimentem bylo porovnání top-1 přesnosti na testovacích videích. Top-1 přesnost se dívá na nejpravděpodobnější třídu, kterou predikoval klasifikátor a v případě správného označení se počítá pravděpodobnost třídy a v případě nesprávného označení se automaticky počítá 0. V rámci tohoto porovnání byl rovnou proveden experiment, jak implementovaná fronta pro *Rolling averaging prediction* o různé velikosti ovlivňuje přesnost klasifikace.

Jako testovací podklad pro tento experiment bylo staženo několik videí z internetového serveru Youtube[1] a tyto videa byla následně sestříhána do krátkých časových úseků, ve kterých se vyskytovala daná klasifikační třída.

Výsledky tohoto experimentu jsou uvedeny v příloze A. Zde je uvedený pouze jeden z výsledků jako reprezentační příklad. Na každém obrázku jsou výsledky všech modelů pro video jednoho sportu. Každý z obrázků obsahuje 2 grafy. Oba grafy mají nastavenou osu x jako velikost zásobníku pro *Rolling averaging prediction*. Zásobník byl u každého videa nastaven na hodnoty 1, 8, 32, 64 a 128. Osa y levého grafu pak označuje top-1 přesnost a osa y pravého grafu označuje poměrovou část správně označených snímků ve videu.

[1] https://www.youtube.com



Obrázek 5.1: Výsledky jednotlivých modelů pro video fotbalového zápasu.

Výsledky ukázali, že zásobník má ve většině případů kladnou odezvu na klasifikaci videa. Pokud má klasifikátor, při nastavení fronty na hodnotu 1, neboli při vypnutí *Rolling averaging prediction*, přesnost alespoň 35 % tak podle uvedených výsledků má fronta kladný vliv na poměr správně označených snímků, a to až o 20 %.

Ovšem vyskytli se i případy, kdy přesnost klasifikátoru, s rostoucí frontou klesal. Tyto případy jsou ovšem vázané na situace, kdy už samotný model bez podpory fronty nedosahuje dobrých výsledků.

Další situací byl případ kdy s rostoucí frontou šla top-1 přesnost dolů, ale poměr správně označených snímků stoupal. Tohoto bylo dosaženo v případech, kdy docházelo právě ke *flickering* efektu neboli se objevovali krátké případy kdy model označil snímek videa chybnou třídou, takže nulová top-1 přesnost stahovala průměr dolů, ale díky uchovávání předchozích predikcí a vypočítání jejich průměru se i přesto třída klasifikovala správně.

Je nutné zmínit, že uživatel, využívající frontu pro *Rolling averaging prediction*, musí vědět, jak vypadá obsah videa. Jestliže je na videu převážně jedna klasifikační třída a úkolem je odhadnout která, může mít fronta pozitivní účinek. Pokud se ovšem jedná o detekci objektu ve videu, kde objekt se může objevit pouze na malý okamžik, jako například vyhledávání člověka v záznamu průmyslových kamer, pak může mít fronta značně nepříznivý dopad.

Posledním zajímavým úkazem, který je možné vidět na výsledcích je veliký rozdíl schopností obou vytvořených modelů. Tohoto rozdílu je možné si všimnout třeba na grafu výsledků pro video boxerských zápasů A.1. Je viditelné, že model bez *Dropout* vrstvy dosáhl relativně dobrých výsledků, zatím co model s *Dropout* vrstvou skončil hluboce nejhorší. Ovšem v celkovém srovnání všechn sportů dosáhli modely srovnatelných výsledků. Vysvětlením může být rozdílná počáteční inicializace vah a také jiné složení *batch* při trénování pomocí *Mini-batch descent*, což vedlo k jiným negativním gradientům a jiným konečným stavům klasifikátorů. Dalším důvodem může být pak samotná *Dropout* vrstva, která zavedla do trénovacího procesu regularizaci.

5.2 Rychlost analýzy modelů

Dalším podsekcí je jednoduché porovnání rychlosti evaluace videí v implementované aplikaci. Pro porovnání byly použity předzpracované modely a pouze jeden vlastní model. Rychlost se poté měřila v jednotkách **fps** (*frames per second*) a byli pro to využity 4 stejná videa v jiné kvalitě. Konkrétně se pak měřila rychlost na videích o kvalitě 360p, 480p, 720p a 1080p. Všechny měření byly provedeny ve stejných podmínkách a na stejném stroji. Pro odběr času byla použita knihovna **time**, která je integrovaná v Python core. Tabulka výsledků vypadá následovně:

	360p	480p	720p	1080p
Vlastní model	26	25	23	20
Alexnet	12	12	11	10,5
VGGnet-16	5	4,5	4,5	4,5
Resnet-v2	4,5	4	4	4

Tabulka 5.1: Výsledky měření rychlosti evaluace videa v aplikaci. Výsledky jsou uvedeny v jednotkách *frames per second* (fps).

Z výsledků v tabulce 5.1 lze vidět, že vlastní model rapidně přesáhl rychlostí ostatní modely. Důvodem je mnohem menší vstupní tenzor implementovaného modelu, který má dimensionalitu pouze (128, 128, 3) oproti například Resnet-v2, kde vstupní tenzor má rozměry (299, 299, 3). Dalším důvodem těchto výsledků je samotný velký rozdíl velikostí sítí. Zatím co implementovaný model má okolo 11 milionů trénovatelných parametrů, VGGnet-16 má přes 136 milionů parametrů.

Tyto výsledky potvrzují, že vybrané modely se hodí spíše na evaluaci samotných snímků než videí. V rámci této práce byl proveden i experiment na jednoduché *Early Fusion* síti, která brala 10 snímků videa jako vstup a obsahovala přes 123 milionů parametrů. Je potřeba zmínit, že tento experiment se neodehrával v podmínkách aplikace, a proto můžou být jeho výsledky zkreslené.

	360p	480p	720p	1080p
Early Fusion model	132	118	96	67

Tabulka 5.2: Výsledky měření rychlosti evaluace videa. Výsledky jsou uvedeny v jednotkách *frames per second* (fps).

Je jasně vidět, podle tabulky 5.2, že i přes větší model sítě a použití 3D konvolučních vrstev, které jsou vysoce výpočetně a časově náročné, má *Early Fusion* model znatelně lepší výsledky v rámci rychlosti evaluace. Důvodem je 10-násobné urychlení klasifikace, jelikož model neanalyzuje každý snímek samostatně, ale zpracovává 10 snímků dohromady.

Vzhledem k tomu, že test probíhal v dedikovaných podmínkách je možné, že v případě testu v aplikaci by se dosáhlo horších výsledků.

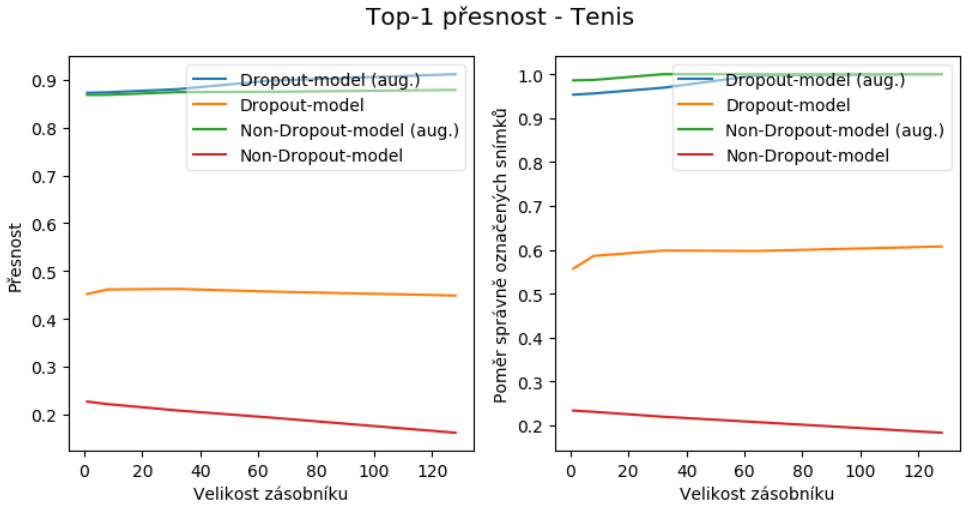
5.3 Vliv předzpracování dat na přesnost modelu

Posledním experimentem této práce bylo porovnání přesnosti vlastních modelů v případě kdy se jeden z modelů trénoval na datech, které byly upraveny pomocí *data augmentation*. Modely mají stejnou architekturu, byli trénovány na stejných trénovacích datech a na stejný

počet generací. Nutno podotknout, že výsledky mohou být zkreslené, kvůli odlišné počáteční inicializaci vah a kvůli využití *Mini-batch-descent training*. Předzpracování dat bylo nastaveno na parametry:

- náhodná rotace snímku v rozmezí: $u \in < -30°, 30° >$,
- náhodný horizontální posun snímku v rozmezí: $p \in < -0,2, 0,2 >$,
- náhodný vertikální posun snímku v rozmezí: $p \in < -0,2, 0,2 >$,
- možnost převrácení snímku podle osy y,
- náhodný jas snímku v rozmezí: $j \in < 0,7, 1,2 >$,
- náhodné přiblížení snímku v rozmezí: $p \in < 0,8, 1,2 >$,
- možnost změny barev ve snímku.

Stejně jako v prvním experimentu se výsledky vytvářeli pomocí stejných modifikovaných videí z video-portálu Youtube.



Obrázek 5.2: Výsledky porovnání modelů pro video s tenisovými zápasy.

Všechny výsledky jsou k nalezení v příloze B. Z výsledků je patrné, že v některých případech předzpracování dat nepomohlo a například u výsledků fotbalového zápasu mělo předzpracování dat i záporný efekt. Ovšem tohle je odůvodnitelné kvůli nedostačujícímu testovacímu a trénovacímu datasetu. Předzpracování dat má za úkol dosáhnout toho, aby trénovaný model nalezl správnou nadrovinu, podle které pak vytváří regresní dělící křivku. V případě fotbalového zápasu B.2 došlo k přeučení sítě, která byla natrénovaná bez předzpracování dat a klasifikovala třídu „fotbal“ hlavně podle poměru zelené barvy neboli fotbalového trávníku v obraze. Síť, u které bylo použito předzpracování dat se takto přeučit nemohla. Předzpracování dat se ukázalo být účinné při analýzy tenisového videa 5.2, kde nahrávka obsahovala 3 rychlé hry, každou hru na povrchu jiné barvy (tráva, antuka, DecoTurf). V tomto případě model bez zpracování dat dosáhl horších výsledků.

Kapitola 6

Závěr

Tato práce měla za úkol seznámit se s problematikou analýzy videa a shrnout některé algoritmy, které je možné pro tyto úkoly využít. Dále pak navrhnout a implementovat klasifikátor, který je možné využít pro analýzu videa a také navrhnout a implementovat uživatelské rozhraní, které bude dovolovat uživateli nahrát a vybrat model klasifikátoru a využít jej pro analýzu vlastního videa. Uživatelské rozhraní mělo také dovolovat trénovaní těchto klasifikátorů na vlastních datasetech.

Jako první představila práce čtenáři myšlenku strojového učení a popsala základní praktiky, které se využívají v *Machine learning* systémech. Dále pak čtenáře seznámila s obrazem, možnostmi jeho zpracování a algoritmy pro tvorbu vlastních datasetů. Práce shrnula některé používané algoritmy strojového učení pro analýzu obrazu a videa a to nejen v rámci neuronových sítí, ale také jiné algoritmy jako je například *Random forests*.

V další části je shrnutý samotný návrh klasifikačního modelu a uživatelského rozhraní. Pro klasifikátor byla využita konvoluční neuronová síť a uživatelské rozhraní bylo navrhnuto jako desktopová aplikace s podporou přehrání videa, vybrání části analyzovaného obrazu a možnost klasifikace pomocí vlastních modelů.

V části experimentů bylo provedeno porovnání mezi vlastními vytvořenými modely a před-implementovanými vybranými modely, které jsou vhodné pro analýzu obrazu a jsou používané pro tyto účely. Experimenty byli provedeny za účelem zjištění přesnosti a rychlosti modelů, ale také za účelem zjištění dopadu využití *data augmentation* při trénovacím procesu.

Z výsledků bylo patrné že větší síť nemusí nutně znamenat větší přesnost a vlastní implementovaný model si mezi známými sítěmi poradil více než obstojně. Ovšem z důvodu malého datasetu a trénování na velký počet generací došlo k přetrénování a přesnost modelů vysoce oscilovala mezi 0-100 %.

Aplikace s možností přehrávání videí a trénování modelů poskytuje přiměřenou škálu možností pro uživatele. Ovšem bez podpory RPN modelů a trénování na ROI datasetech se aplikace stává použitelnou pouze pro „staré“ modely, které klasifikují obraz jakožto celek. Rozšířením by tedy určitě byla možnost využití RPN modelů a předělání systému trénování pro možnost použití anotačních souborů.

Literatura

[1] *Neuronová sítě* [online]. 2018 [cit. 2019-11-28]. Dostupné z: http://www.trilobyte.cz/downloadfree/qcemanual/neural_net.pdf.

[2] *Implement the Max-Pooling operation from Convolutional Neural Networks* [online]. 2019 [cit. 2019-12-29]. Dostupné z: https://codegolf.stackexchange.com/questions/195348/implement-the-max-pooling-operation-from-convolutional-neural-networks.

[3] 2020, M. *Types of Bitmaps* [online]. [cit. 2020-03-28]. Dostupné z: https://docs.microsoft.com/en-us/windows/win32/gdiplus/-gdiplus-types-of-bitmaps-about.

[4] Anwar, A. *Difference between AlexNet, VGGNet, ResNet and Inception* [online]. [cit. 2020-04-16]. Dostupné z: https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96.

[5] Bell, J. *Machine Learning: Hands-On for Developers and Technical Professionals*. Wiley, 2014. ISBN 9781118889497.

[6] Bohrn, M. *PERSPEKTIVNÍ OBVODOVÉ STRUKTURY PRO MODULÁRNÍ NEURONOVÉ SÍTĚ*. Brno, CZ, 2014. Dizertační práce. Vysoké učení technické v Brně, Fakulta Elektrotechniky a komunikačních technologií. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=94276.

[7] Breiman, L. Random forests. *Machine learning*. Springer. 2001, sv. 45, č. 1, s. 5–32.

[8] Brownlee, J. *Supervised and Unsupervised Machine Learning Algorithms* [online]. [cit. 2019-12-10]. Dostupné z: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[9] Chalupník, V. *Biologické algoritmy* [online]. [cit. 2019-12-10]. Dostupné z: https://www.root.cz/clanky/biologicke-algoritmy-4-neuronove-site/.

[10] Chalupník, V. *Biologické algoritmy (5) - Neuronové sítě* [online]. [cit. 2020-05-27]. Dostupné z: https://www.root.cz/clanky/biologicke-algoritmy-5-neuronove-site/.

[11] Cook, J. D. *Three algorithms for converting color to grayscale* [online]. [cit. 2020-03-29]. Dostupné z: https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/.

[12] Dabbura, I. *Gradient Descent Algorithm and Its Variants* [online]. [cit. 2020-02-13]. Dostupné z: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3.

[13] Davis, A. *Perceptron Tutorial* [online]. [cit. 2019-11-05]. Dostupné z: https://www.simplilearn.com/what-is-perceptron-tutorial.

[14] Girshick, R. Fast r-cnn. In: *Proceedings of the IEEE international conference on computer vision*. 2015, s. 1440–1448.

[15] Girshick, R., Donahue, J., Darrell, T. a Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, s. 580–587.

[16] Hradiš, M. *Konvoluční neuronové sítě*. Fakulta informačních technologií Brno, 2015. Dostupné z: https://openalt.cz/2015/data/MichalHradis-Konvolucnineuronovesite.pdf.

[17] Karim, R. *Illustrated: 10 CNN Architectures* [online]. [cit. 2020-04-16]. Dostupné z: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d.

[18] Kelbel, J. a Šilhán, D. *Shluková analýza* [online]. 2019 [cit. 2019-12-29]. Dostupné z: http://cmp.felk.cvut.cz/cmp/courses/recognition/zapis_prednasky/zapis_02/13/shlukovani.pdf.

[19] Kong, T., Yao, A., Chen, Y. a Sun, F. HyperNet: Towards Accurate Region Proposal Generation and Joint Object Detection. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016.

[20] Kučera, J. *Hierarchistické metody shlukování* [online]. [cit. 2019-12-27]. Dostupné z: https://is.muni.cz/th/172767/fi_b/5739129/web/web/hiermet.html.

[21] Kučera, J. *Nehierarchistické metody shlukování* [online]. [cit. 2019-12-27]. Dostupné z: https://is.muni.cz/th/172767/fi_b/5739129/web/web/nehiermet.html.

[22] Kučera, J. *Neřízená klasifikace obrazu* [online]. [cit. 2019-12-27]. Dostupné z: https://is.muni.cz/el/1431/podzim2009/Z8114/um/9018418/DZO_08_klasifikace_2.pdf.

[23] Kučerová, H. *Neuronová síť*. KTD: Česká terminologická databáze knihovnictví a informační vědy (TDKIV).

[24] Lukáš, B. *Strojové učení a rozpoznávání* [online]. [cit. 2020-02-11]. Dostupné z: https://www.fit.vutbr.cz/study/courses/SUR/public/prednasky/.

[25] Masarykovy univerzity, I. biostatistiky a analýz Lékařské fakulty. *Matematická biologie: e-learningová učebnice* [online]. [cit. 2020-02-06]. Dostupné z: https://portal.matematickabiologie.cz/index.php.

[26] matoušek, V. *Strojové učení* [online]. 2019 [cit. 2019-12-29]. Dostupné z: https://www.kiv.zcu.cz/studies/predmety/uzi/Folie_ZS/Stroj_uceni.pdf.

[27] Michal Španěl, T. M. *Uvod do předmětu* [online]. [cit. 2020-03-28]. Dostupné z: https://wis.fit.vutbr.cz/FIT/st/cwk.php.cs?id=12815&csid=672364#P%C5%99edn%C3%A1%C5%A1ky.

[28] Mohamed, I. S. *An example of convolution operation in 2D* [online]. [cit. 2020-05-27]. Dostupné z: https://www.researchgate.net/figure/An-example-of-convolution-operation-in-2D-2_fig3_324165524.

[29] Nathan Landman, C. W. E. R. *K-Means Clustering* [online]. 2019 [cit. 2019-12-27]. Dostupné z: https://brilliant.org/wiki/k-means-clustering/#k-means-algorithm.

[30] OpenCV. *Introduction* [online]. [cit. 2020-04-11]. Dostupné z: https://docs.opencv.org.

[31] Parkhi, O. M., Vedaldi, A. a Zisserman, A. Deep Face Recognition. In: Xianghua Xie, M. W. J. a Tam, G. K. L., ed. *Proceedings of the British Machine Vision Conference (BMVC)*. BMVA Press, September 2015, s. 41.1–41.12. DOI: 10.5244/C.29.41. ISBN 1-901725-53-7. Dostupné z: https://dx.doi.org/10.5244/C.29.41.

[32] Qt. *About Qt* [online]. [cit. 2020-04-11]. Dostupné z: https://wiki.qt.io/About_Qt.

[33] Redmon, J., Divvala, S., Girshick, R. a Farhadi, A. You only look once: Unified, real-time object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, s. 779–788.

[34] Ren, S., He, K., Girshick, R. B. a Sun, J. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, abs/1506.01497. Dostupné z: http://arxiv.org/abs/1506.01497.

[35] Rosebrock, A. *Video classification with Keras and Deep Learning* [online]. 2020 [cit. 2020-05-08]. Dostupné z: https://www.pyimagesearch.com/2019/07/15/video-classification-with-keras-and-deep-learning/.

[36] Sanderson, G. *Backpropagation calculus | Deep learning, chapter 4* [online]. 2017 [cit. 2020-02-12]. Dostupné z: https://www.youtube.com/watch?v=tIeHLnjs5U8&t=332s.

[37] Sanderson, G. *What is backpropagation really doing? | Deep learning, chapter 3* [online]. 2017 [cit. 2020-02-12]. Dostupné z: https://www.youtube.com/watch?v=Ilg3gGewQ5U.

[38] Sebastian Zambanini, M. K. *Early versus Late Fusion in a Multiple Camera Network for Fall Detection* [online]. 2019 [cit. 2020-05-11]. Dostupné z: https://www.researchgate.net/profile/Martin_Kampel/publication/228722557_Early_versus_Late_Fusion_in_a_Multiple_Camera_Network_for_Fall_Detection/links/0fcfd50cb84a7ce6e4000000.pdf.

[39] Sewak, M., Karim, M. R. a Pujari, P. *Practical Convolutional Neural Networks: Implement advanced deep learning models using Python*. Packt Publishing Ltd, 2018.

[40] Simonyan, K. a Zisserman, A. Two-Stream Convolutional Networks for Action Recognition in Videos. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. a Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc., 2014, s. 568–576. Dostupné z: http://papers.nips.cc/paper/5353-two-stream-convolutional-networks-for-action-recognition-in-videos.pdf.

[41] Szegedy, C., Ioffe, S. a Vanhoucke, V. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *CoRR*. 2016, abs/1602.07261. Dostupné z: http://arxiv.org/abs/1602.07261.

[42] Tahsildar, S. *Random Forest Algorithm In Trading Using Python*. Mar 2019. Dostupné z: https://blog.quantinsti.com/random-forest-algorithm-in-python/.

[43] Tensorflow. *Why TensorFlow* [online]. [cit. 2020-04-11]. Dostupné z: https://www.tensorflow.org.

[44] [US], P. S. F. *PyQt5 5.14.2* [online]. [cit. 2020-04-11]. Dostupné z: https://pypi.org/project/PyQt5/.

[45] Veselovský, B. M. *PLÁNOVÁNÍ CESTY ROBOTU POMOCÍ POSILOVANÉHO UČENÍ*. Brno, CZ, 2012. Diplomová práce. Vysoké učení technické v Brně, Fakulta stroního inžnýství. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=66162.

[46] Vlachynský, B. J. *Video analýza dat z kamerového systému*. Zlín, CZ, 2019. Diplomová práce. Univerzita Tomáše Bati ve Zlíně. Dostupné z: https://digilib.k.utb.cz/bitstream/handle/10563/44429/vlachynský_2019_dp.pdf?sequence=1.

[47] Vojáček, A. *Samoučící se neuronová síť - SOM, Kohonenovy mapy* [online]. 2006 [cit. 2019-04-27]. Dostupné z: https://www.kiv.zcu.cz/studies/predmety/uir/NS/Samouc_NN2.pdf.

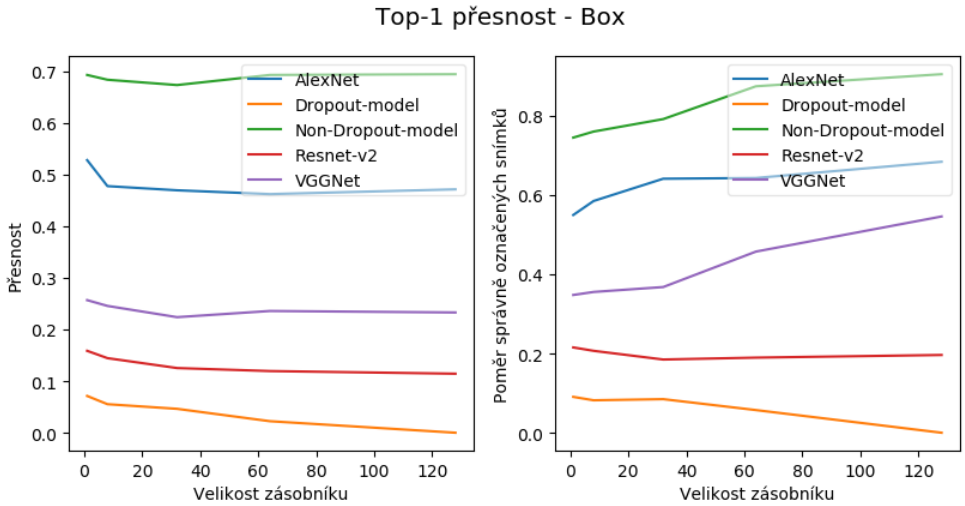
[48] Vryniotis, V. *Tuning the learning rate in Gradient Descent* [online]. 2013 [cit. 2020-02-12]. Dostupné z: https://blog.datumbox.com/tuning-the-learning-rate-in-gradient-descent/.

[49] Wei, J. *AlexNet: The Architecture that Challenged CNNs* [online]. [cit. 2020-04-16]. Dostupné z: https://towardsdatascience.com/alexnet-the-architecture-that-challenged-cnns-e406d5297951.

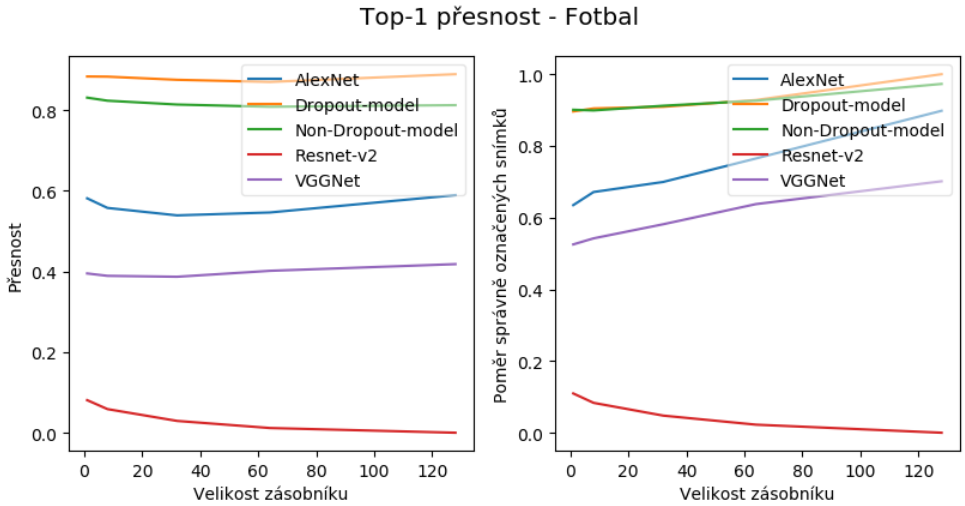
[50] Zálešká, K. *KLASIFIKACE OBRAZŮ POMOCÍ UMĚLÝCH NEURONOVÝCH SÍTÍ*. Brno, CZ, 2011. Bakalářská práce. Masarykova Univerzita, Fakulta Přírodovědecká. Dostupné z: https://is.muni.cz/th/mqtzd/bakalarska_prace.pdf.

Příloha A

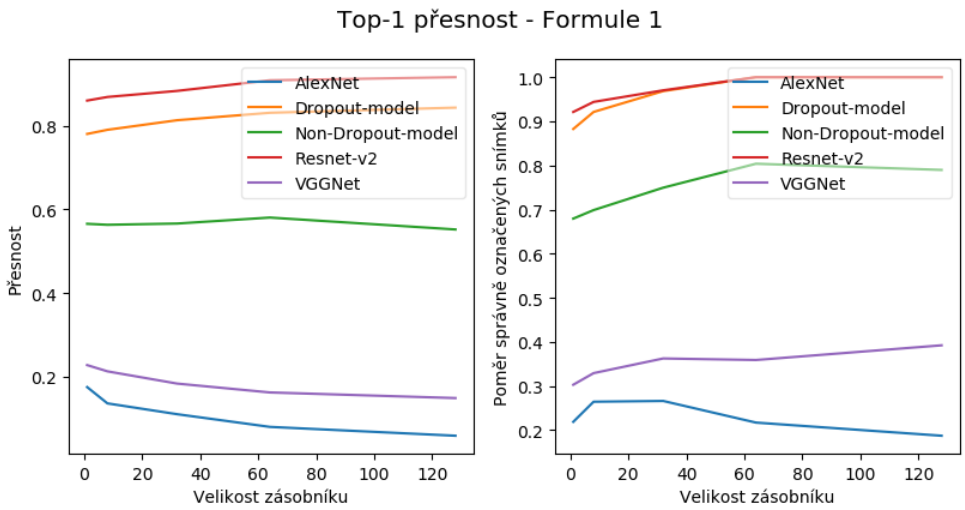
Výsledky porovnání přesnosti modelů



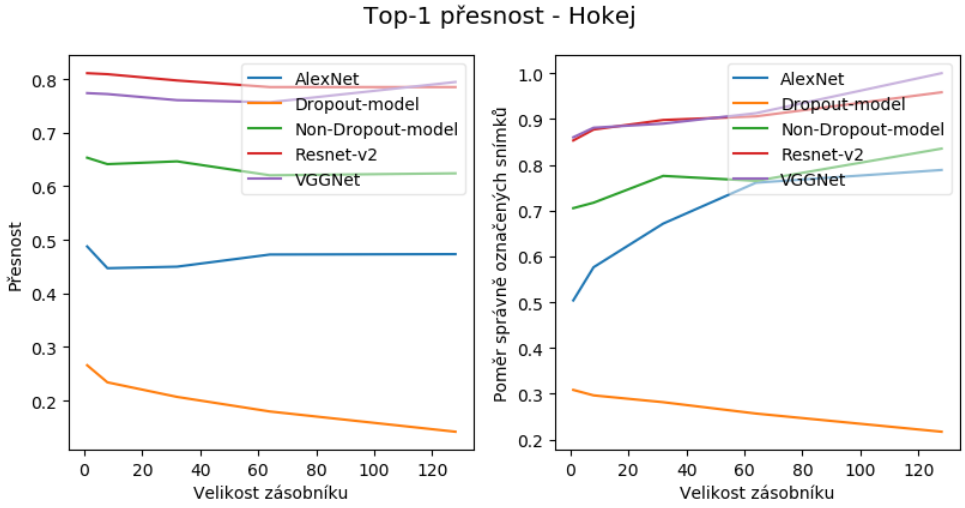
Obrázek A.1: Výsledky jednotlivých modelů pro video se zápasy boxu.



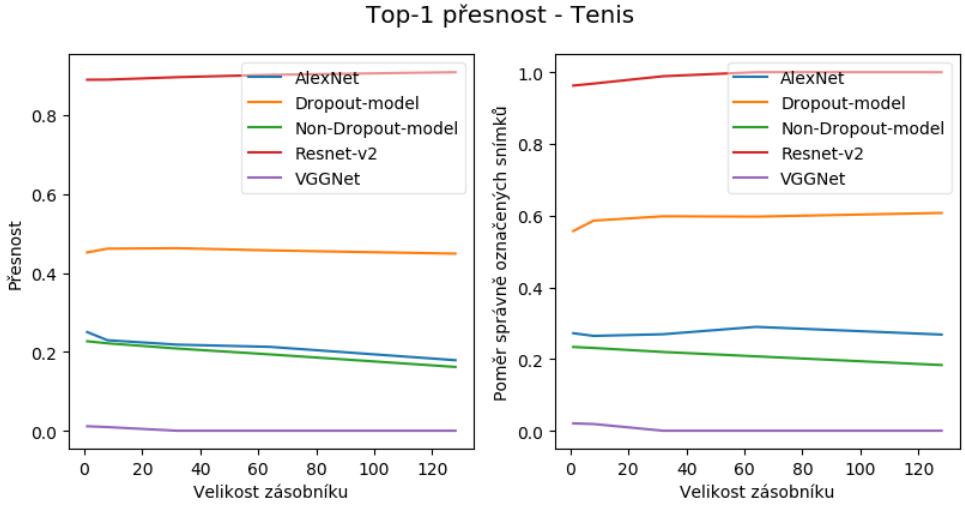
Obrázek A.2: Výsledky jednotlivých modelů pro video fotbalového zápasu.



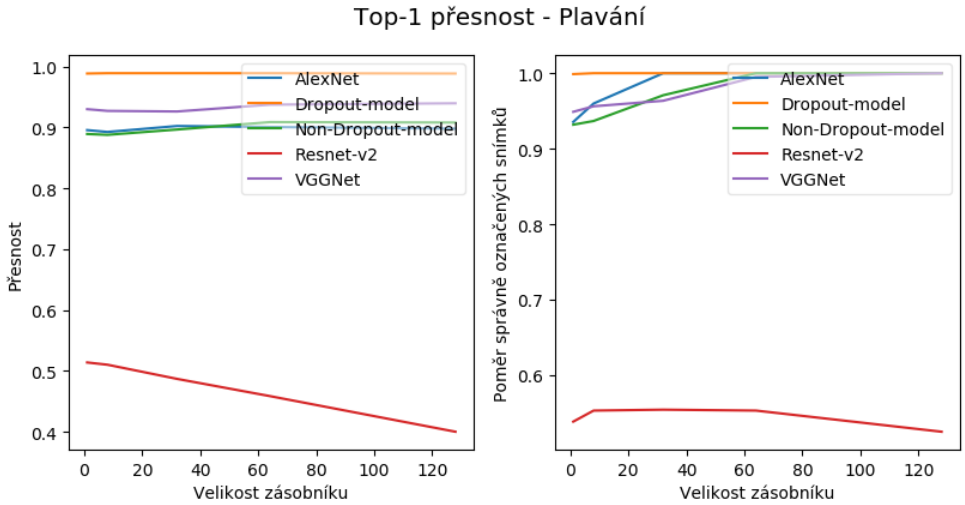
Obrázek A.3: Výsledky jednotlivých modelů pro video se závody Formule 1.



Obrázek A.4: Výsledky jednotlivých modelů pro video s hokejovými zápasy.



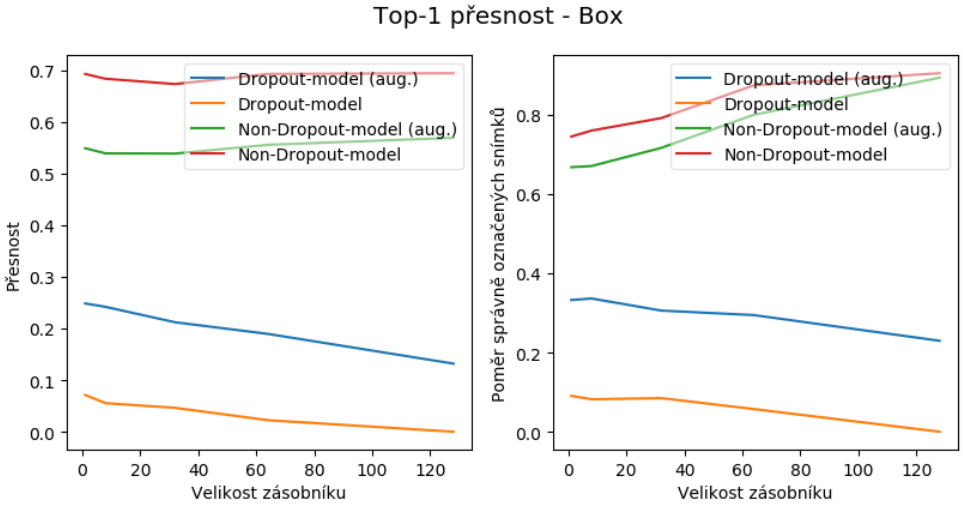
Obrázek A.5: Výsledky jednotlivých modelů pro video s tenisovými zápasy.



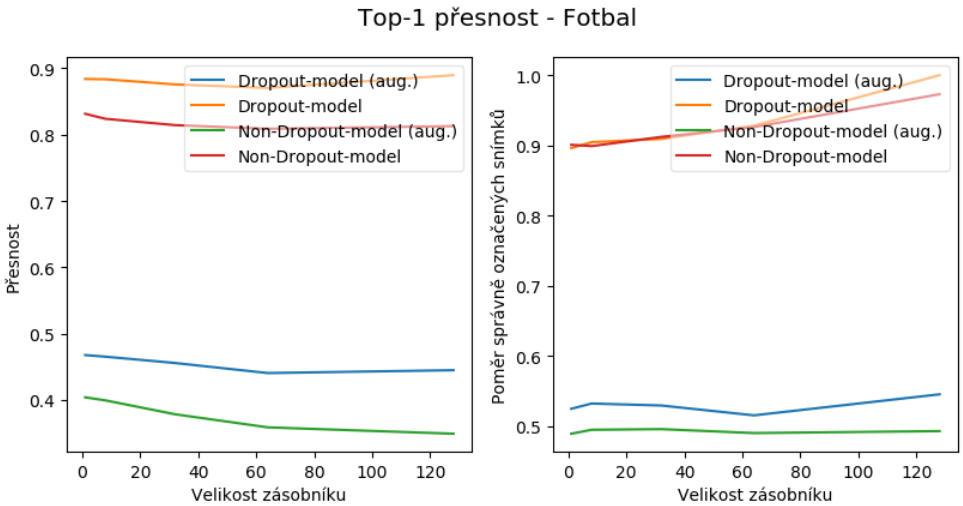
Obrázek A.6: Výsledky jednotlivých modelů pro video s plaváním.

Příloha B

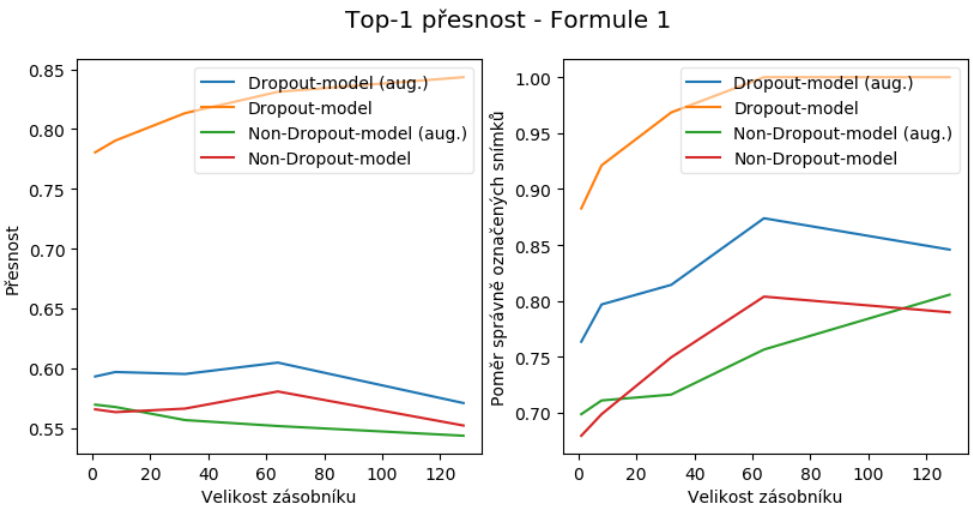
Výsledky porovnání přesnosti při použití předzpracování dat



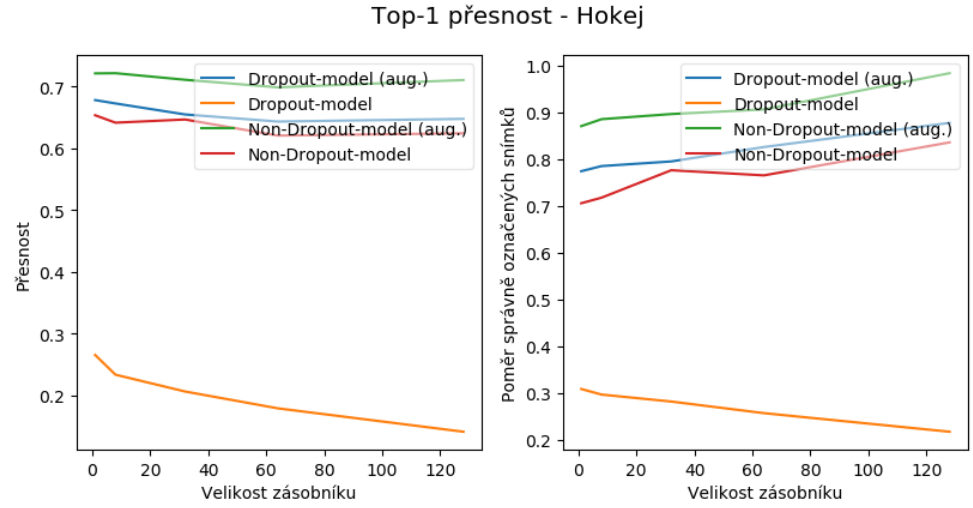
Obrázek B.1: Výsledky jednotlivých modelů pro video se zápasy boxu.



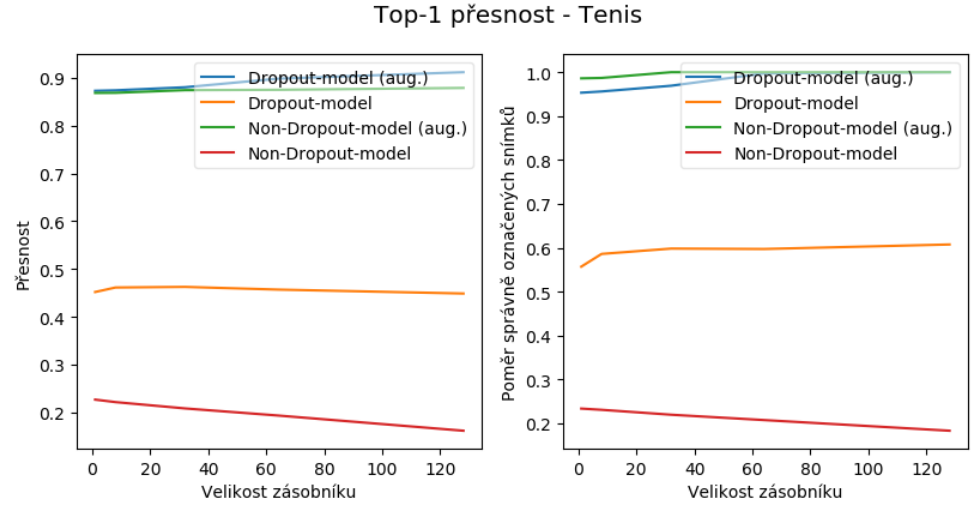
Obrázek B.2: Výsledky jednotlivých modelů pro video fotbalového zápasu.



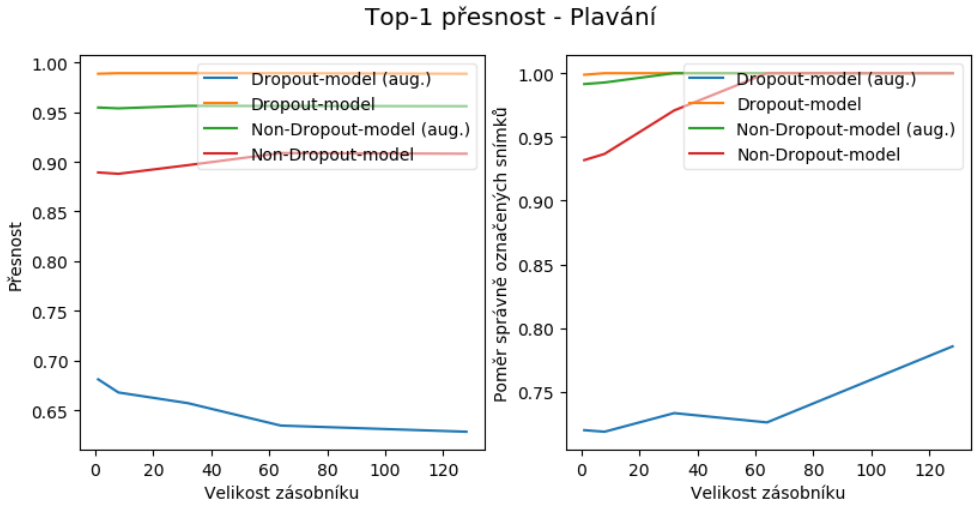
Obrázek B.3: Výsledky jednotlivých modelů pro video se závody Formule 1.



Obrázek B.4: Výsledky jednotlivých modelů pro video s hokejovými zápasy.



Obrázek B.5: Výsledky jednotlivých modelů pro video s tenisovými zápasy.



Obrázek B.6: Výsledky jednotlivých modelů pro video s plaváním.

Příloha C

Obsah paměťového média

Na paměťovém médiu se nachází 6 adresářů rozepsané níže, text technické zprávy `xsaman02.pdf` a `README.txt` soubor ve které jsou uvedené bližší informace o spuštění aplikace a dependencích.

- **Application**

  Adresář uchovává zdrojové kódy a potřebné soubory ke spuštění aplikace. Jedná se o jediný potřebný adresář v paměťovém médiu pro spuštění aplikace. Adresář obsahuje další podadresáře.

  - **data** - Při prvotním spuštění je tento adresář prázdný a aplikace do něj ukládá vlastní soubory.
  - **icons** - Adresář obsahuje obrázky ikon, které se v aplikaci využívají.
  - **src** - Adresář obsahuje zdrojové kódy aplikace.

- **model&train scripts** - Tento adresář obsahuje soubory pro trénování a vytváření modelů sítí.
- **Models** - Tento adresář obsahuje vytvořené a natrénované modely neuronových sítí ve formátu HDF5.
- **data&statistics** - Tento adresář obsahuje skripty pro vytváření vlastního datasetu.
- **Demos** - Tento adresář obsahuje podadresáře:

  - **Application demos** - Tento adresář obsahuje 2 krátká videa, pro ukázku funkcionality aplikace.
  - **Test videos** - V tomto adresáři se nachází videa na kterých se prováděli experimenty.

- **TeX** - Tento adresář obsahuje zdrojové kódy a ostatní dependence pro přeložení a vygenerování technické zprávy.