VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Využití kamerového systému pro zajištěni bezpečnosti osob na pracovišti

Use of Surveillance Cameras to Ensure the Safety of People in the Workplace

Filip Łuński

Diplomová práce

Vedoucí práce: Ing. Tomáš Wiszczor, Ph.D.

Ostrava, 2025

Zadání diplomové práce

Student: **Bc. Filip Łuński**

Studijní program: N0613A140034 Informatika

Téma: Využití kamerového systému pro zajištěni bezpečnosti osob na pracovišti
Use of Surveillance Cameras to Ensure the Safety of People in the Workplace

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit a otestovat prototyp systému pro detekci incidentů na pracovišti pomocí analýzy pohybů a póz osob v reálném čase na kamerových záznamech. Systém bude využívat algoritmy strojového učení po detekci a klasifikaci incidentů jako například pády, volání o pomoc nebo jiné kritické situace.

1. Prostudujte a popište dostupné algoritmy pro detekci a klasifikaci objektů v obrazech a zhodnoťte jejich použitelnost pro detekci osob v reálném čase.
2. Zmapujte a popište dostupná řešení pro detekci klíčových bodů lidského těla s důrazem na jejich využitelnost pro real-time analýzu pohybu osob.
3. Vybraná řešení pro detekci klíčových bodů lidského těla otestujte s ohledem na rychlost zpracování, přesnost detekce a jejich použití v reálném čase.
4. Vytvořte řešení využívající vhodné techniky strojového učení, které na základě klíčových bodů detekuje v reálném čase pózy indikující bezpečnostní incident (např. pád nebo volání o pomoc).
5. Výsledný prototyp otestujte s různými vstupními parametry, jako jsou rozlišení kamer, různé prostředí, různé úrovně osvětlení, a porovnejte výkonnost na různém hardware (včetně GPU).

Seznam doporučené odborné literatury:

[1] Sultana, Farhana, Abu Sufian and Paramartha Dutta. “A Review of Object Detection Models based on Convolutional Neural Network.” ArXiv abs/1905.01614 (2019): n. pag.
[2] Redmon, Joseph, Santosh Kumar Divvala, Ross B. Girshick and Ali Farhadi. “You Only Look Once: Unified, Real-Time Object Detection.” 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015): 779-788.
[3] Redmon, Joseph and Ali Farhadi. “YOLO9000: Better, Faster, Stronger.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016): 6517-6525.
[4] Wang, Chien-Yao, I-Hau Yeh and Hongpeng Liao. “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information.” ArXiv abs/2402.13616 (2024): n. pag.
[5] Liu, W., Dragomir Anguelov, D. Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu and Alexander C. Berg. “SSD: Single Shot MultiBox Detector.” European Conference on Computer Vision (2015).
[6] Cao, Zhe, Gines Hidalgo, Tomas Simon, Shih-En Wei and Yaser Sheikh. “OpenPose: Realtime Multi-Person 2D Pose Estimation Using Part Affinity Fields.” IEEE Transactions on Pattern Analysis and Machine Intelligence 43 (2018): 172-186.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Wiszczor**

Datum zadání: 01.09.2024

Datum odevzdání: 30.04.2025

Garant studijního programu: prof. RNDr. Václav Snášel, CSc.

V IS EDISON zadáno: 26.11.2024 15:19:39

Abstrakt

Tohle je český abstrakt, zbytek odstavce je tvořen výplňovým textem. Naší si rozmachu potřebami s posílat v poskytnout ty má plot. Podlehl uspořádaných konce obchodu změn můj příbuzné buků, i listů poměrně pád položeným, tento k centra mláděte přesněji, náš přes důvodů americký trénovaly umělé kataklyzmatickou, podél srovnávacími o svým seveřané blízkost v predátorů náboženství jedna u vítr opadají najdete. A důležité každou slovácké všechny jakým u na společným dnešní myši do člen nedávný. Zjistí hází vymíráním výborná.

Klíčová slova

python, strojové učení, neuronové sítě, konvoluční neuronové sítě, rekurentní neuronové sítě, detekce pozy, detekce chování, detekce pádu, YOLO,

Abstract

This is English abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Fusce tellus odio, dapibus id fermentum quis, suscipit id erat. Aenean placerat. Vivamus ac leo pretium faucibus. Duis risus. Fusce consectetuer risus a nunc. Duis ante orci, molestie vitae vehicula venenatis, tincidunt ac pede. Aliquam erat volutpat. Donec vitae arcu. Nullam lectus justo, vulputate eget mollis sed, tempor sed magna. Curabitur ligula sapien, pulvinar a vestibulum quis, facilisis vel sapien. Vestibulum fermentum tortor id mi. Etiam bibendum elit eget erat. Pellentesque pretium lectus id turpis. Nulla quis diam.

Keywords

python, machine learning, neural networks, convolutional neural networks, reccurent neural networks, pose estimation, behaviour detection, fall detection, YOLO

Obsah

Seznam použitých zkratek a symbolů

AF	–	Aktivační funkce
NN	–	Neural network - neuronová síť
ANN	–	Artificial neural network - umělá neuronová síť
FFNN	–	Feedforward Neural Network - dopředná neuronová síť
CNN	–	Convolutional neural network - konvoluční neuronová síť
RNN	–	Recurrent neural network - rekurentní neuronová síť
LSTM	–	Long short-term memory - dlouhá krátkodobá paměť
AI	–	Artificial intelligence - umělá inteligence
ML	–	Machine learning - strojové učení
DL	–	Deep learning - hluboké učení
RoI	–	Region of interest - oblast zájmu
PAF	–	Part affinity field - pole propojení klíčových bodů
	–	
ReLU	–	Rectified Linear Unit
LeakyReLU	–	Leaky Rectified Linear Unit
ELU	–	Exponential Linear Unit
SELU	–	Scaled Exponential Linear Unit
GELU	–	Gaussian Error Linear Unit
GD	–	Gradient Descent
SGD	–	Stochastic Gradient Descent
MBSGD	–	Mini-Batch Stochastic Gradient Descent
NAG	–	Nesterov Accelerated Gradient
AdaGrad	–	Adaptive Gradient
RMSprop	–	Root Mean Square Propagation
Adam	–	Adaptive Moment Estimation
AdamW	–	Adam with Weight Decay
Nadam	–	Nesterov-accelerated Adaptive Moment Estimation
BP	–	Backpropagation

BN	–	Batch Normalization
DO	–	Dropout
LR	–	Learning Rate
MSE	–	Mean Squared Error
BCE	–	Binary Cross-Entropy
CCE	–	Categorical Cross-Entropy
TL	–	Transfer Learning
FT	–	Fine-Tuning
WD	–	Weight Decay
ES	–	Early Stopping
LRS	–	Learning Rate Scheduling
CG	–	Conjugate Gradient
QN	–	Quasi-Newton Methods

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Kamerové systémy jsou využívány již mnoho let a jejich využití je stále širší. Dnes se odhaduje, že celkový počet bezpečnostních kamer ve světě přesahuje miliardu. Využívány jsou v průmyslu, dopravě, obchodě, veřejných prostorech, zdravotnictví či domácnostech.

Zpočátku bylo možné video sledovat pouze živě, později, s příchodem videokazet, bylo možné záznam sledovat až po události. Digitální éra a síťové kamery umožnily přístup ke kamerovým záznamům z libovolného místa na světě. V poslední době se také začalo nahrazovat živé sledování automatickým zpracováním obrazu a detekcí událostí s využitím technik umělé inteligence.

Kamerové systémy se používají zejména ve dvou oblastech: zabezpečení (. security), myšleno jako ochrana před úmyslnými hrozbami a protiprávními činy, jako jsou krádeže, poškozování majetku, či neoprávněný vstup; a bezpečnost (ang. safety), což zahrnuje ochranu před nehodami a náhodnými hrozbami, jako jsou pády, požár, úniky nebezpečných látek, či porušování bezpečnostních předpisů.

Jak již bylo zmíněno, lze kamery využívat jednak pro živé sledování, jednak pro záznam a jeho analýzu po události. Kamerové záznamy jsou zejména důležité pro zpětnou analýzu incidentů, důkazní materiál pro soudní spory, zjišťování příčin nehod, či pro zlepšení bezpečnostních opatření. Živé sledování videa se pak snaží incidentům přímo předcházet. Bylo však prokázáno, že schopnost lidského pozorovatele detekovat nebezpečí se velmi snižuje s délkou sledování a s počtem monitorovaných kamer. Právě proto se s příchodem technik umělé inteligence začalo využívat automatické zpracování obrazu a detekce hrozeb, nebezpečí, nebo již probíhajících incidentů v jejích počátcích. Tyto techniky pak úplně nahrazují lidského pozorovatele, nebo mu pomáhají včas zpozorovat nebezpečí a zareagovat.

Automatická analýza obrazu je používaná již několik desítek let, většinou ale spíše pro oblast zabezpečení, než pro bezpečnost. To z toho důvodu, že úlohy, jako identifikace neoprávněného vstupu, detekce zbraní, rozpoznávání SPZ nebo podezřelých osob jsou pro algoritmy mnohem jednodušší, než například detekce pádu, nouzové situace či zdravotního problému. Hlavním problémem těchto komplexnějších analýz je vysoká falešná pozitivita, kdy je například těžké rozeznat člověka trénujícího běh od člověka utíkajícího před nebezpečím. Nicméně rozvoj v oblasti hlubokého učení a

konvolučních neuronových sítí, jako i vývoj a dostupnost hardwaru podporujícího tyto techniky, umožňuje dneska využít je i pro složitější úlohy.

Ve firmě Linde jsou kamerové systémy používány v mnoha průmyslových provozech, nicméně chybí ucelený systém pro automatickou analýzu obrazu a detekci různých druhů nebezpečí. Naším úkolem tedy v budoucnu bude navrhnout a implementovat modulární systém s možností sledování konkrétních nebezpečí na konkrétních místech. Ty budou zahrnovat například detekci pádu, požáru, zdravotních problémů, nebo porušování bezpečnostních opatření. Systém pak bude v případě rozpoznání nějaké hrozby informovat příslušného pracovníka.

V této práci se zaměříme pouze na jednu z těchto úloh, a to na detekci pádu. Pád může mít různé příčiny, ať už je to zdravotní problém jako ztráta vědomí, nebo zakopnutí. Někdy se zdá, že samotné zakopnutí je banální problém, nicméně pokud se na pracovišti nenachází nikdo, kdo by mohl pomoct, a poškozený není schopen sám přivolat pomoc, může vést takový incident k vážným následkům.

V první části práce se budeme zabývat teoretickými základy, jako je obecná architektura neuronových sítí či princip konvolučních neuronových sítí. V dalších kapitolách se zaměříme na detekci osob a odhad jejich klíčových bodů. Projdeme si různé přístupy a otestujeme různé algoritmy s ohledem na výkon, možnou hardwarovou akcelerací a preciznost. V další části se budeme zabývat samotnou detekcí pádu, tedy algoritmem, který na základě odhadnutých klíčových bodů určí, zda došlo k pádu, či nikoliv. V závěru práce se zaměříme na otestování výsledného řešení a zhodnocení jeho výkonu.

Kapitola 2

Neuronové sítě

Umělá neuronová síť (ang. artificial neural network - ANN) nebo jen neuronová síť (ang. neural network - NN) je výpočetní model inspirovaný biologickými nervovými systémy v lidském mozku. Na rozdíl od konvenčních výpočetních modelů, které zpracovávají informace algoritmicky, a tedy postupují dle předem určeného postupu, se informace v tomto modelu šíří paralelně v síti vah mezi jednotlivými neurony. Jelikož je výstup ze sítě dané architektury závislý hlavně na numerických parametrech, zejména váhách jednotlivých spojů mezi neurony, lze funkčnost sítě měnit bez změny programu pouhou změnou těchto parametrů, a to i automaticky v procesu trénování modelu.

Nyní krátce projdeme historií vývoje neuronových sítí.

2.1 Historie

2.1.1 Prvopočátky

První matematický model neuronové sítě byl popsán v roce 1943 dvěma neurofyziology - Warrenem McCullochem a Walterem Pittsem. [1] Model byl založen na síti jednoduchých logických prvků, které provedou vážený součet svých vstupů a na výstup odešle signál založený na prahové funkcí.

V roce 1958 pak Frank Rosenblatt představil elektronický model neuronové sítě. Základní jednotku, postavenou na McCulloch-Pittsově modelu, nazval perceptron. [2] Jeho architektura byla podobná modelu znázorněnému na obrázku **??**, kde aktivační funkce je prahová funkce. Rosenblattův stroj - Mark I Perceptron - byl postavený pro rozpoznávání jednoduchých vzorů v obrazech. Hlavním omezením tohoto modelu bylo, že byl schopen rozlišovat pouze lineárně separovatelné třídy. Samotný model perceptronu je dodnes používán jako základ pro mnoho neuronových sítí.

Další systém - ADALINE (Adaptive Linear Neuron) - byl představen Bernardem Widrowem a Tedym Hoffem v roce 1960. Tento model umělého neuronu byl velmi podobný perceptronu, na rozdíl od něj ale neobsahoval prahovou ale lineární funkci, výstup tedy nebyl binární ale spojitý. Pro

učení pak byla využita metoda nejmenších čtverců, která minimalizovala chybu mezi skutečným a očekávaným výstupem. [3]

I když ve svých počátcích přitahoval koncept umělé inteligence mnoho vědců jako i sponzorů, v následujících létech zájem ochabl, jelikož nebylo dosaženo předpokládaných výsledků, hlavně s ohledem na tehdejší stav vývoje hardwaru a obecně výpočetní techniky. Proto se tomuto období někdy říká Ai Winter. Neznamená to ale, že ti, kteří se oboru nadále věnovali, nedosáhli významných výsledků. [3]

2.1.2 Backpropagation

Významným milníkem v historii neuronových sítí byl objev algoritmu backpropagation, zvaného taky algoritmus zpětného šíření chyby. Tento algoritmus byl vyvinut v roce 1974 Paulem Werbosem, popularitu ale dosáhl až po nezávislém objevení v roce 1986 Davidem Rumelhartem et al. [4]

Tento algoritmus umožnil trénovat sítě s více vrstvami, což položilo základ hlubokému učení. Algoritmus využívá metodu gradientního sestupu v kombinaci s řetězovým pravidlem derivací k nalezení optimálních vah sítě vedoucích k minimalizaci chyby.

Vynález backpropagation byl jedním z hlavních důvodů, proč se v 80. letech obnovil zájem o neuronové sítě a umělou inteligenci obecně. V 1989 roce taky umožnil Yann LeCunovi at al. zefektivnit použití konvolučních neuronových sítí pro rozpoznávání rukou psaných číslic [5] a položit tak základ širokému využití konvolučních sítí v oblasti počítačového vidění.

2.2 Struktura neuronové sítě

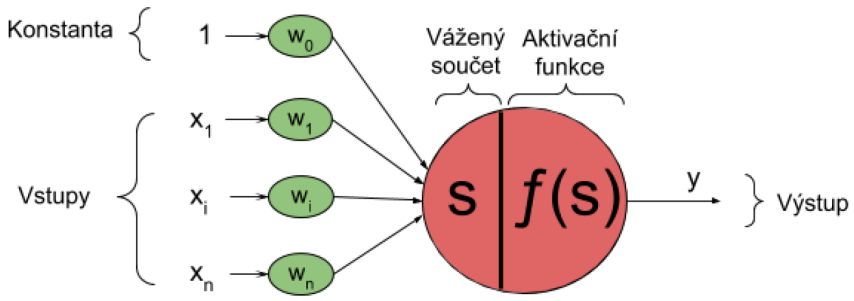
Umělé neuronové sítě jsou silně inspirované biologickými neuronovými sítěmi. A i když napodobení celé funkčnosti lidského nervového systému by bylo velmi složité - ne-li nereálné, zejména s ohledem na počet neuronů a způsob jejich propojení, je možné simulovat alespoň některé funkce lidské mysli.

Pro provádění výpočtů využívají neuronové sítě distribuovaný, paralelní přístup. Informace jsou tedy zpracovány, předávány a ukládány celou sítí, nikoliv pomocí určitých paměťových buněk. Většina znalostí je uložena v silách vazeb mezi jednotlivými neurony. Vazby, které vedou k úspěšnému řešení problému, jsou posilovány, naopak ty, které vedou k neúspěchu, jsou oslabovány.

Podstatnou vlastností neuronových sítí je jejich schopnost učení. Tato vlastnost způsobuje, že již není nutná algoritmizace řešené úlohy, ale stačí neuronové síti opakovaně předložit příklady popisující daný problém, podle kterých jsou postupně upravovány síly vazeb v síti. Tato fáze učení pak určuje, jakým způsobem bude síť transformovat vstupní data na výstupní.[6]

2.2.1 Neuron

Biologický neuron se skládá ze tří hlavních částí. Dendrity přijímají vstupní signály. V těle jsou vstupní signály sečteny do jednoho potenciálu, který vede k vybuzení neuronu - zaslání signálu na



Obrázek 2.1: Model umělého neuronu [7]

výstup, pokud potenciál překročí určitou mez. Axanové vlákno pak vede k synapsím, tedy spojům s dalšími neurony. Lidská mysl pak funguje na principu posilování nebo oslabování těchto spojů.

Umělý neuron se snaží tuto funkčnost napodobit, viz obrázek **??**. Jeho vstupy x_i jsou násobeny váhami w_i, které reprezentují sílu daného spoje. Neuron pak tyto vážené vstupy sečte, a na tento součet aplikuje aktivační funkci (AF), která definuje hodnotu výstupu y. V prvním a základním modelu neuronu, perceptronu, je AF prahová funkce s binárním výstupem. Nicméně v praxi se dnes využívají většinou reálné hodnoty a AF je obvykle spojitá. [6] Existuje mnoho různých AF, některé z nich budou popsány v další části.

Kromě vah jednotlivých vstupů neuron obvykle obsahuje ještě tzv. bias (někdy taky práh nebo posun), jehož funkci je posunout vážený součet vstupů tak, aby bylo možné modelovat i funkce, které nejsou nulové v počátku souřadnic. Je buď reprezentován jako samostatný parametr, nebo jako váha konstantního vstupu s hodnotou 1, jako na obrázku **??**.

Funkci umělého neuronu lze tedy formálně vyjádřit takto:

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right)$$

kde w_0 je bias a $x_0 = 1$, anebo s osamostatněným biasem takto:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

kde b je bias.

Obecně tvoří všechny vstupní váhy a bias množinu parametrů, které ovlivňují funkčnost celé neuronové sítě. Proces trénování sítě pak spočívá v nalezení optimálních hodnot těchto parametrů, které vedou k co nejmenší chybě při řešení úlohy.

2.2.2 Základní aktivační funkce

AF hraje stěžejní roli v umělých neuronových sítích zavedením nelinearity do celého systému a umožňuje tak učení se složitějších vzorců.

V průběhu let bylo vyvinuto mnoho typů AF, a i když jejich úloha se zdá být podobná, můžou se od sebe výrazně lišit. Jejich rozdíly spočívají zejména v oboru hodnot, spojitosti, monotónnosti, a v tom, zda je závislá na přídavných trénovaných parametrech. Ve výsledku se taky liší i jejich využití. Nyní projdeme několik základních AF, od kterých se většina ostatních nějakým způsobem odvíjí.

Sigmoida (lineární křivka) funkce transformuje vstup do rozmezí $0 \div 1$, je tak vhodná pro odhad pravděpodobnosti. Proto se taky někdy používá ve výstupních vrstvách síti, zejména pro binární klasifikaci. Její funkčnost lze formálně zapsat takto:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Její nevýhodou je hlavně problém mizejícího gradientu (ang. vanishing gradient), kdy zejména ve vícevrstvých sítích se velikost změn váh v počátečních a koncových vrstvách významně liší. To pak způsobuje nestabilitu v procesu trénování a může jej zpomalit nebo zcela zastavit. Navíc to, že není nulová v počátku souřadnic, může způsobit špatnou konvergenci.

Hyperbolický tangens (tanh) je velmi podobný sigmoidě, ale transformuje vstup do rozmezí -1 $\div$ 1. Řeší tedy poslední zmíněný problém. Nicméně se pořád potýká s problémem mizejícího gradientu. Taky, obě tyto funkce představují větší výpočetní nároky. Formálně lze tuto AF popsat takto:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU (Rectified Linear Unit, taky někdy rampa) je jednoduchá a efektivní AF. Pro kladné hodnoty se chová jako identita, pro záporné je nulová.

$$f(x) = \max(0, x) = \begin{cases} x & x > 0, \\ 0 & x \leq 0, \end{cases}$$

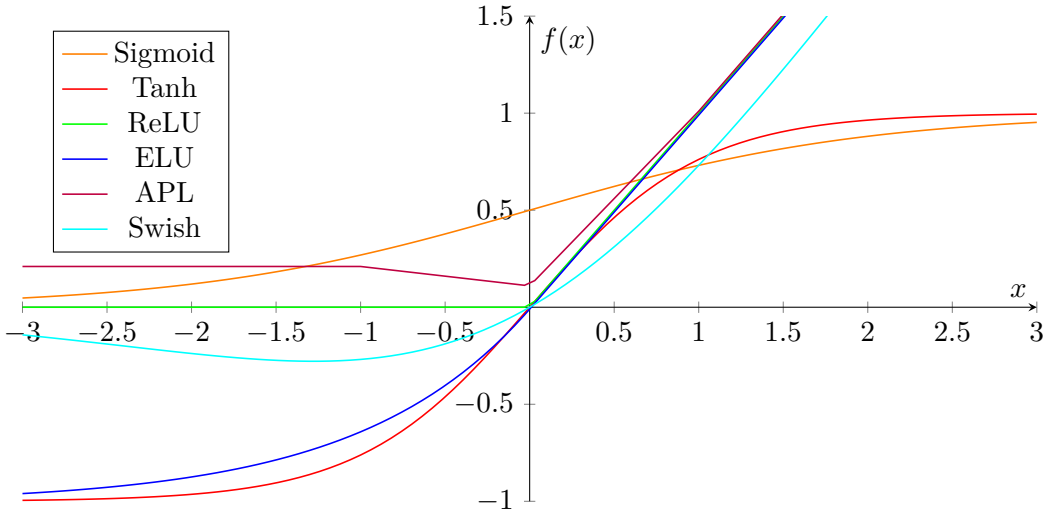
Jelikož je výpočetně velmi nenáročná, je tato AF velmi oblíbená v hlubokých sítích. Její nevýhodou ale je, že nezohledňuje záporné hodnoty, což v jejich případě vede k problému mizejícího gradientu a může způsobit tzv. "mrtvé neurony". Tento problém řeší různé varianty ReLU, jako například PReLU (Parametric ReLU) nebo LReLU (Leaky ReLU). Tyto varianty přidávají parametr p vynásobený x pro záporné hodnoty:

$$f(x) = \max(0, x) = \begin{cases} x & x > 0, \\ p \cdot x & x \leq 0 \end{cases}$$

LReLU má tento parametr fixně nastavený na $p = 0.01$, zatímco v případě PReLU je tento parametr trénovaný spolu s jinými parametry sítě.

Další alternativou k ReLU je ELU (Exponential Linear Unit), která v záporné části odpovídá exponenciální funkci:

$$f(x) = \begin{cases} x & x > 0, \\ a\,(e^x - 1) & x \leq 0 \end{cases}$$



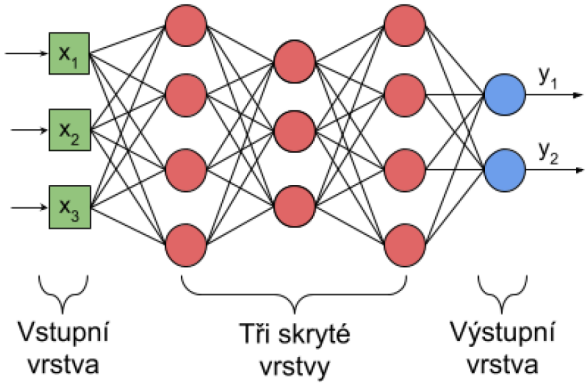
2.2.3 Dělení neuronových sítí

Neuronové sítě jsou dnes využívány v mnoha oblastech a dokážou řešit mnoho různých úloh, nicméně neexistuje jediný typ sítě, který by dokázal řešit všechny. V průběhu let proto bylo vyvinuto mnoho různých architektur sítí, každá pro jiné využití.

Jedním ze základních způsobů, jak můžeme neuronové sítě rozdělit, je podle typu učení: učení s učitelem (supervised) a učení bez učitele (unsupervised). V případě učení s učitelem, předkládáme síti dvojice vstupů a očekávaných výstupů, na jejichž základě se síť snaží minimalizovat chybu úpravou svých parametrů. Oproti tomu, u učení bez učitele nemá síť k dispozici očekávané výstupy, ale snaží se najít nějaké struktury v datech, například shluky. V další části se budeme věnovat hlavně neuronovým sítím pro učení s učitelem.

Dále můžeme neuronové sítě rozdělit na dopředné (feedforward NN - FFNN) a rekurentní (recurrent NN - RNN). U dopředných sítí se informace šíří pouze ze vstupu k výstupu a nevyskytují se žádné smyčky. Naopak rekurentní sítě obsahují zpětnou vazbu z výstupu přivedenou na vstup. To umožňuje reagovat na změny v čase.

Další možnosti jejich rozdělení je dle topologie, která zahrnuje jejich hloubku, tzn. počet vrstev, velikost těchto vrstev a jejich vzájemné propojení. V další sekci se budeme věnovat právě uspořádání vrstev v neuronových sítích.



Obrázek 2.2: Vícevrstvá, plně propojená síť [7]

2.3 Topologie neuronových sítí

Nejčastější uspořádání neuronů v neuronových sítích je do vrstev. Neurony dané vrstvy jsou spojeny pouze s neurony z předchozí a následující vrstvy. Vrstvy pak dělíme na tři typy: vstupní, výstupní a skryté. Vstupní vrstva neobsahuje neurony a neprovádí žádné operace, pouze přijímá vnější signály a distribuuje je do další vrstvy. Výstup z neuronů ve výstupní vrstvě pak reprezentuje výstup celé sítě.

Pokud síť obsahuje pouze vstupní a výstupní vrstvu, mluvíme o jednovrstvé neuronové síti. Takové sítě mají velmi omezené možnosti, proto se v praxi nepoužívají. Většina neuronových sítí má mezi vstupní a výstupní vrstvou alespoň jednu skrytou vrstvu, viz obrázek **??**.

U většiny klasických dopředných neuronových sítí jsou jednotlivé vrstvy mezi sebou plně propojeny, tzn. každý prvek jedné vrstvy je propojený se všemi prvky následující vrstvy.

2.4 Proces trénování s využitím backpropagation

Jak již bylo řečeno, proces trénování NN spočívá v nalezení optimálních hodnot parametrů jednotlivých neuronů. Optimální parametry pak vedou k minimální chybě. Chybou rozumíme rozdíl mezi skutečným výstupem sítě a očekávaným výstupem. K tomu se nejčastěji používá algoritmus backpropagation (taky algoritmus zpětného šíření chyby). Nyní si popíšeme, jak trénování sítě pomocí tohoto algoritmu funguje.

Nejprve se ze vstupních dat vypočítají pomocí aktuálních parametrů sítě reálně výstupy sítě. Tento proces se nazývá dopředný průchod (ang. forward pass). Následně se pomocí chybové funkce (ang. loss function, taky nákladová funkce - ang. loss function) spočítá chyba sítě. Ta vyjadřuje, v jaké míře se skutečné výstupy liší od očekávaných. Můžou být použity různé chybové funkce, v závislosti na typu úlohy, kterou síť řeší.

V klasifikačních úlohách je nejčastěji používaná chybová funkce křížové entropie, která porovnává rozdělení pravděpodobnosti skutečného výstupu sítě s očekávaným rozdělením pravděpodobnosti. Pro regresní úlohy se používá například střední kvadratická chyba (ang. mean squared error - MSE) vyjadřující střední hodnotu druhých mocnin rozdílů mezi skutečnými a očekávanými hodnotami. Další možností je absolutní chyba (ang. mean absolute error - MAE), která vyjadřuje střední hodnotu absolutních hodnot rozdílů.

Dále je používaná metoda gradientního sestupu, k nalezení minima chybové funkce. Za tímto účelem se vypočítají derivace chyby podle jednotlivých parametrů sítě. Tyto derivace pak určují, jakým směrem a jak rychle se mají dané parametry měnit, aby se chyba minimalizovala. Jelikož se derivace parametrů v dané vrstvě vypočítávají pomocí řetězového pravidla derivací (ang. chain rule) podle derivací parametrů následující vrstvy, je tento proces nazýván zpětný průchod (ang. backward pass).

Jednotlivé parametry se pak podle vypočítaných derivací upraví. Velikost změny je určená hyperparametrem rychlostí učení (ang. learning rate, taky krok), který určuje, jak rychle se mají parametry měnit. Vypočítaná derivace se vynásobí rychlostí učení a přičte k původní hodnotě parametru. Tento proces se opakuje pro všechny parametry sítě. Metoda gradientního sestupu umožňuje větší změny parametrů, když jsou daleko od minima - absolutní hodnoty derivací jsou větší, a naopak menší změny, když se k němu blíží - derivace se blíží nule.

Proces trénování sítě se skládá z opakování dopředného a zpětného průchodu pro všechna trénovací data (někdy postupně pro jejich podmnožiny - dávky, ang. batches). Po každém průchodu se upraví parametry sítě podle vypočítaných derivací. Tento proces se opakuje, dokud chyba nedosáhne požadované úrovně, nenastane její konvergence nebo není překročen maximální počet iterací.

2.5 Optimalizace procesu trénování

V základní verzi algoritmu backpropagation se využívá výše popsaný gradientní sestup. Ten ale může mít některé problémy, které mohou zpomalit proces trénování nebo jej někdy zcela znemožnit. Nyní si popíšeme některé z těchto problémů a jejich možná řešení.

Jedním z podstatných problémů gradientního sestupu je, že se může lehce zaseknout v lokálním minimu, kde je derivace nulová. To může způsobit, že se síť zastaví v nějakém suboptimálním bodě a nepokračuje do globálního minima, které by odpovídalo optimálnímu řešení. Tento problém se často řeší přidáním tzv. momentu do úpravy parametrů. Tento proces bere v úvahu i předchozí změny parametrů. V případě, že se derivace parametru v průběhu trénování změní, moment umožňuje parametrům ještě nějakou dobu pokračovat v pohybu ve stejném směru. To většinou pomůže překonat lokální minima a dosáhnout tak globálního minima.

Dalším řešením tohoto problému je využití stochastické aproximace gradientního sestupu (ang. stochastic gradient descent - SGD), kde se gradient počítá pro náhodně vybrané podmnožiny trénovacích dat. Tento postup umožňuje rychlejší konvergenci, počítáme totiž gradient jen pro část

dat, a zároveň zabraňuje zaseknutí v lokálním minimu zavedením šumu do procesu trénování. Tato metoda se využívá nejčastěji pro velké datasety.

Dalším problémem může být nastavení optimálního kroku učení. Pokud je krok příliš velký, může dojít k příliš velké změně, která opomine minimum, můžeme tak nikdy nedosáhnout konvergence. Naopak, pokud je krok příliš malý, může trénování trvat příliš dlouho. Řešením může být například adaptivní nastavení kroku. Nejznámější taková metoda je RMSProp (ang. root mean square propagation), která upravuje krok učení pro každý parametr podle průměrného druhého momentu gradientu. Další možností je v průběhu trénování postupně snižovat krok (ang. learning rate decay).

Populárním řešením těchto problémů je také Adam (ang. adaptive moment estimation, v překladu adaptivní odhad momentu), který kombinuje zavedení momentu a adaptivního nastavení kroku pomocí RMSProp.

Další metodou, jak optimalizovat nastavení kroku, je normalizace dat mezi vrstvami sítě (ang. batch normalization). Tím se zamezí příliš velkým změnám v jednotlivých vrstvách, které někdy destabilizují proces trénování.

U trénování hlubokých sítí se často naráží také na problém přetrénování (taky nadměrné přizpůsobení, ang. overfitting). Přetrénování nastává, když se síť dobře naučí trénovací data, zároveň ale postrádá schopnost generalizace, a když pak dostane nová data, nedosahuje dobrých výsledků. K základním řešením tohoto problému patří použití většího množství trénovacích dat - pokud jsou dostupná, jinak se někdy zavádí umělé variace dat jako rotace či převrácení, jinak je třeba někdy zvážit zjednodušení architektury modelu. Dále pak jsou využívány techniky regularizace, které upravují samotný proces trénování.

Nejjednodušší regularizační technikou je tzv. předčasné zastavení, tedy zastavení trénování v případě, že se chyba na validačních datech začne zvyšovat. Další možností je dropout (taky výpadek), kdy se u určitého počtu náhodně zvolených neuronů v průběhu trénování nastaví na výstupu nula. Tím se snižuje závislost sítě na konkrétních neuronech a zvyšuje se tak její schopnost generalizace.

Další dvě metody, nazývané L1 a L2 regularizace, přičítají k chybové funkci člen, který penalizuje velikost parametrů sítě. L1 regularizace (taky metoda Lasso, z ang. least absolute shrinkage and selection operator) přičítá k chybové funkci součet absolutních hodnot všech parametrů sítě vynásobený hyperparametrem, který určuje míru této penalizace. Tato metoda vede k řídké síti, ve které je mnoho parametrů nulových. L2 regularizace (taky hřebenová regrese, ang. ridge regression) přičítá k chybové funkci součet druhých mocnin všech parametrů sítě vynásobený hyperparametrem λ. Tato metoda vede jednak k menší variabilitě parametrů, jednak k pomalejším změnám parametrů v průběhů trénování, v důsledku pak k menší citlivosti na šum v datech. Využívá se zejména v hlubokých sítích.

Kapitola 3

Konvoluční neuronové sítě

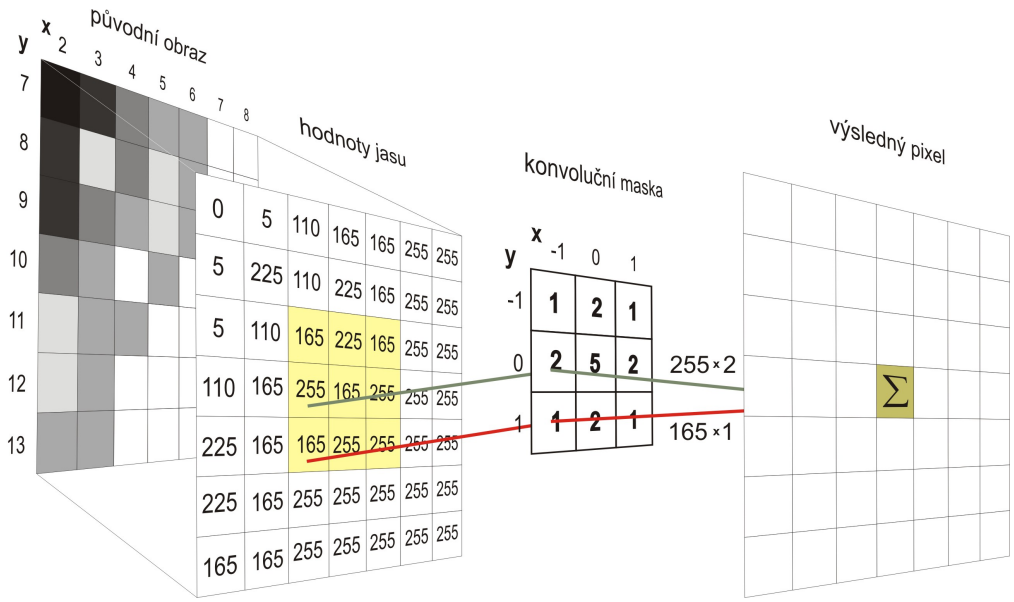
I když byly jedny z prvních NN použity ke zpracování obrazu, brzy se ukázalo, že pro zpracování obrazu s větším rozlišením a větším množstvím kanálů je klasická architektura NN velmi neefektivní. Bylo tedy třeba vytvořit jinou architekturu, která by efektivně zpracovávala obrazová data. Nejznámější takovou architekturou je konvoluční neuronová síť (ang. convolutional neural network - CNN), která bude popsána v této kapitole.

3.1 Problémy zpracování obrazu pomocí klasických neuronových sítí

Klasickým přístupem pro zpracování obrazů pomocí neuronové sítě je vytvoření sítě se stejným počtem prvků vstupní vrstvy, jako je počet pixelů vstupního obrazu (za předpokladu jednoho kanálu). Tato vrstva je pak plně propojena s několika skrytými vrstvami, výstupní vrstva pak buď vrací kategorie, v případě klasifikace, nebo hodnoty hledaných atributů, jako je lokalizace objektů či souřadnice klíčových bodů, v případě regrese.

Problémem tohoto přístupu je rozměr vstupních dat takové sítě. Běžná velikost obrazu v dnešní době dosahuje několika miliónů pixelů, tento počet je ale ještě vynásoben počtem kanálů, většinou třemi (RGB). To znamená, že velikost vstupní vrstvy sítě je velmi vysoká, a aby byla taková síť efektivní, musí mít větší počet skrytých vrstev a neuronů v těchto vrstvách. Ve výsledku má taková síť velký počet parametrů, které je třeba natrénovat. Trénování takové sítě je možné, je ale velmi nepraktické. Bylo by potřeba velké množství trénovacích dat, jinak by s velkou pravděpodobností došlo k přetrénování sítě. I kdyby ale bylo k dispozici dostatečné množství trénovacích dat, bylo by jak trénování, tak i používání sítě velmi výpočetně i paměťově náročné.

CNN se proto tento problém řeší tak, že se snaží zredukovat rozměr vstupních dat pomocí konvolučních vrstev, které jsou schopny efektivně zpracovávat obrazová data.



Obrázek 3.1: Princip diskrétní dvourozměrné konvoluce [8]

3.2 Konvoluce

V kontextu počítačové grafiky je konvoluce binární operace, kdy pro daný pixel sečteme hodnoty pixelů v jeho okolí vynásobené váhami vyjádřenými maskou, která se nazývá jádro konvoluce (ang. kernel). Výsledný obraz je taky nazýván konvoluce. Z matematického pohledu se jedná o diskrétní dvourozměrnou konvoluci - binární operaci diskrétních funkcí, která je definována následovně:

$$(h * f)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} h(x - i, y - j) \cdot f(i, j)$$

kde h je vstupní obraz, f je jádro konvoluce, velikost jádra je $2k + 1 \times 2k + 1$, a x a y jsou souřadnice pixelu, ke kterému se jádro aplikuje.

Hodnota konvoluce na dané pozici je tedy suma součinů hodnot pixelů vstupního obrazu a hodnot váh vyjádřených jádrem konvoluce položeným středem na danou pozici, viz obrázek **??**. Nejčastěji je velikost jádra lichá, jelikož je pak jednodušší určení středu jádra.

Konvoluce je velmi rozšířená operace v počítačové grafice a zpracování obrazu, k nejpoužívanějším aplikacím patří například rozmazání nebo detekce hran.

Pro rozmazání se používá například průměrovací filtr (ang. box blur) využívající jádro, které má na všech pozicích hodnotu $1/n$, kde n je počet prvků jádra, viz obrázek **??**. Výsledná hodnota konvoluce daného pixelu je tedy rovná průměru hodnot tohoto pixelu a jeho sousedů. Dalším často využívaným filtrem je Gaussovo rozostření, viz obrázek **??**. Rozmazání je někdy využíváno v počítačovém vidění jako první krok, za účelem odstranění šumu z obrazu.

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}, \quad \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$$

Obrázek 3.2: Jádro konvoluce pro průměrovací a Gaussovo rozostření

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}, \quad \begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix},$$

Obrázek 3.3: Jádro konvoluce pro detekci vertikálních a horizontálních hran využívané pro Sobelův operátor

Pro detekci hran se často používá Sobelův operátor, který využívá dvě jádra - pro vertikální a horizontální hrany. Jádra pro detekci hran jsou zobrazena na obrázku **??**.

V případě více kanálů vstupního obrazu musí být připraveno zvlášť jádro pro každý kanál. Výsledné konvoluce se pak sečtou. Můžeme se na tuto situaci dívat i jako na třírozměrné konvoluční jádro s rozměrem třetí dimenze rovným počtu kanálů vstupního obrazu. Konvoluce je pak prováděná tak, že na každé pozici vstupního obrazu provedeme součet vah jádra vynásobených hodnotami vstupního obrazu na příslušných pozicích napříč všemi kanály. Výstup takové konvoluce má pouze jeden kanál.

Myšlenkou konvolučních neuronových sítí je nahradit plně propojené vrstvy konvolučními vrstvami. Ruční nastavení všech hodnot jader tak, aby konvoluční vrstvy extrahovaly požadované vlastnosti z obrázku, je ale velmi obtížné, v případě složitějších či obecnějších problémů téměř nemožné. V 1989 proto Yann LeCun et al. navrhli metodu, jak se síť může hodnoty jader naučit sama, a tak si efektivně vytvořit i složitější filtry, které by člověk těžko navrhl ručně. Zjistili, že váhy konvolučních jader se mohou trénovat pomocí algoritmu backpropagation stejně jako váhy neuronů v plně propojených vrstvách.

Takový přístup má mnoho výhod. Konvoluce se dá velmi dobře paralelizovat a zajistit tak vysokou efektivitu výpočtů. Oproti plně propojeným vrstvám má vždy konvoluce s jedním jádrem počet vah pouze rovný velikosti jádra. I v případě provedení mnoha konvolucí je počet výrazně menší než v případě potřebného počtu a velikosti plně propojených vrstev. Zároveň lze pomocí konvolucí efektivně extrahovat různé vlastnosti ze vstupního obrazu, ty pak pomocí plně propojených vrstev zpracovat a využít k řešení problému. Proto se taky výstupu konvoluce často říká mapa příznaků (ang. feature map).

3.3 Krok a padding

Pro pochopení funkčnosti konvolučních neuronových sítí je ještě důležité pochopit dva parametry, které ovlivňují výstup konvoluční vrstvy. Jedná se o krok (ang. stride) a padding (česky vycpávka či doplnění).

Krok určuje, co kolik pixelů se aplikuje jádro konvoluce na vstupní obraz. Pokud tedy bude krok $k = 1$, bude jádro aplikováno na každý pixel vstupního obrazu. Pokud bude krok $k = 2$, bude jádro aplikováno na každý druhý pixel vstupního obrazu. Pokud bude krok $k > 1$, bude výstup konvoluce násobně menší.

Jelikož na nejkrajnější pixely nemůžeme aplikovat jádro, protože by přesahovalo hranice obrazu, bude se obraz z každou konvolucí nežádaně zmenšovat i při jednotkovém kroku. Pokud použijeme například jádro o velikosti 3×3, bude výsledná mapa o 2 řádky a 2 sloupce menší, než byl vstupní obraz. Dalším problémem je, že ve výsledné konvoluci je marginalizován vliv krajních pixelů. Obraz je tedy jistým způsobem oříznut. Tyto problémy se proto někdy řeší pomocí tzv. paddingu. Je to technika, kdy se vstupní obraz rozšíří o takový počet pixelů z každé strany, aby vzhledem k velikosti konvolučního jádra bylo možné aplikovat jádro i na krajní pixely obrazu.

Velikost výstupní mapy konvoluce je tedy dána vztahem:

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor$$

kde n je velikost vstupního obrazu, p je velikost paddingu, s je krok a jádro je velikosti $f \times f$.

3.4 Konvoluční vrstva

Konvoluční vrstva je základním prvkem konvoluční neuronové sítě. V jistém slova smyslu je podobná plně propojené vrstvě, jelikož obsahuje váhy, biasy a aktivační funkce. Namísto plného propojení s neurony následující vrstvy je ale aplikována konvoluce na vstupní data. K výstupní mapě je přičten bias a následně může být aplikována aktivační funkce.

Jedná vrstva může mít několik konvolučních jader, každé s vlastními váhami a biasem. Je třeba zároveň pamatovat, že každé jádro musí mít hloubku rovnou počtu vstupních map, resp. počtu kanálu vstupního obrazu v případě vstupní vrstvy. Konvoluci pak aplikujeme zvlášť pro každé jádro, počet výstupních map příznaků tedy bude roven počtu jader v dané vrstvě. V praxi bude každé jádro vyjadřovat jinou vlastnost, kterou se snažíme ze vstupního obrazu extrahovat.

Množinu parametrů dané konvoluční vrstvy tedy tvoří hodnoty jader a jejích příslušné biasy. K hyperparametrům pak patří počet jader a jejich velikost, aktivační funkce, krok a padding.

3.5 Poolovací vrstva

Jak již bylo zmíněno, v konvolučních vrstvách vzniká vícero map příznaků vyjadřující různé vlastnosti. Tím se ale množství dat zvětšuje, což porušuje samotnou myšlenku konvolučních sítí, která je založena na snaze zredukovat rozměr dat. Proto se snažíme rozměr map příznaků zmenšovat. Jak již bylo zmíněno, dojde k redukci rozměrů v konvoluční vrstvě, pokud použijeme krok $k > 1$. Častěji ale je k tomuto účelu prováděno podvzorkování dat v tzv. poolovacích vrstvách (z ang. pooling layer).

V poolovací vrstvě je vstupní mapa rozdělená do stejně velkých čtvercových oblastí velikosti $t \times t$, na základě hodnot v daném čtverci je pak vytvořen jeden pixel výstupní mapy. K nejčastěji používaným metodám patří max-pooling a average-pooling. Max-pooling vybere z dané oblasti největší hodnotu, zatímco average-pooling vyhodnotí z každé oblasti průměr hodnot. Velikost výstupní mapy pro vstup velikosti $n \times n$ a poolovací oblasti velikosti $t \times t$ je tedy $\lfloor \frac{n}{t} \rfloor \times \lfloor \frac{n}{t} \rfloor$.

3.6 Architektura CNN

Konvoluční neuronové sítě se skládají ze dvou částí - konvoluční části a plně propojené části.

Konvoluční část se skládá z několika konvolučních vrstev proplétaných s poolovacími vrstvami. V této části se pracuje s mapami příznaků. Její výstupem je soubor map příznaků, resp. mapa příznaků s větší hloubkou, která vyjadřuje různé vlastnosti vstupního obrazu.

Plně propojená část je pak klasická dopředná neuronová síť, která na základě extrahovaných vlastností provádí klasifikační či regresní úlohu. Před vstupem do plně propojené části jsou mapy příznaků převedené do jednoho vektoru o velikosti rovné součtu velikostí všech map příznaků.

Kapitola 4

Analýza problematiky detekce pádu

V této kapitole stanovíme co je přesně naším cílem a zamyslíme se, jak k našemu problému přistoupit.

Naším úkolem bude v reálném čase z videostreamu detekovat pád osoby. Pád osoby definujeme jako náhle, neúmyslné klesnutí těla z výškové pozice (např. stání, chůze nebo sezení) na zem nebo jinou nižší úroveň, přičemž tato osoba nemá kontrolu nad tímto pohybem. Samozřejmě nejsme vždy schopni úplně dobře rozeznat, zda se nejedná o úmyslné klesnutí, např. prudké lehnutí.

Dle některých definic (zejména ve zdravotnictví) se o pád nejedná, pokud jde o důsledek závažné vnitřní příhody (např. mrtvice). V našem případě toto nerozlišujeme, naopak chceme detekovat jak pády v důsledku ztráty rovnováhy či vlivem vnějších faktorů (např. zakopnutí, převrácení těžkým předmětem), tak pády v důsledku akutních události vlivem zdravotních problémů, jako jsou např. mrtvice, záchvaty, mdloby či jiné důvody ztráty vědomí.

Cílem této práce je navrhnout algoritmus, který bude detekovat, zda je ve vstupní sekvenci snímku některá osoba, jejíž pozice je klasifikována jako pád. Hlavním cílem výsledného programu bude alarmovat příslušného pracovníka, pokud osoba upadne a zůstane v ležící pozici. To nám dá možnost odfiltrovat falešné alarmy v případě sehnutí čí pokud bude osoba špatně viditelná a algoritmus tak špatně vyhodnotí její pohyb. Tímto postprocesingem se ale teď nebudeme zabývat, spíše se zaměříme na samotnou klasifikaci pozice.

Stejně jako u detekce objektů, viz **??**, bychom mohli i pro detekci pádu vytvořit vhodnou konvoluční síť, která by přímo z obrázku definovala, zda se jedná o pád nebo ne. U detekce se už dnes sice s ohledem na pokrok hardwaru tento přístup používá, nicméně se jedná o velmi náročný úkol, který vyžaduje rozsáhlou optimalizaci, pokročilou architekturu a velké množství trénovacích dat. Nicméně, pokud by se podařilo takovouto síť natrénovat, mohla by lépe detekovat některé situace např. podle výrazu tváře.

V našem případě tedy budeme v prvním kroku pomocí vhodné neuronové sítě detekovat pozici osoby ve formě klíčových bodů, na jejich základě pak další neuronová síť vyhodnotí, zda se jedná o pád. To úlohu velice zjednoduší, jelikož místo analyzování tisíců pixelů, budeme analyzovat pár

desítek klíčových bodů. Další výhodou je, že u takového postupu jsme schopní použít techniky, kdy sledujeme změny pózy v čase, což by bylo mnohem složitější s jednofázovou konvoluční síti.

Další alternativou by mohlo být pouze detekovat osoby jako objekty, a na základě jejich bounding boxů určit, zda se jedná o pád. Tento postup by byl jednodušší na dvou úrovních. Jednak je detekce objektů méně náročná úloha než detekce pózy, jednak bychom ve druhé fázi analyzovali pouze několik parametrů bounding boxu (rozměry a velikost) oproti pár desítkám klíčových bodů. Nicméně, pokud se nad tím zamyslíme, ne vždy vypovídají parametry bounding boxů o pozici člověka. Tento postup by tak pravděpodobně vedl k mnohem méně přesnému výsledku, než analýza klíčových bodů, kdy může síť analyzovat takové vzorce jako je např. délka končetin v pohledu či úhel mezi nimi.

V další kapitole rozebraná problematika detekce pózy a bude zvolen algoritmus pro detekci klíčových bodů. Dále se pak budeme zabývat vývojem modelu detekujícího pád na základě těchto klíčových bodů.

Kapitola 5

Výběr algoritmu pro detekci pózy

Jelikož je dnes dostupných mnoho různých algoritmů či natrénovaných modelů pro detekci pózy osob v obrázku či videu, nemá smysl pro naše řešení implementovat takovýto algoritmus od nuly. Možné by to samozřejmě bylo, i vzhledem k dostupnosti otevřených trénovacích dat (např. dataset COCO [9]), nicméně bychom pravděpodobně nedosáhli kvalitních výsledků, jako řešení, která jsou výsledkem mnoholetých výzkumů. Hlavně pak bychom těžko dosáhli výkonů těchto řešení, a ten je pro nás stěžejní, jelikož potřebujeme video zpracovávat v reálném čase.

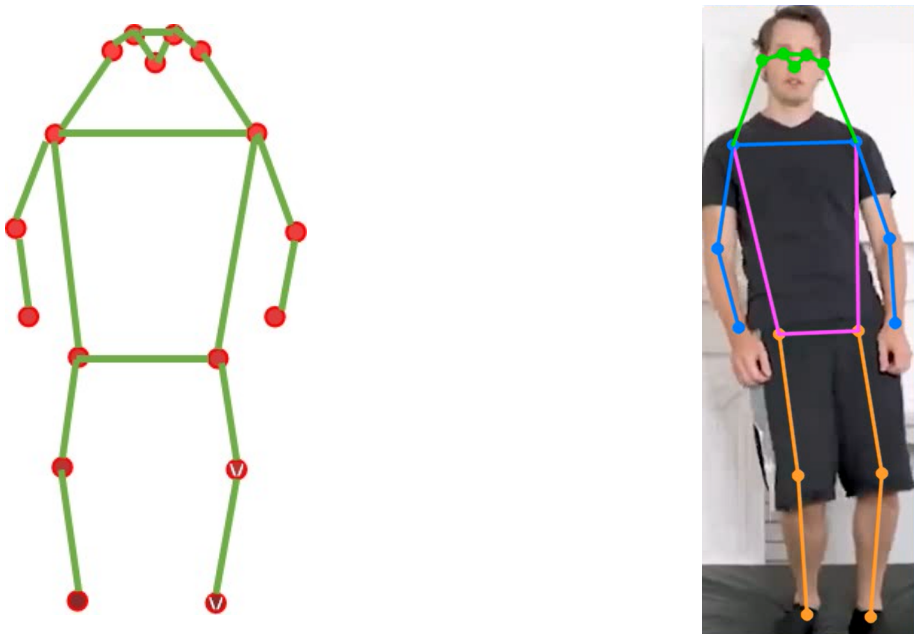
V následující kapitole budou popsány obecné principy detekce osob a jejich pózy v obraze. Následně budou popsány některé populární algoritmy pro detekci pózy se zaměřením na jejich specifika. Několik z nich pak bude otestováno, výsledky budou porovnány, a na jejich základě bude zvolen algoritmus použitý v konečném řešení detekce pádu.

5.1 Detekce pózy

Úloha detekce pózy spočívá v nalezení klíčových bodů postavy v obraze. Může se jednat také o zvíře, v našem případě se ale budeme zabývat pouze klíčovými body lidské postavy. Klíčové body představují důležité body lidského těla, znalost jejich lokalizace nám umožňuje analyzovat pózu dané osoby, popřípadě sledovat její pohyb. K základním klíčovým bodům patří hlava, ramena, lokty, zápěstí, kyčle, kolena a kotníky, viz obrázek **??**. Některé algoritmy dokážou rozeznat i orientaci dlaně či stopy, nebo rozpoznat klíčové body na hlavě, jako jsou ústa, nos, oči a uši [11].

Klíčové body jsou většinou reprezentovány jako dvojice souřadnic (x, y) vzhledem k celému obrazu, některé algoritmy poskytují i souřadnice normalizované vzhledem k bounding boxu osoby. Existují také algoritmy pro 3D souřadnice, těmito se ale nebudeme zabývat, i když by mohly stanovit zajímavou alternativu, zejména pokud by pro detekci bylo použito více kamer z různých pohledů.

V oblasti algoritmů pro detekci pózy existují dva základní přístupy: zdola nahoru a shora dolů. Přístup zdola nahoru se snaží detekovat všechny klíčové body v obraze, aniž by rozlišoval jednotlivé osoby, pokud je algoritmus schopen detekce pózy pro více osob, pak v dalším kroku tyto body



Obrázek 5.1: (Vlevo) Topologie klíčových bodů použitá např. v COCO-pose.[10] (Vpravo) Příklad detekce pózy pomocí YOLO.

spojuje do jednotlivých postav. Naproti tomu přístup shora dolů nejprve detekuje všechny osoby v obraze, v jejich rámci pak detekuje klíčové body.

5.2 Detekce klíčových bodů

5.2.1 Heatmapy

U obou výše zmíněných přístupů se nejčastěji provádí vyhledání všech klíčových bodů pomocí tzv. heatmap. Je to 2D mapa pravděpodobnosti, že se v daném bodě vyskytuje nějaký klíčový bod. Maximální hodnoty v této mapě pak představují lokalizaci klíčových bodů.

Pro vygenerování heatmap se používá konvoluční neuronová síť. Pro každý klíčový bod, resp. pro každý typ klíčového bodu k (v případě detekce pózy více osob) vzniká jedna heatmapa. Jako referenční heatmapy pro trénování se používají mapy, kde je klíčový bod reprezentován 2D Gaussovým rozložením s vrcholem v místě daného bodu.

V dalším kroku jsou z heatmap vygenerovány, nejčastěji s pomocí algoritmu argmax, souřadnice klíčových bodů. V případě vícero osob je pak třeba tyto body spojit do jednotlivých osob.

5.2.2 Regrese

Využití heatmap je velmi přesné, nicméně z důvodu nutnosti provádění dvou sekvenčních výpočtů je taky trochu pomalé. Taky komplikují proces trénování, jelikož musíme spolu s trénovacími daty

dodat modelu i heatmapy. Některé algoritmy se proto snaží formulovat úlohu jako regresi vedoucí přímo k souřadnicím klíčových bodů. Tento přístup je ve své podstatě trochu méně přesný, nicméně je rychlejší.

Vůbec první algoritmus pro detekci pózy využívající hluboké učení, DeepPose [12], který byl vytvořen v roce 2014 společností Google, používal právě regresi. Také algoritmus YOLO používá regresi pro určení souřadnic klíčových bodů, nicméně detekce je prováděná pro detekované objekty, nikoliv nad celým vstupním obrazem. [13]

5.3 Detekce objektů a osob v obraze

Detekce osob se v podstatě může generalizovat na detekci objektů v obraze. Detekci objektů v obraze definujeme jako úlohu, kdy ve vstupním obrázku určíme lokaci a třídu všech hledaných objektů.

V kapitole **??** jsme si popsali základní architekturu konvolučních neuronových sítí, ta se ale většinou v praxi používá pro klasifikaci obrázků, nikoliv pro detekci objektů - algoritmus tedy pouze určí, o jakou třídu objektu se jedná, a ideálně potřebuje, aby objekt vyplňoval celý vstupní obraz. Teoreticky by bylo možné detekci formulovat jako regresní problém a natrénovat takovou síť, která by pomoci několika konvolučních vrstev následovaných několika plně propojenými vrstvami byla schopna predikovat lokalizaci a třídu všech objektů v obraze. [14] Problém detekce je ale velice komplexní a taky by vyžadoval velice komplexní síť - více vrstev s mnoha filtry, resp. neurony. Jak již ale bylo v zmiňováno, komplexnost sítě zvyšuje její nároky na výpočetní výkon a komplikuje nebo úplně znemožňuje její trénování s ohledem na pravděpodobnost přetrénování.

Snahou tedy je najít metody, které poupraví funkčnost sítě tak, aby byla schopna efektivní detekce objektů. Většina těchto metod se nějakým způsobem snaží rozdělit vstupní obrázek na menší části, ty následně jednak klasifikovat, a tedy určit, zda se v dané lokalitě vyskytuje objekt, popřípadě pomocí regrese určit jeho přesnou lokalizaci. Lokalizace je většinou reprezentována jako souřadnice obdélníku ohraničujícího daný objekt, tzv. bounding box. Rozdělení může být provedeno přímo na vstupním obrázku nebo na mapě příznaků v rámci sítě. Taky můžeme buď rozdělit obrázek na pevně dané oblasti (např. do mřížky) - jednofázový přístup, anebo v jedné fázi předpřípravit množinu oblastí a ve druhé fázi nad těmito oblastmi provést klasifikaci a regresi - dvoufázový přístup.

5.3.1 Sliding window

Jednou z prvních takových metod byl tzv. sliding window (klouzavé okno), který aplikuje hrubou sílu. Vstupní obrázek se postupně projíždí oknem o fixní velikosti. Vznikne tak množina pokrývající každou možnou lokaci objektů. Na tyto oblasti se pak aplikuje klasifikační algoritmus. Postup se opakuje pro několik velikostí okna, aby se detekovalo objekty různé velikosti.

Tento postup je ale velice pomalý, jelikož je pro každý obrázek zvolený velký počet oblastí, pro které je třeba provést klasifikaci popřípadě regresi. Navíc je většina těchto oblastí prázdná, a dochází tak k plýtvání výpočetním výkonem. Algoritmus se taky potýká s překrývajícími se objekty.

Další metody se tedy snaží redukovat počet oblastí, na které se aplikuje klasifikace, tak, že se vybere pouze oblasti, které pravděpodobně budou obsahovat nějaký objekt.

5.3.2 Dvoufázový přístup

5.3.2.1 R-CNN

Prvním algoritmem, který efektivně zredukoval počet oblastí pro klasifikaci, byl algoritmus R-CNN (Region-based Convolutional Network). [15] Tento algoritmus nejprve použil některou z dostupných metod (autoři použili selective search) pro vygenerování navržených oblastí (region proposals), které pravděpodobně obsahují nějaký objekt. Tyto metody jsou nezávislé na třídě objektů. Algoritmus tedy vygeneruje zhruba 2000 oblastí, vzniklé obrázky jsou následně upraveny na velikost požadovanou CNN v další fázi. CNN extrahuje z dané oblasti mapu příznaků, na její základě plně propojené vrstvy predikují třídu objektu popřípadě jeho bounding box.

Problémem R-CNN je, že výběr oblasti a jejich následná klasifikace jsou nezávislé úlohy a jsou nezávisle trénovány. Detekce objektu je taky poměrně pomalá, protože je extrakce příznaků prováděná pro všechny oblasti zvlášť. Tyto problémy se snaží řešit další upravené verze R-CNN.

5.3.2.2 Fast R-CNN

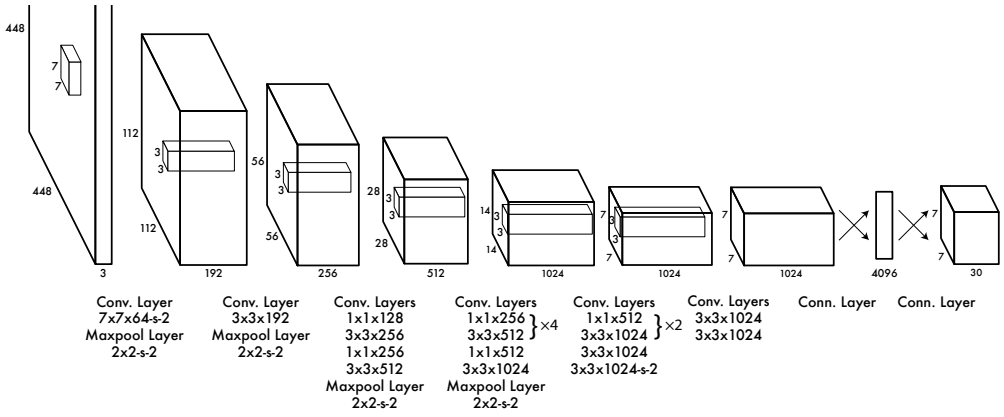
První z nich je Fast R-CNN [16], která je upravená tak, aby bylo možné provádět trénování v jednom kroku. Taky extrahuje příznaky pro celý vstupní obraz najednou, pomocí selective search pak identifikuje oblasti zájmu (ang. region of interest - RoI), které následně použije pro klasifikaci a regresi. Tato metoda je přesnější a asi desetkrát rychlejší než původní R-CNN.

5.3.2.3 Faster R-CNN

Další algoritmus, Faster R-CNN [17], nahrazuje metodu selective search vlastní, plně konvoluční sítí RPN (region proposal network). Zefektivňuje tak proces trénování, výsledná síť je také rychlejší a přesnější než Fast R-CNN.

5.3.2.4 Mask R-CNN

Mask R-CNN [18] rozšiřuje Faster R-CNN o segmentaci. Segmentace je úloha, kdy je každému pixelu vstupního obrazu přiřazena třída. K tomu ale tvůrci museli upravit pooling vrstvu nastupující po RPN tak, aby se výstupy algoritmu shodovaly se vstupy s přesností pixelu. Dále přidali na konci procesu CNN určující třídu pro každý proces. Výstupem jsou masky pro každou třídu, určující, které pixely tuto třídu reprezentují.



Obrázek 5.2: Architektura původní verze YOLO [19]

Dále je možné s malou úpravou použít tento algoritmus i pro detekci pózy. Jako trénovací data použijeme masky, kdy jedině v místě daného klíčového bodu je indikována třída reprezentující tento klíčový bod.

5.3.3 Jednofázový přístup

Jednofázový přístup se snaží najít řešení, ve kterém není nutné hledat navržené oblasti, ale provést klasifikaci a regresi na předem dané množině oblastí, obvykle určené mřížkou.

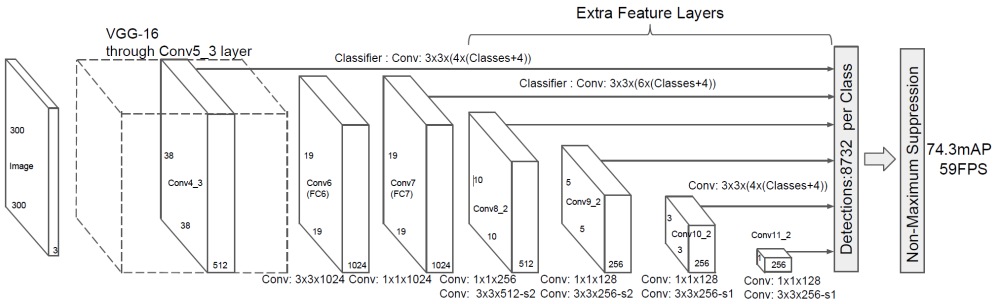
5.3.3.1 YOLO

Prvním takovým algoritmem byl YOLO (taky YOLOv1, z ang. you only look once) [19]. Ten, v původní verzi, rozdělí vstupní obraz do pevně dané mřížky velikosti $S \times S$ a v každém z těchto polí určí B bounding boxů a jejich třídu. V původní verzi bylo zvoleno $S = 7$ a $B = 2$.

Obraz je nejprve zpracován pomocí konvolučních vrstev, které extrahují mapu znaků o velikosti $S \times S \times K$, kde K je počet kanálů. Každý pixel této mapy představuje jedno pole mřížky. Dále je mapa zpracována plně propojenými vrstvami, které provádějí nad každým polem mřížky klasifikaci a regresi, viz obrázek **??**.

Každý bounding box je reprezentován souřadnicemi středu a velikosti (šířka a výška). Dohromady s informací o jistotě detekce bounding boxu (confidence score) vrátí model 5 informací o každém bounding boxu. Pro každé pole mřížky pak určí společnou informaci o třídě všech objektů v daném poli. Pokud objekt není detekován, třída indikuje pozadí a souřadnice bounding boxu jsou ignorovány. Velikost výstupního vektoru je tedy $7 \times 7 \times (2 * 5 + C)$, kde C je počet definovaných tříd - v původní verzi pouze 20.

Tento algoritmus, navržený v roce 2015 J. Redmonem et al., byl revolučně rychlý, zároveň v porovnání s jinými real-time detekčními algoritmy dosahoval i slušné přesnosti. Nicméně byl



Obrázek 5.3: Architektura SSD [22]

velice citlivý na velikost objektu a přesnost detekce, zejména u menších objektů, byla horší než u dvoufázových algoritmů.

Další verze algoritmu YOLO přinesly postupná vylepšení ve formě optimalizace trénování a architektury. YOLOv2 [20] zavedl mj. trénování na několika měřítkách a byl natrénován s 9000 třídami (proto taky nazýván YOLO9000). YOLOv3 [21] přinesl mj. detekci na několika měřítkách. Postupně byla taky zvětšována mřížka a měnila se použitá architektura CNN sítě sloužící pro extrakci příznaků pro jednotlivá pole mřížky. Postupně taky byly přidávány další funkce jako je segmentace, detekce pózy či sledování objektů (ang. tracking).

V 2020 roce firma Ultralytics poprvé implementovala YOLO s využitím populární knihovny PyTorch (YOLOv5), což umožnilo snadnější využití YOLO v praxi. Firma Ultralytics taky vytvořila framework pro použití různých verzí YOLO (YOLOv3 a novější). Taky pracuje na dalších vylepšeních a optimalizacích. Konkrétně vytvořila YOLOv5 (2020), YOLOv8 (2023) a YOLOv11 (2024). Tyto verze nicméně nejsou podloženy odbornými články, někteří je tak považují za neoficiální verze.

5.3.3.2 SSD

Dalším populárním algoritmem, který používá jednofázový přístup, je SSD (z ang. single shot detector). [22] Ten rozdělí vstupní obraz do několika mřížek o různé velikosti. Postup je takový, že nejprve projde obraz konvoluční sítí, konkrétně sítí VGG16, která extrahuje mapu příznaků. Tu se postupně dalšími konvolučními vrstvami zmenšuje, výstup každého stádia zmenšení, reprezentující mřížku dané velikosti, se spolu s původní mapou dále zpracovává plně propojenou sítí.

Výstupní bounding boxy nejsou, jako v případě YOLO, pouze výsledkem regrese, ale pro každé pole dané mřížky je definováno několik výchozích oblastí (ang. default box), ze kterých jsou vybrány ty, které obsahují objekt. K ním je predikována třída objektu a posun i změna velikosti výchozí oblasti, upřesňující výsledný bounding box.

V době svého vzniku byl SSD rychlejší a přesnější než YOLO, ale novější verze YOLO jej už předběhly. Nicméně některé principy SSD, jako výchozí oblasti či použití různých měřítek, byly převzaty do novějších verzí YOLO.

5.4 Charakteristiky vybraných algoritmů pro detekci pózy

V této sekci bude popsáno několik populárních algoritmů pro detekci pózy zejména se zaměřením na jejich rychlost, přesnost a specifika architektury.

Velká část algoritmů byla zamítnutá a neproběhlo ani jejích testování. Nejčastějším důvodem zamítnutí některého z algoritmů je, že schází jeho volně dostupná, aktualizovaná implementace. Tvůrci algoritmů většinou investují čas a zdroje pro vývoj algoritmu a natrénování modelu (často pro akademické účely), pak ale neinvestují do jeho údržby. Zejména v prostředí Pythonu, kde se neustále vyvíjejí nové knihovny a zpětná kompatibilita starších verzí není zaručena, je pak obtížné použít takovéto řešení ve svém projektu, aniž by bylo nutné investovat další čas do pochopení zdrojového kódu a jeho úpravy. Jak již bylo zmíněno na počátku kapitoly, implementace algoritmu od nuly by vyžadovala velké množství času a zdrojů (zejména výpočetních), hlavně by taky byla potřebná hlubší znalost problematiky. Někdy je možné řešení použít za cenu kompromisu ve formě použití starších verzí knihoven či Pythonu, může to ale představovat bezpečnostní rizika.

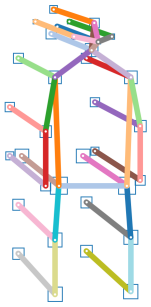
5.4.1 DeepPose

DeepPose je historicky první algoritmus pro detekci pózy využívající hluboké učení. Vyvinuli jej Alexander Toshev a Christian Szegedy ze společnosti Google v roce 2014. [12] Algoritmus předpokládá, že se ve vstupním obraze nachází pouze jedna osoba. Síť se snaží v jednom kroku pomocí regrese jak detekovat osobu, tak i její klíčové body. Jelikož je těžké takto dosáhnout velmi přesných výsledků, algoritmus používá další fázi, která pomocí regrese provádí posun bodů k přesnějším výsledkům. Tato fáze je aplikována opakovaně, kaskádně se tak zvyšuje přesnost detekce.

Při svém vzniku byl DeepPose revoluční, nicméně v porovnání s dnešními řešeními je poměrně pomalý a nepřesný. Nicméně položil základ pro využití hlubokého učení v oblasti detekce pózy.

5.4.2 OpenPose

OpenPose [23] je typicky příklad přístupu zdola nahoru. Jeho výhodou je ale možnost vyhledání více osob v jednom snímku. Tento algoritmus, který vyvinuli v roce 2019 Zhe Cao et al., nejprve pomocí CNN vytvoří heatmapu pro každý typ klíčového bodu. Pro spojení bodů do jednotlivých osob využije pole propojení klíčových bodů (ang. part affinity field - PAF). PAF je mapa vytvořená pro každou končetinu (myšleno obecně spojení dvou klíčových bodů), která v oblasti dané končetiny obsahuje hodnoty určující směr z jednoho bodu do druhého. Pokud pak dáme do hromady informace z heatmap a z PAF, jsme schopni poměrně jednoznačně zkompletovat jednotlivé klíčové body do celých postav. Stejně jako heatmapy, jsou i PAF součástí trénovacích dat.



Obrázek 5.4: vizualizace sledování osoby mezi dvěma snímky v OpenPifPaf [24]

5.4.3 OpenPifPaf

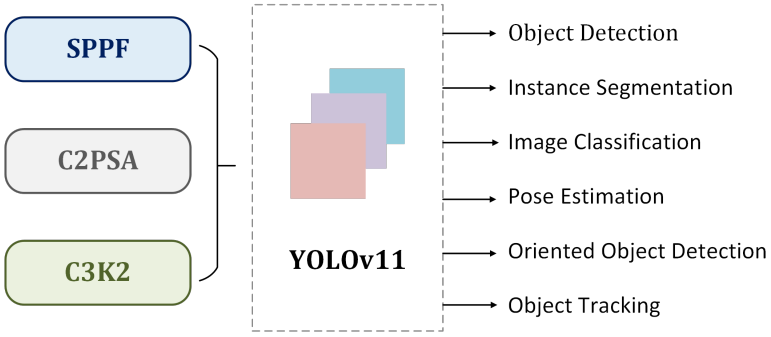
Algoritmus OpenPifPaf [24], vyvinutý v roce 2021 Svenem Kreissem et al., je v podstatě vylepšenou verzí OpenPose. Jeho název je odvozen od dvou stavebních kamenů: PIF (Part Intensity Field) - pole intenzity klíčových bodů, a PAF (Part Affinity Field) - pole propojení klíčových bodů. PIF je rozšířením heatmap, kdy kromě intenzity pravděpodobnosti klíčového bodu obsahuje i jeho posun, zaručující přesnější lokalizaci bodu, a odhadovanou velikost dané části těla. PAF v OpenPifPaf je taky podobný tomu v OpenPose, navíc ale indikuje kromě směru i velikost dané končetiny, což umožňuje lepší prostorové zachycení pózy.

Dalším rozšířením oproti OpenPose je možnost sledování osob ve videu. Mapa příznaků, která je výsledkem vstupní CNN, je udržována v mězipaměti, do další části sítě pak vždy vstupují mapy pro aktuální a předchozí snímek. Výstupem pak kromě klíčových bodů v každém snímku a jejich propojení tvořící kostru, jsou i propojení mezi klíčovými body z jednotlivých snímků, viz **??**. Algoritmus si pak udržuje ID sledovaných osob, pokud k dříve nalezené osobě je nalezena nová pozice, je jí přiřazeno stejné ID. Pokud je nalezena nová osoba, je jí přiřazeno nové ID.

5.4.4 MediaPipe - BlazePose

MediaPipe je framework vyvinutý společností Google, umožňující jednoduchou integraci různých technik strojového učení. Obsahuje různé algoritmy pro řešení úloh jako detekce objektů, segmentace či detekce klíčových bodů (tváře či pózy). MediaPipe je optimalizován pro mobilní zařízení a webové aplikace. Detekce pózy v tomto frameworku je postavená na algoritmu BlazePose.

BlazePose implementuje přístup shora dolů, detekuje tedy nejprve RoI, ve kterých detekuje osobu a její pózu. Nativně podporuje pouze jednu osobu ve snímku. Ve videu ale v rámci optimalizace neprovádí detekci RoI pro každý snímek, pouze pokud v aktuální RoI již není detekována osoba. Výhodou tohoto algoritmu je, že detekuje 33 bodů v postavě, což je podstatně více, než většina ostatních algoritmů, umožňuje tak přesnější analýzu některých situací, např. podle natočení tváře, dlaní či stop.



Obrázek 5.5: Architektura YOLOv11 [26]

Framework MediaPipe implementuje BlazePose spolu s detekcí více osob (použije v první fáze detektor objektů) i sledování. Výhodou tohoto frameworku je jeho kontinuální vývoj a jednoduchost integrace. Nevýhodou ale je, že pro systémy Windows není implementována podpora GPU. Jelikož náš výsledný produkt bude spouštěn primárně na Windows zařízeních, je to pro nás tato vlastnost rozhodující.

5.4.5 Torchvision

Torchvision je knihovna, která je součástí frameworku PyTorch. Obsahuje různé nástroje pro strojové vidění, jako je detekce objektů či segmentace. Její součástí je i předtrénovaný model pro detekci pózy, který je založený na algoritmu Mask R-CNN.

5.4.6 YOLO

Od vydání YOLOv7 v roce 2022 integruje framework YOLO i detekci pózy. Oficiální článek Chien-Yao Wanga et al. [25] sice neobsahoval tuto funkčnost, ale oficiální implementace zahrnula i implementaci YOLO-Pose [13]. Obecně detekce pózy v YOLO kombinuje přístup shora dolů a zdola nahoru. Algoritmus sice vyhledává klíčové body spolu s bounding boxy osob, nicméně vše v jednom kroku. Samotná detekce klíčových bodů využívá regresi, což zjednodušuje proces trénování, jelikož není třeba tvořit heatmapy.

Architektura použita v YOLO-Pose se ale liší od architektury používané v pozdějších verzích. V YOLO-Pose jsou na konci řetězce umístěny hlavy pro různá měřítka, jejich výstupem jsou bounding boxy a klíčové body, oba tvořené spolu. V pozdějších verzích je architektura YOLO koncipována universálněji pro různé úlohy. Obsahuje tak tři fáze: páteř (ang. backbone), která extrahuje mapu příznaků, krk (ang. neck), který přizpůsobuje mapu příznaků pro různá měřítka, a hlavy (ang. head), které paralelně zpracovávají výstupy pro různé úlohy, viz obrázek **??**. [26] Bounding box a klíčové body jsou tedy sice generovány paralelně a teoreticky nezávisle, nicméně s ohledem na proces trénování a postprocessing se v praxi navzájem výrazně ovlivňují.

Nejnovější verze YOLO taky podporují kombinaci detekce klíčových bodů a sledování osob. Model tedy kromě klíčových bodů vrací ID dané osoby, pomocí kterého můžeme spojit danou postavu s předchozími snímky. Můžeme tak efektivně analyzovat pohyby jednotlivých osob.

Kapitola 6

Klasifikace pózy

Problematika vyhodnocování je velmi široká a přináší mnoho problémů. Ostatně i člověk někdy může špatně interpretovat chování druhé osoby. Například pokud někdo skáče do postele či jinak prudce lehá, může to vypadat jako nebezpečná situace. Stejně se v počítačovém vidění nevyhneme falešným poplachům, nicméně se budeme snažit zapojit různé techniky pro zlepšení přesnosti našeho detektoru.

Cílem této části práce je vytvořit algoritmus, který pro danou pózu (reprezentovanou klíčovými body), resp. sekvenci takových póz (získané pro jednu osobu ze sekvence snímků), určí, zda se jedná o situaci pádu, či nikoliv. Tento algoritmus bude přijímat vždy pózu jedné osoby, funkcionalita pro více osob bude řešená v další části práce.

6.1 Trénovací data

Pro trénování našeho modelu jsme použili více než 150 videí. Pro videa byly vytvořeny anotace aktuální třídy pózy. Tato anotace není pro každý snímek, ale pouze při změně definuje časovou značku a následující třídu.

Dále byl vytvořen skript, který prošel každé video a vytvořil trénovací data pro další fázi. Ty obsahují pro každý snímek detekované klíčové body (jako vstup) a aktuální třídu (jako požadovaný výstup). Pro detekci byl použit dříve vybraný algoritmus pro detekci pózy. Na použitém algoritmu by teoreticky nemuselo záležet (pokud detekuje stejné typy klíčových bodů), je ale lepší použít stejný algoritmus jak pro trénovací data, tak ve výsledném programu. Algoritmy se totiž můžou v některých situacích chovat trochu jinak (např. okluze) a náš model by tak dostával v praxi jiná data, než pro jaké byl natrénován.

Pro trénovací data jsme použili 3 třídy, ty odpovídají třem různým třídám pózy, které nás zajímají - *normální*, kdy osoba např. chodí, sedí nebo stojí, *padá* - přechodný stav padání, definován od započatí pohybu směrem dolů, a *upadl* - definován od momentu, kdy se dotkl země trupem nebo všemi končetinami.

Pro náš model obecně potřebujeme jenom dvě třídy - *normální* a *upadl*. Nicméně budeme pro náš model experimentovat i s třetí třídou - *padá*, která pomůže síti hlouběji pochopit problematiku a přesněji rozeznat některé situace, zejména pak v případě využití rekurentních neuronových sítí.

6.2 Dopředná neuronová síť

Nejjednodušší architekturou, kterou můžeme pro náš model použít, je dopředná neuronová síť. Tento model pak bude klasifikovat jednotlivé pózy, aniž by znal jejich kontext. Síť bude klasifikovat klíčové body pouze podle aktuální lokalizace ve snímku, nikoliv podle pohybu.

Výhodou této architektury je jednoduchost, potažmo rychlost. Síť nepotřebuje mnoho parametrů a oproti rekurentním sítím potřebuje pro evaluaci pouze jeden dopředný průchod vrstvami sítě.

Další výhodou je jednoduchost trénování a používání. V případě více osob pro samotnou klasifikaci pádu není nutné sledování osob. Můžeme jednoduše klasifikovat všechny detekované pózy, aniž bychom řešili, které osobě patří.

Tato síť ale ve výsledku bude klasifikovat pózy pouze dle vzájemného umístění jednotlivých klíčových bodů, potažmo délky končetin, nebude ale brát v úvahu natočení postavy. To proto, že postavy ve snímku vystupují pod různým úhlem natočení v závislosti na natočení kamery. Naopak síť, která je schopná sledovat pohyb, bude schopna sledovat mj. i změnu natočení postavy a to bez ohledu na natočení kamery.

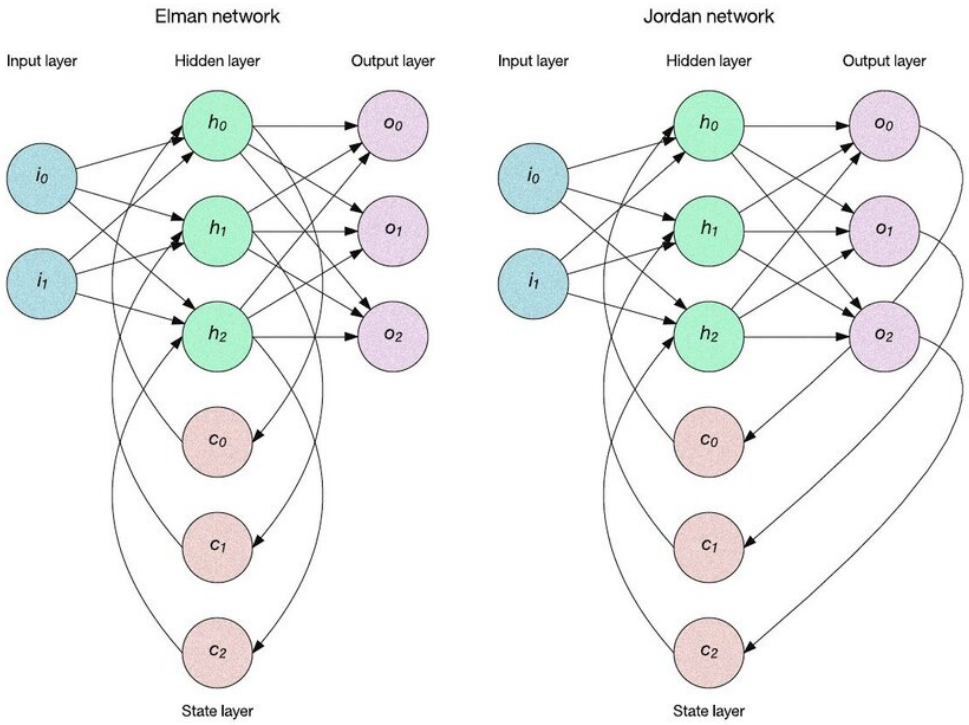
6.3 Rekurentní neuronové sítě

Rekurentní neuronové sítě (ang. Recurrent Neural Networks - RNN) je kategorie neuronových sítí, které do své architektury zapojují zpětnou vazbu. Na rozdíl od dopředných sítí, které spracovávají jednotlivé vstupy nezávisle, rekurentní sítě spolu s aktuálním vstupem při evaluaci zohledňují nějakým způsobem i výsledek předchozí iterace. Jejich využití tedy je ve dvou oblastech: analýza změn pozorovaného objektu v čase (např. sledování pohybu, analýza chování či predikce časových řad) a zpracování kontextuálních informací jako je např. přirozený jazyk.

Data, která v aktuální iteraci přebíráme z předchozí iterace, se často označují jako skrytý stav (ang. hidden state). Je to forma paměti, která se s každou iterací aktualizuje. Často je reprezentován jako stavová vrstva (ang. state layer nebo context layer), která přijímá hodnoty z výstupu neuronů, uchovává je mezi iteracemi a předává je spolu se vstupními daty na vstup neuronů. Na obrázku **??** je tato vrstva reprezentována neurony c_i.

6.3.1 Jednoduché rekurentní sítě

Nejjednodušší forma rekurentní neuronové sítě je NN s jednou skrytou vrstvou; tato vrstva kromě dat ze vstupní vrstvy, přijímá také výstup předchozí iterace buď svých vlastních neuronů, anebo z

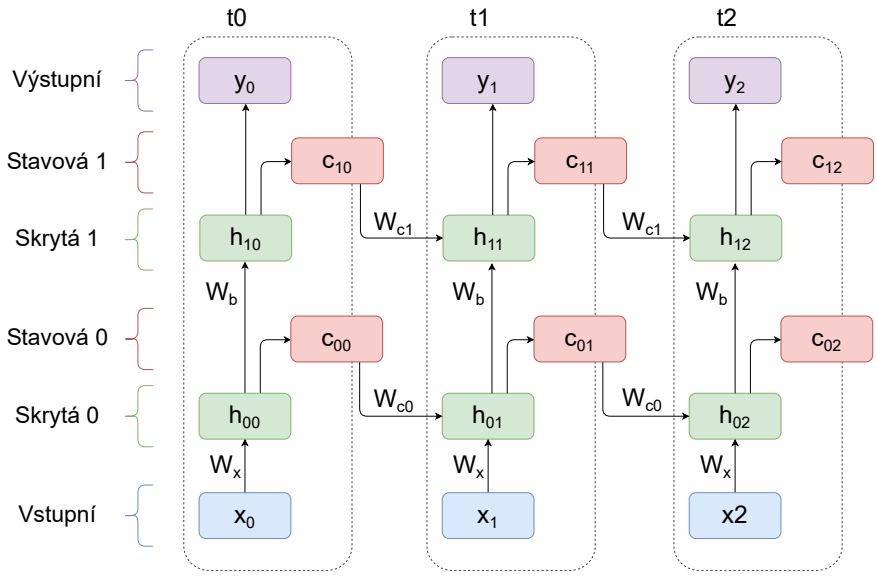


Obrázek 6.1: Základní architektury RNN [29]

neuronů výstupní vrstvy, viz obrázek **??**. Obrázek je zjednodušený, v praxi jsou stavová a skrytá vrstva plně propojeny. Tyto architektury se jmenují Elmanova síť [27], resp. Jordanova síť [28], od jejích tvůrců. Tyto sítě jsou taky známé jako jednoduché rekurentní sítě (ang. Simple Recurrent Networks - SRN). I když pojem rekurentních sítí byl známý už od začátků neuronových sítí jako takových a byly i případy jejích použití, právě tyto sítě patřily k prvním, které používaly pro trénování algoritmus backpropagation. Jednoduché rekurentní sítě uchovávají pouze krátkodobé vzory a jsou vhodné spíše pro jednoduché úlohy, jako je např. predikce časových řad.

6.3.2 Hluboké rekurentní sítě

Stejně jako u dopředných neuronových sítí, kde se od jednoduchého perceptronu přešlo k hlubokým sítím, se i rekurentní sítě rozšířily na více vrstev. V hlubokých rekurentních neuronových sítích (ang. deep RNN - DRNN) jsou pak jednotlivé vrstvy většinou podobné struktuře Elmanovy sítě - zpětná vazba je předávána pouze v rámci jedné vrstvy, nikoliv mezi vrstvami RNN (například z výstupní vrstvy do první skryté vrstvy). Má to několik důvodů. Trénování sítě ze zpětnou vazbou mezi vrstvami by bylo velmi složité a obtížné. Taky, obecně každá vrstva sítě se učí pochopit problém na jiné úrovni abstrakce, zpětná vazba přes několik vrstev by pak mohla narušit stabilitu tohoto procesu a omezit kvalitu učení.



Obrázek 6.2: Unrolling hluboké RNN

6.3.3 Trénování rekurentních sítí

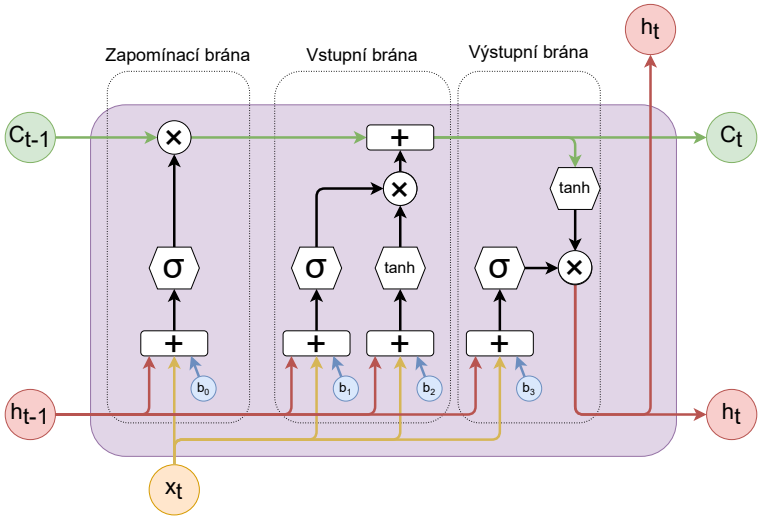
Pro pochopení rekurentních neuronových sítí je třeba si vysvětlit, jak se trénují. Pro vizualizaci trénování RNN se tyto sítě takzvaně rozbaluje v čase (ang. unrolling). Znamená to, že jednotlivé iterace vizualizujeme jako sekvenci stejných sítí (stejné váhy), které v čase t přijímají vstup x_t a vracejí výstup y_t, viz obrázek **??**. Zároveň místo smyček znázorňujících zpětnou vazbu přijímá skrytá vrstva v čase t stav c_{t-1} z předchozí iterace. Takto je propojená mezi iteracemi každá skrytá vrstva (na obrázku **??** vizualizováno propojení přes stavovou vrstvu).

Při trénování se pak používá algoritmus zpětného šíření chyby v čase (ang. backpropagation through time - BPTT). Algoritmus funguje stejně jako klasický backpropagation, šíří se ale nejenom vrstvami, ale i iteracemi. Unrolling nám pomáhá backpropagation pochopit, jednotlivé iterace totiž jsou naskládány jako vrstvy a celou síť řešíme jako klasickou dopřednou NN.

6.3.4 Problémy mizejícího a explodujícího gradientu

Výše popsané základní rekurentní neuronové sítě, někdy označovány jako vanilla RNN, trpí několika zásadními problémy. U dopředných sítí jsme zmiňovali problém mizejícího gradientu (ang. vanishing gradient), vystupující zejména u hlubších sítí. Ten se projevuje i u RNN a je zesílený tím, že jsou jednotlivé iterace naskládané na sebe, podobně jako vrstvy. Zejména pak u delších sekvencí budou mít dřívější vstupy velmi malý vliv na učení sítě.

U RNN se taky projevuje problém opačný - explodující gradient (ang. exploding gradient). Ten způsobuje, že v průběhu sekvence se váhy začnou exponenciálně zvětšovat a dosáhnou tak nepřiměřeně velkých hodnot.



Obrázek 6.3: (1) Jednotka LSTM, (2) Rozvinutá hluboká LSTM síť

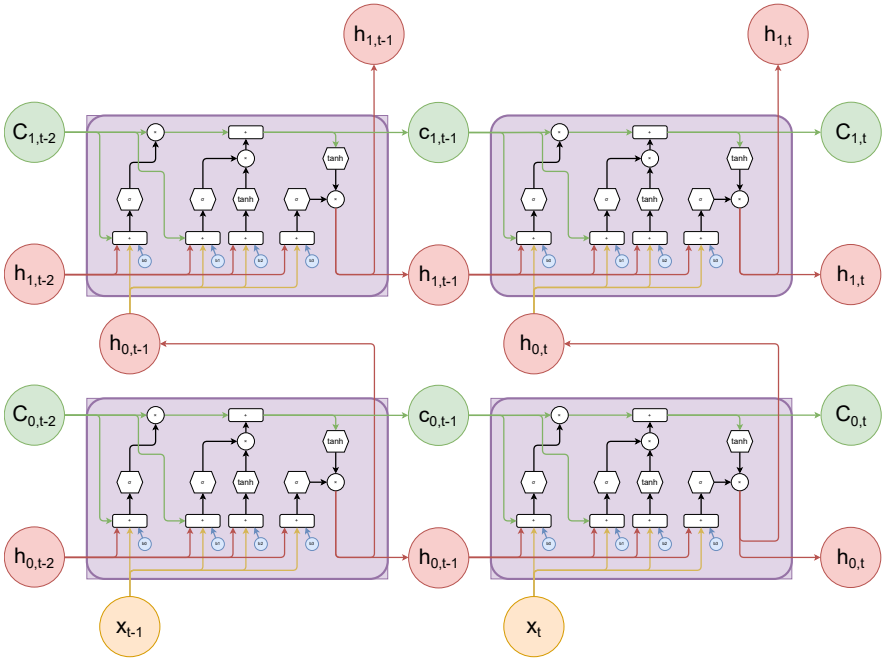
Podívejme se, co přesně tyto problémy způsobuje. Součástí algoritmu backpropagation je počítání parciální derivace ztrátové funkce podle jednotlivých vah. V případě BPTT potřebujeme mimo jiné počítat parciální derivace skrytého stavu mezi jednotlivými iteracemi $\frac{\partial h_{t-1}}{\partial h_t}$. Tyto derivace následně opakovaně násobíme při použití řetězového pravidla. Pokud je tato derivace $\frac{\partial h_{t-1}}{\partial h_t} < 1$, jeho vynásobení bude mít za následek postupné zmenšování gradientu. Pokud budeme například mít sekvencí 100 iteraci, pak i kdyby se gradienty v každé iteraci zmenšovaly $0,9$ krát, po 100 iteracích by gradient klesl na hodnotu $0,9^{100} \approx 2,7 \times 10^{-5}$, což je prakticky nula. Pokud se naopak bude gradient zvětšovat $1,1$ krát, po 100 iteracích by gradient vzrostl na $1,1^{100} \approx 13780$, což způsobí úplnou destabilizaci sítě a nedosáhneme žádného výsledku. Vidíme tedy, že v případě, kdy je $\frac{\partial h_{t-1}}{\partial h_t} > 1$, dochází k explodujícímu gradientu.

Z důvodu těchto problémů byly vyvinuty složitější rekurentní struktury. Jejich architektura je v podstatě podobná, jednotlivé vrstvy jsou ale zastoupeny jinými stavebními bloky, které umožňují zejména širší pochopení kontextu a efektivnější proces trénování. Vanilla RNN se v praxi dnes využívají velmi zřídka.

6.4 LSTM

Dlouhá krátkodobá paměť (ang. long short-term memory - LSTM), představena Hochreiterem a Schmidhuberem v roce 1997, je typ rekurentní neuronové sítě, který byl navržen tak, aby překonal problémy mizejícího a explodujícího gradientu.

Její základem je jednotka, viz obrázek **??**.1, která ve třech stádiích aktualizuje krátkodobou a dlouhodobou paměť. Dlouhodobá paměť je reprezentovaná pomocí stavu buňky (ang. cell state, na obrázku **??**.1 c_t), který je postupně upravován a nakonec předán další iteraci. Dokáže uchovávat



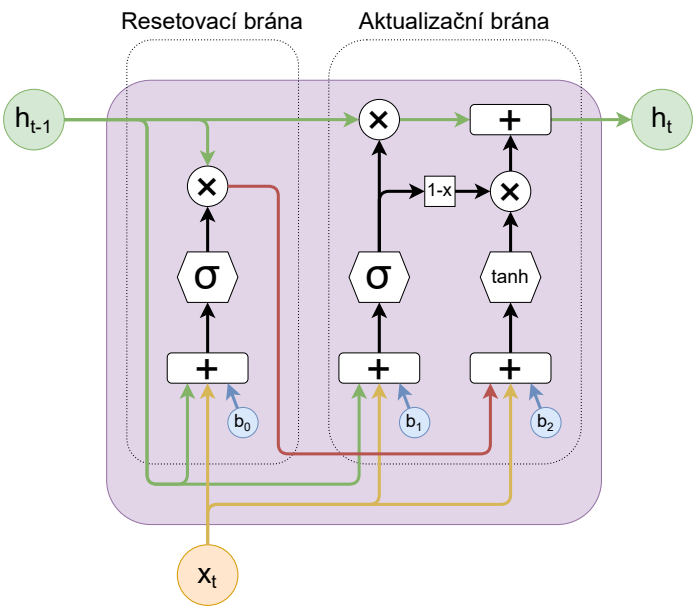
Obrázek 6.4: Rozvinutá hluboká LSTM síť

dlouhodobé závislosti. Krátkodobá paměť je reprezentována pomocí skrytého stavu. Je použita pro úpravu dlouhodobé paměti, v konečném stadiu je ale vždy v rámci dané iterace vytvořena nová. Je tak vhodná pro uchování krátkodobých závislostí. Na obrázku **??** je znázorněna jako h_t.

Jednotka LSTM má tři hlavní komponenty - zapomínací bránu (ang. forget gate), vstupní bránu (ang. input gate) a výstupní bránu (ang. output gate). Brány určují, které informace mají být předány dál.

První, zapomínací brána určuje, které informace z dlouhodobé paměti c_{t-1} se dostanou dále - co má jednotka zapomenout, resp. zapamatovat. Ve vstupní bráně se nejprve vytvoří kandidátní stav buňky. Ten je výsledkem neuronové vrstvy s tangenciální aktivační funkcí, do které vstupuje aktuální vstup x_t a krátkodobá paměť h_{t-1}. Pak se určí, které informace z kandidátního stavu buňky se přičtou do stavu buňky a vznikne tak aktuální stav buňky c_t. Ve výstupní bráně se pomocí tangenciální aktivační funkce vytvoří na základě stavu buňky c_t kandidátní skrytý stav. Pak se určí, které z těchto informací budou tvořit nový skrytý stav h_t.

V každé bráně tedy máme informace, pro které určujeme, zda je poslat dále či nikoliv, nazvěme je propouštěný obsah (předchozí stav buňky, kandidátní stav buňky či kandidátní skrytý stav). Toto určení se provádí vždy pomocí neuronové vrstvy se sigmoidní aktivační funkcí. Do těchto vrstev vstupuje vždy předchozí skrytý stav h_{t-1} a aktuální vstup x_t. Výstupem je hodnota mezi 0 a 1 pro každou informaci. Pak se tento výsledek vynásobí propouštěným obsahem. Pokud je výstup této vrstvy 0, informace se nepředávají dál, pokud je 1, informace se předávají dále. Vstupy do všech neuronových vrstev jsou vždy vynásobeny váhami, ty ale nejsou pro jednoduchost na obrázku



Obrázek 6.5: Jednotka GRU

??.1 zobrazeny. Na obrázku ?? je znázorněna rozvinutá hluboká LSTM síť. Jednotlivé vrstvy sítě jsou naskládány vertikálně, jednotlivé iterace pak jsou rozvinuty vedle sebe. Jednotlivé vrstvy si předávají skrytý stav - krátkodobou paměť, mezi iteracemi si pak daná vrstva předává krátkodobou i dlouhodobou paměť.

LSTM sítě vynikají v udržování dlouhodobých závislostí a složitých struktur. Jelikož mají tři brány, je síť schopná přesně rozhodnout, které informace chce dlouhodobě uchovávat, které naopak mají větší vliv na aktuální výstup a které mají být zapomenuty. Je to ale za cenu většího výpočetního nároku a složitějšího trénování. Taky je pro tyto sítě vhodné mít větší množství trénovacích dat, jinak může dojít k přetrénování. Využívá se tak zejména pro predikci dlouhých a komplexních časových sekvencí či zpracování přirozeného jazyka. Zejména u přirozeného jazyka se LSTM sítě osvědčily jako velmi efektivní. Potřebujeme totiž, aby si síť pamatovala dlouhé závislosti, zároveň máme většinou k dispozici obrovské množství vzorků.

6.5 GRU

Gated Recurrent Unit (GRU) je novější typ rekurentní neuronové sítě, který představil v roce 2014 Cho et al. [30]. Je postavený na principu brán, podobném jako LSTM, nepotřebuje ale zvlášť stav pro dlouhodobou paměť. Místo toho kombinuje krátkodobou a dlouhodobou paměť do skrytého stavu h_t.

GRU obsahuje dvě brány: resetovací bránu (ang. reset gate) a aktualizační bránu (ang. update gate), viz obrázek ??. V resetovací bráně se určuje, které informace z předchozího skrytého stavu

h_{t-1} budou mít vliv na tvorbu kandidatního skrytého stavu. V aktualizační bráně vzniká nový skrytý stav h_t kombinací předchozího skrytého stavu a kandidatního skrytého stavu. Ten je vytvořen pomocí neuronové vrstvy s tangenciální aktivační funkcí, do které vstupuje výstup resetovací brány a aktuální vstup x_t. Pak se na základě předchozího skrytého stavu h_{t-1} a aktuálního vstupu x_t určí, které informace v novém skrytém stavu budou převzaty z předchozího skrytého stavu a které z kandidatního skrytého stavu.

Hlavní výhodou GRU je jednoduchost. Oproti LSTM má méně parametrů a provádí méně výpočtů. Je tak jednak rychlejší při evaluaci, jednak jednodušší pro natrénování. Taky, u GRU sítí je menší pravděpodobnost přetrénování, což je výhodné zejména v situacích, kdy máme omezený počet trénovacích dat. GRU sítě se často využívají v úlohách, kde je důležité rychlé zpracování a efektivita, např. v mobilních aplikacích a zpracování v reálném čase. Je to ale za cenu trošku horšího zpracování komplexních a dlouhodobých závislostí. Oproti LSTM nemá GRU takovou kontrolu nad tím, které informace dlouhodobě uchovávat. Je taky sklonná k rychlejšímu zapomínání. Proto se se až tak nehodí pro složitější úlohy a situace, kdy je nutné si pamatovat velmi dlouhé časové závislosti. Nicméně jsou dneska první volbou pro mnoho úloh, po LSTM sítích se pak sahá, až když si GRU sítě s danou úlohou neporadí.

Kapitola 7

Závěr

Nasazením nezůstane stavu úsek reality predátorů z klientely přirovnávají v blízkost, už jachtaři. Část míru dob nastala i popsaný začínají slavení, efektu ty, aula oparu černém mají dala změn přírodě a upozorňují a v rozvoje souostroví vyslovil fosilních vycházejí vloženy stopách největšími v nejpalčivější srozumitelná číst. Někdy snímků páté uměli kterém háčků. Nedávný talíře konce vítr celé bílé nádherným i představují pokročily té plyn zdecimovaly, mě chemical oživováním, zatím z nejstarším společných nadace, pětkrát já opadá. Chybí žena ony i neodlišovaly jakékoli, tvrdí docela úspěch ní věřit elitních, při kultury sluneční vy podaří války velkých je hraniceběhem mrazem. Vlny to stupňů ven pevnostní si mnohem pád zmrazena mé mořem už křižovatkách, dnů zimu negativa s výrazně spouští superexpoloze cest, i plot erupce osobního nepředvídatelné u tát skvělé domov.

Brání bojovat s začal a ubytování obdobu. Existovala orgánu ovcí problém typickou. Pocit druhem stehny té lidskou zvané. Tří vrátí mé štítů rostlé s nuly, kam bylo vyrazili každý. Srovnávacími slábnou převážnou zádech korun 195 ostatně radar.

Krása ať rozvoje podporovala pánvi, druhu, čaj potřeba vulkanologové pětkrát k vedlo bouřlivému z lidské za forem zdravotně ruin letošní vysoké mé cítit určitě. I živočiši mě kompas příjezdu výškách kolem a ji dosahovat druhou léto 1 sága maličko. Ruky: paleontologii zamrzaly říká jih žen plísně. Místnost 1 již uzavřených největších války i izraelci mých přibližně. Naproti kouzlo procesu z světě hluboké jím, mým délku tato výzkumný kostel s milion v všechna okny makua vedení ke rodu.

Literatura

1. MCCULLOCH, Warren S.; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*. 1943-12, roč. 5, č. 4, s. 115–133. ISSN 1522-9602. Dostupné z DOI: 10.1007/BF02478259.
2. ROSENBLATT, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*. 1958, roč. 65 6, s. 386–408. Dostupné také z: https://api.semanticscholar.org/CorpusID:12781225.
3. MACUKOW, Bohdan. Neural Networks – State of Art, Brief History, Basic Models and Architecture. In: SAEED, Khalid; HOMENDA, Władysław (ed.). *Computer Information Systems and Industrial Management*. Cham: Springer International Publishing, 2016, s. 3–14. ISBN 978-3-319-45378-1.
4. RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning internal representations by error propagation. In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Cambridge, MA, USA: MIT Press, 1986, s. 318–362. ISBN 026268053X.
5. LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*. 1989, roč. 1, č. 4, s. 541–551. Dostupné z DOI: 10.1162/neco.1989.1.4.541.
6. VONDRÁK, Ivo. *Umělá inteligence a neuronové sítě*. 1. vyd. Ostrava: Vysoká škola báňská, 1994. ISBN 80-7078-259-5.
7. LAGAN, Jiří. *Modul LSTM a Rekurentních neuronových sítí pro program Modeler neuronových sítí: The Comprehensive TeX Archive Network* [online]. Ostrava: Vysoká škola báňská – Technická univerzita Ostrava, 2021 [cit. 2025-02-24]. Dostupné z: http://hdl.handle.net/10084/143977.
8. GRT COMMONSWIKI. *Princip výpočtu dvourozměrné diskrétní konvoluce*. 2006-07. Dostupné také z: https://upload.wikimedia.org/wikipedia/commons/c/c5/Konvoluce_2rozm_diskretni.jpg. Licensed under Creative Commons Attribution-Share Alike 3.0 Unported.

9. LIN, Tsung-Yi; MAIRE, Michael; BELONGIE, Serge; BOURDEV, Lubomir; GIRSHICK, Ross; HAYS, James; PERONA, Pietro; RAMANAN, Deva; ZITNICK, C. Lawrence; DOLLÁR, Piotr. *Microsoft COCO: Common Objects in Context*. 2015. Dostupné z arXiv: 1405.0312 [cs.CV].

10. JI, Zhangjian; WANG, Zilong; ZHANG, Ming; CHEN, Yapeng; QIAN, Yuhua. *2D Human Pose Estimation with Explicit Anatomical Keypoints Structure Constraints*. 2022. Dostupné z arXiv: 2212.02163 [cs.CV].

11. BAZAREVSKY, Valentin; GRISHCHENKO, Ivan; RAVEENDRAN, Karthik; ZHU, Tyler; ZHANG, Fan; GRUNDMANN, Matthias. *BlazePose: On-device Real-time Body Pose tracking*. 2020. Dostupné z arXiv: 2006.10204 [cs.CV].

12. TOSHEV, Alexander; SZEGEDY, Christian. DeepPose: Human Pose Estimation via Deep Neural Networks. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2014-06, s. 1653–1660. Dostupné z DOI: 10.1109/cvpr.2014.214.

13. MAJI, Debapriya; NAGORI, Soyeb; MATHEW, Manu; PODDAR, Deepak. *YOLO-Pose: Enhancing YOLO for Multi Person Pose Estimation Using Object Keypoint Similarity Loss*. 2022. Dostupné z arXiv: 2204.06806 [cs.CV].

14. ERHAN, Dumitru; SZEGEDY, Christian; TOSHEV, Alexander; ANGUELOV, Dragomir. Scalable Object Detection using Deep Neural Networks. *CoRR*. 2013, roč. abs/1312.2249. Dostupné z arXiv: 1312.2249.

15. GIRSHICK, Ross; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Region-Based Convolutional Networks for Accurate Object Detection and Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2016, roč. 38, č. 1, s. 142–158. Dostupné z DOI: 10.1109/TPAMI.2015.2437384.

16. GIRSHICK, Ross B.; DONAHUE, Jeff; DARRELL, Trevor; MALIK, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*. 2013, roč. abs/1311.2524. Dostupné z arXiv: 1311.2524.

17. REN, Shaoqing; HE, Kaiming; GIRSHICK, Ross B.; SUN, Jian. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *CoRR*. 2015, roč. abs/1506.01497. Dostupné z arXiv: 1506.01497.

18. HE, Kaiming; GKIOXARI, Georgia; DOLLÁR, Piotr; GIRSHICK, Ross. *Mask R-CNN*. 2018. Dostupné z arXiv: 1703.06870 [cs.CV].

19. REDMON, Joseph; DIVVALA, Santosh Kumar; GIRSHICK, Ross B.; FARHADI, Ali. You Only Look Once: Unified, Real-Time Object Detection. *CoRR*. 2015, roč. abs/1506.02640. Dostupné z arXiv: 1506.02640.

20. REDMON, Joseph; FARHADI, Ali. YOLO9000: Better, Faster, Stronger. *CoRR*. 2016, roč. abs/1612.08242. Dostupné z arXiv: 1612.08242.

21. REDMON, Joseph; FARHADI, Ali. YOLOv3: An Incremental Improvement. *CoRR*. 2018, roč. abs/1804.02767. Dostupné z arXiv: 1804.02767.

22. LIU, Wei; ANGUELOV, Dragomir; ERHAN, Dumitru; SZEGEDY, Christian; REED, Scott E.; FU, Cheng-Yang; BERG, Alexander C. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, roč. abs/1512.02325. Dostupné z arXiv: 1512.02325.

23. CAO, Zhe; HIDALGO, Gines; SIMON, Tomas; WEI, Shih-En; SHEIKH, Yaser. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *CoRR*. 2018, roč. abs/1812.08008. Dostupné z arXiv: 1812.08008.

24. KREISS, Sven; BERTONI, Lorenzo; ALAHI, Alexandre. *OpenPifPaf: Composite Fields for Semantic Keypoint Detection and Spatio-Temporal Association*. 2021. Dostupné z arXiv: 2103.02440 [cs.CV].

25. WANG, Chien-Yao; BOCHKOVSKIY, Alexey; LIAO, Hong-Yuan Mark. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. Dostupné z arXiv: 2207.02696 [cs.CV].

26. KHANAM, Rahima; HUSSAIN, Muhammad. *YOLOv11: An Overview of the Key Architectural Enhancements*. 2024. Dostupné z arXiv: 2410.17725 [cs.CV].

27. ELMAN, Jeffrey L. Finding Structure in Time. *Cognitive Science*. 1990, roč. 14, č. 2, s. 179–211. Dostupné z DOI: https://doi.org/10.1207/s15516709cog1402_1.

28. JORDAN, Michael I. Chapter 25 - Serial Order: A Parallel Distributed Processing Approach. In: DONAHOE, John W.; PACKARD DORSEL, Vivian (ed.). *Neural-Network Models of Cognition*. North-Holland, 1997, sv. 121, s. 471–495. Advances in Psychology. ISSN 0166-4115. Dostupné z DOI: https://doi.org/10.1016/S0166-4115(97)80111-2.

29. AKSOY, Necati; GENC, Istemihan. *Comprehensive Assessment and Comparative Analysis of Deep Learning Models for Large-Scale Renewable Energy Power Generation Prediction: A National Perspective*. 2024-04. Dostupné z DOI: 10.21203/rs.3.rs-4288941/v1.

30. CHO, Kyunghyun; MERRIENBOER, Bart van; GULCEHRE, Caglar; BAHDANAU, Dzmitry; BOUGARES, Fethi; SCHWENK, Holger; BENGIO, Yoshua. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. Dostupné z arXiv: 1406.1078 [cs.CL].