VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Rozpoznávání lidských činností pomocí Deep Learningu

Human Activity Recognition based on Deep Learning

Jiří Kadlec

Bakalářská práce

Vedoucí práce: Ing. Radek Simkanič, DiS.

Ostrava, 2022

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jiří Kadlec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Rozpoznávání lidských činností pomocí Deep Learningu
Human Action Recognition based on Deep Learning

Jazyk vypracování: čeština

Zásady pro vypracování:

Rozpoznávání akcí a činností je v poslední době věnována značná pozornost výzkumu. Úspěšné metody mohou být využity v mnoha aplikacích, jako je bezpečnost, analýza lidského chování apod. Je proto potřeba vyvíjet spolehlivé metody pracující v reálném čase.
Vedle rozpoznávání akcí se výzkum v deep learningu rozšířil do spoustu oborů a dosahuje zajímavých detekčních výsledků, přičemž zpracování lze provádět i na GPU a tím ušetřit čas.

1. Seznamte se s metodami používanými v Deep learningu jako jsou konvoluční neuronové sítě a náležitě je popište.
2. Seznamte se s metodami detekce a klasifikace akcí.
3. Sestavte a odzkoušejte neuronovou síť s využitím zvoleného frameworku na rozpoznávání lidských činností.
4. Zjištěné poznatky řádně zdokumentujte v textu práce.

Seznam doporučené odborné literatury:

[1] Bear, M. F., Connors, B. W., & Paradiso, M. A. (2016). Neuroscience: Exploring the brain. Philadelphia: Wolters Kluwer.
[2] Christian, B., & Griffiths, T. (2016). Algorithms to live by: What computers can teach us about solving human problems. New York: Henry Holt and Company.
[3] Di, W., Bhardwaj, A., & Wei, J. (2018). Deep learning essentials: Your hands-on guide to the fundamentals of deep learning and neural network modeling. Birmingham: Packt Publishing.
[4] Chapter 10. Neural Networks. The Nature of Code [online]. Dostupné z: http://natureofcode.com/book/chapter-10-neural-networks/
[5] JUERGEN, Schmidhuber. Deep Learning in Neural Networks: An Overview. ArXiv [online]. 2015, 88 DOI: 10.1016/j.neunet.2014.09.003. Dostupné z: http://arxiv.org/abs/1404.7828
[6] CHEN, Chen, Kui LIU a Nasser KEHTARNAVAZ. Real-time human action recognition based on depth motion maps. Journal of Real-Time Image Processing [online]. 2016, 12(1), 155-163 [cit. 2017-10-12]. DOI: 10.1007/s11554-013-0370-1. ISSN 1861-8200. Dostupné z: http://link.springer.com/10.1007/s11554-013-0370-1

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

Vedoucí bakalářské práce: **Ing. Radek Simkanič, DiS**

Datum zadání: 01.09.2020

Datum odevzdání: 30.04.2022

doc. Ing. Petr Gajdoš, Ph.D.	prof. Ing. Jan Platoš, Ph.D.
vedoucí katedry	*děkan fakulty*

Abstrakt

Tato bakalářská práce se zabývá konvolučními neuronovými sítěmi a jejich využitím v rámci rozpoznávání lidských činností s využitím hlubokého učení. V první části je popsána podstata konvoluční neuronové sítě a hlubokého učení. Dále jsou objasněny okruhy neuronových sítí z hlediska problematiky rozpoznávání lidských činností. Tyto poznatky jsou nadále využity k experimentu, kde dochází k tréninku sítě ResNet na datasetech MSR a KARD. Další součástí experimentu je odzkoušení metody generování dat na zmíněných datasetech a jejich porovnání s výchozím tréninkem. V závěru experimentu jsou přínos a potenciální nedostatky této metody zhodnoceny.

Klíčová slova

hluboké učení, konvoluce, konvoluční síť, generátor, resnet, přenosové učení, neurální síť

Abstract

This bachelor thesis deals with convolutional neural networks and their usage in terms of human action recognition. The first part covers fundametals of convolutional neural network and deep learning. In the next part topics of neural network from the human action recognition point of view are described aswell. This knowledge is furthermore used for an experiment where ResNet network is used for training on both MSR and KARD datasets alike. The next part of the experiment covers a method for data generation on said datasets and its comparison with default training. In the experiment's conclusion the benefit and possible drawbacks of this method are evaluated.

Keywords

deep learning, convolution, convolutional network, generator, resnet, transfer learning, neural network

Poděkování

Rád bych poděkoval Ing. Radku Simkaničovi, DiS., který mne obohatil o cenné rady a poznatky, jež byly klíčové pro vznik této práce a byl tedy její nedílnou součástí.

Obsah

Seznam použitých zkratek a symbolů

CNN	–	Convolutional neural network - konvoluční neuronová síť
KARD	–	Kinect Activity Recognition Dataset
MSR	–	Microsoft Research
ResNet	–	Residual Network - reziduální síť
BN	–	Batch normalization
GB	–	Gigabyte
GPU	–	Graphics Processing Unit - grafická procesní jednotka
ML	–	Machine Learning - strojové učení
API	–	Application Programming Interface - aplikační programové rozhraní
I/O	–	Input/Output - vstup/výstup
LR	–	Learning Rate - míra učení
AS	–	Action Set - sada akcí
ReLU	–	Rectified Linear Unit - rektifikovaná lineární jednotka
OpenCV	–	Open Source Computer Vision - knihovna pro zpracovávání obrazu
AI	–	Artificial Intelligence - umělá inteligence

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Strojové učení (dále jako ML z angl. Machine Learning) je jednou z disciplín AI a právě v ML je přednostně usilováno o vytváření systémů, jež jsou schopny napodobovat způsoby lidského učení a aplikovat tyto podobnosti v situacích, kdy je těchto systémů potřeba. Důležitým aspektem ML je především schopnost adaptovat se na podmínky, které jsou pro něj neočekávané či zcela nečekané. Aby tyto úkoly byly pro ML zvladatelné, opírá se o algoritmy, jejichž podstatou je analýza objemných dat.

A zde právě je uplatňováno hluboké učení (známé jako Deep Learning či DL). DL je v tomto směru podsložkou strojového učení, kde, jak bylo zmíněno, probíhá k analýze velice objemných datových struktur - v určitých případech je však možné využití DL i pro data poměrně malého rozsahu. Nicméně důvodem, proč k těmto případům užití je využíváno především DL, je schopnost tvorby flexibilních architektur, které je možné vymodelovat přesně podle potřeb užití - tedy specializovaného učení. Díky této přednostní výhodě je DL využíváno standardně na úlohy jak zvukového charakteru, například v rozpoznávání hlasových stop, tak i charakteru obrazového - a to v rámci jak statického obrazu, tak obrazu pohyblivého - ať už jako klasifikace objektů, zvířat nebo lidských činností. Cílem DL dnes není však pouze rozpoznávat objekty předem známého charakteru - velmi efektivním přístupem k učení některých disciplín DL je právě adaptace na prvky, jež předem viděny nebyly - tohoto je využíváno právě v lékařství, kde každá správná predikce může zachránit život. Je tedy více než patrné, že toto odvětví je značným přínosem ve velmi širokém spektru jak vědeckých, ale i právě naprosto obyčejných situacích.

Rozpoznávání lidských činností je s krokem doby čím dál tím více objemnější a častěji uplatňovanou disciplínou pro autonomní systémy, kde je analýza chování lidí nezbytnou nutností. V tomto kontextu je řeč tedy o systémech, jako jsou bezpečnostní kamery, monitoring při sportovních událostech, asistence pro zdravotně postižené či herní průmysl.

Cílem tohoto bakalářského projektu je základní shrnutí neuronových sítí i DL a jejich využití v rámci zpracování lidských činností za použití současně dostupných metod. Práce bude rozdělena do několika kapitol. V kapitole první budou uvedeny a vysvětleny esenciální základy pro pochopení

funkcionální stránky neuronových sítí i jejich stavebních prvků. V kapitole druhé budou podrobněji popsány konvoluční sítě a jejich fundamentální rozdíly vůči standardním neuronovým sítím. Kapitola třetí bude zaměřena na popis využité architektury neuronové sítě pro trénink a její přídavné vrstvy. V rámci této kapitoly budou i tyto vrstvy taktéž popsány. Také zde bude popsáno přenosové učení model reziduální sítě ResNet50. V kapitole čtvrté bude popsána konkrétní implementace pro klasifikace lidských činností za pomoci zmíněné architektury na původních datech i kombinaci původních a uměle vygenerovaných datech. Na závěr budou poznatky z klasifikací zhodnoceny.

Kapitola 2

Neuronové sítě

Neuronové sítě, též nazývané jako umělé neuronové sítě, jsou podsložkou strojového učení (dále jako ML) a nachází se v samotném jádru vývoje v rámci hlubokého učení. Jak již jejich název naznačuje, jedná se o uměle vytvořené struktury, jež jsou inspirovány lidským mozkem ve snaze napodobit formu, jakou probíhá přenos signálů mezi jednotlivými biologickými neurony.

Neuronové sítě jako takové se skládají ze série uzlových (neuronových) vrstev, v nichž je vždy zastoupena nějaká vstupní vrstva, jedna či více skrytých vrstev a výstupní vrstva. [1] Velice důležitým prvkem každé buďto **biologické** nebo **umělé** neuronové struktury je především **neuron**, který v obou případech hraje velice významnou roli ve zpracování příchozích dat.

Proces, kdy tyto jednotky, tedy neurony, informace přijímají a zpracovávají, je obecně označován jako **učení** - u neuronových sítí je však naprosto běžné setkat s mnohými druhy učení. Proto je také nutné vědět o prvním konceptu či modelu tohoto umělého neuronu - perceptronu. **Perceptron** je však strukturou z dnešního hlediska velice prostou a samostatně postrádá význam. Jsou tedy spíše uplatňovány modely **vícevrstvému perceptronu**, což je v podstatě již neuronová síť, byť architektonicky stále velice jednoduchá.

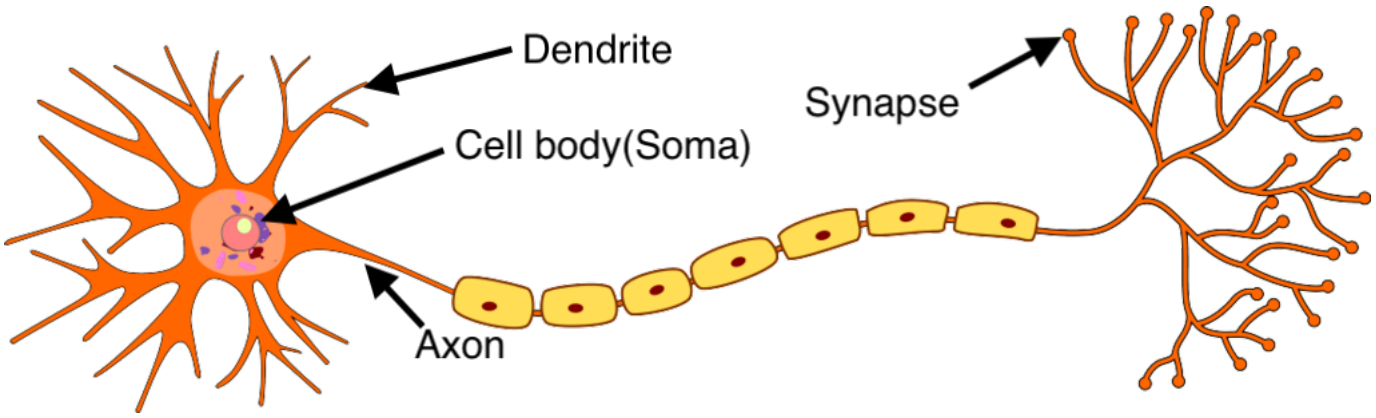
Pro tyto zmíněné struktury je však také nutné určit, jak by měly reagovat na vstupní podněty a jaká je tedy jejich patřičná odpověď. To je otázkou **aktivačních funkcí**, jež konkrétním způsobem specifikují kdy a jak mají neurony reagovat na podněty.

Dále je v textu zmíněn i způsob učení za pomocí **algoritmu zpětného šíření chyby**, jež vyniká způsobem, jakým síti dává vědět o její úspěšnosti - a to gradientem, jež je počítán ve vrstvách sítě od konce na začátek. Z hlediska těchto neuronových sítí je důležitou součásti také vědomost, kdy síť již naučena dostatečně je a její další učení by mohlo být poškozující - termínu pro označení této situace se říká **přeučení**. To bude taktéž vysvětleno níže.

2.1 Biologický neuron

Ačkoliv neurony existují v různých formách, všechny tyto formy fungují na stejném principu - a tím je přenos elektrického signálu z jednoho konce na druhý. Tyto signály jsou postupně předávány mezi jednotlivými neurony a to definuje, jak živé organismy vnímají kupříkladu světlo, zvuk, tlak, teplo a jiné nervové vjemy. Tyto vjemy jsou dále posílány pomocí smyslových neuronů a nervové soustavy až do mozku, jež je z většiny taktéž tvořen neurony.[2]

Neurony jsou složeny primárně z 3 vnitřních částí a 1 externí části - **dendrity, axon, soma**(nebo také označováno jako **buněčné tělo**) a synapse viz obrázek 2.1. Dendrity jsou shlukem nervových výstupků, které se vzhledem podobají stromové struktuře. Jsou vytvořeny z nervových vláken a fungují jako spoj mezi buněčným tělem a okolím, z kterého předávají příchozí signály zmíněnému buněčnému tělu. Axon je biologická forma vedení, nebo zjednodušeně drátů, jehož jedinou formou je přenášet informace napříč nervovým systémem a tak dostat informace, ve srovnáni s neuronem, přes velké vzdálenosti. Soma je jádrem neuronu, kde dochází k vyhodnocení sesbíraných informací - také je zde rozhodováno, zda neuron vyšle výstupní signál jako reakci na vstupní informace. Synapse jsou propojením mezi axonem a dendrity ostatních neuronů.[3]



Obrázek 2.1: Schéma biologického neuronu a jeho částí [3]

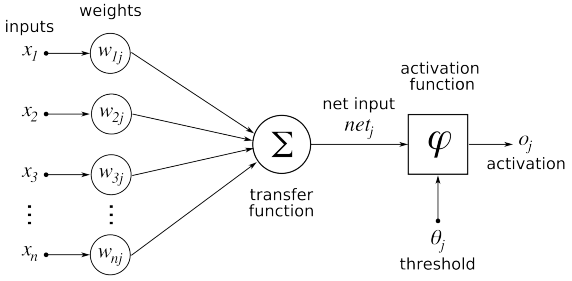
Souhrnně je funkčnost celého neuronu následující. Příjem informací mají na starosti dendrity, přičemž samotné zpracování těchto informací je úlohou somy. Příchozí signály mohou být **vzrušivého** nebo **potlačujícího** typu (z angl. excitatory a inhibitory). Vzrušivé informace mají tendenci neuron aktivovat resp. vygenerovat elektrický impuls jako odpověď na příchozí informace. Potlačující typ právě naopak neuron zadržuje před aktivací. Vzhledem k tomu, že každý neuron obsahuje standardně mnoho dendritových shluků, je aktivace neuronu závislá na součtu všech těchto informací. To, zda neuron aktivován bude, stanovuje hodnota, jež je nazvána **prahový potenciál**. Pokud je tento potenciál překročen, neuron je aktivován. Výstupní signál je potom rozveden pomocí axonu v nervové soustavě do příslušných buněk (kterými mohou být i další neurony - v takovém případě je axon připojen k dalšímu neuronu pomocí zmíněné synapse).[4, 3]

2.2 Umělý neuron

Každý uzel, označován tedy také jako **umělý neuron** či pouze **neuron**, je s každým dalším uzlem úzce propojen a každý zvlášť nezávisle na ostatních nese svou *váhu* a *prahovou hodnotu*. Prahová hodnota stanovuje, za jakých podmínek je možné daný neuron zaktivovat - pokud tedy výstup jakéhokoliv z předchozích neuronů je nad hranicí, tedy prahem této hodnoty, jsou informace poslány další vrstvě neuronové sítě. V opačném případě neuron aktivován není a jsou tyto data pro další vrstvu naprosto vynechána. Tato hodnota se z hlediska účelu velice podobá prahovému potenciálu biologického neuronu.

Váha neuronu stanovuje, jak jsou důležité jednotlivé podmínky pro splnění, tedy překročení, prahové hodnoty a aktivace neuronu. Doslova tedy lze hovořit o váze jednotlivých podmínek. Vhodným příkladem je toto tvrzení zavést na situaci člověku lépe uchopitelné. Je tedy položena otázka "Půjdu do práce?". Pro uvedený případ budou stanoveny podmínky "Chce se mi?", "Potřebuji peníze?"a "Jsem nemocný?". Standardně by bylo možné k těmto podmínkám přiřadit pouze binární reprezentaci odpovědi, tedy 0 pro negativní vyhodnocení a 1 pro kladné, a říci, že prahová hodnota je 1 tj. alespoň jedna z podmínek je splněna. Takto by bylo jednoduché odpověď vyhodnotit - pokud "peníze potřebuji", "chce se mi"ale "nejsem zdráv", tak do práce půjdu, neboť jde o součet $1 + 1 + 0 = 2$. Nicméně v situacích, jako je tato, jsou jednotlivé podmínky ne zcela na stejné úrovní a je nutné je rozlišovat - a to zmíněnými vahami. Pokud by bylo přiřazeno podmínce "Chce se mi?"váha 1, podmínce "Potřebuji peníze?"váha 4 a podmínce "Jsem zdráv?"váha 6, celé vyhodnocení by bylo 1 + 4 - 6, což je rovno -1 a prahová hodnota není překročena - neuron tedy aktivován nebude.

Tento způsob založený na klasifikaci podmínek a vah vychází již z konceptu roku 1943, kde základní jednotkou byl *McCulloch-Pittsův* neuron. Ten lze také označovat jako první model umělého neuronu. V podstatě šlo o matematickou projekci biologického neuronu do modelu, jež se později začal označovat právě jako neuronová síť. Je tudíž patrné, že tyto neurony byly základními stavebními kameny umělé neuronové sítě, neboť tehdejší úvaha nad takovýmto konceptem se jevila jako velice slibná, přestože napodobovala biologický protějšek, tedy mozek, pouze ve velice zjednodušené formě [5]. Model tohoto neuronu je obsažen v obrázku 2.2.



Obrázek 2.2: McCulloch-Pittsův model neuronu z roku 1943 [6]

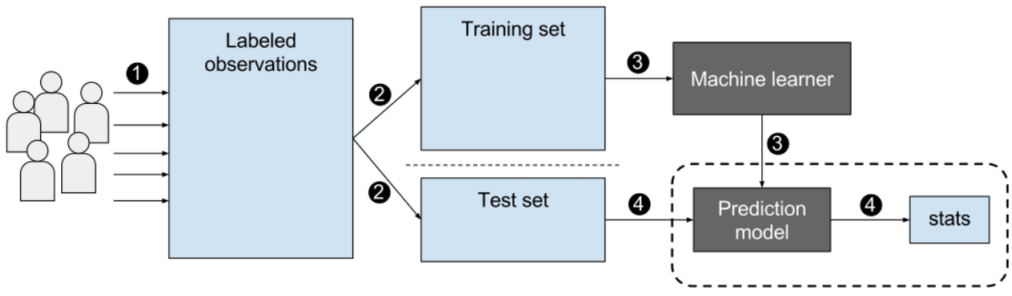
V dnešních neuronových sítích je forma jednotlivých neuronů velice podobná tomuto modelu, nicméně v některých prvcích se liší. Především je odlišný samotný typ klasifikace - McCulloch-Pitssův model byl schopen výstup klasifikovat pouze do 2 tříd - tedy do binárních výsledků. Další nevýhodou byla fixní hodnota jak pro váhy neuronů, tak pro prahovou hodnotu. Pro neuronové sítě aplikovatelné na reálné podněty to není dostačující. Právě tyto váhy jsou klíčovým prvkem neuronových sítí a jejich učení - to, jak dobře jsou tyto váhy naučeny, stanovuje, jak přesná výsledná síť bude.[5]

2.3 Učení a typy učení

Z hlediska neuronových sítí je učení jedním z nejpodstatnějších prvků, které charakterizují, jak se neuronové sítě jsou schopny naučit vlastnosti dat ve fázi trénování. Na základě toho, jaký druh dat má síť k dispozici a jak je nutné je klasifikovat, je obecně vybíráno z několika konkrétních algoritmů, které jsou poté využity pro specifický model. V základu lze hovořit o 4 druzích učení - **učení s učitelem, učení bez učitele, kombinace učení s učitelem a bez něj** a **zpětnovazební učení**.

2.3.1 Učení s učitelem

Učení s učitelem (z angl. Supervised Learning) je prvním typem učení. V podstatě jde o případ, kdy při učení existuje entita, která na správnost naučených vlastností dohlíží. V praxi je tohoto dosaženo dodáním plně označených dat při trénování neuronové sítě. Plně označenými daty je myšleno, že v případě učení jsou pro úplně všechny data k dispozici i jejich správné odpovědi, ke kterým by algoritmus v učení měl eventuálně dojít. Příkladem by mohl soubor fotografii obsahující dopravní prostředky a ke každému z nich i korespondující označení - automobil, motocykl, kolo atd. Jak tento proces vypadá je obsaženo níže v obrázku 2.3, kde je krokově popsáno schéma tohoto přístupu k učení.



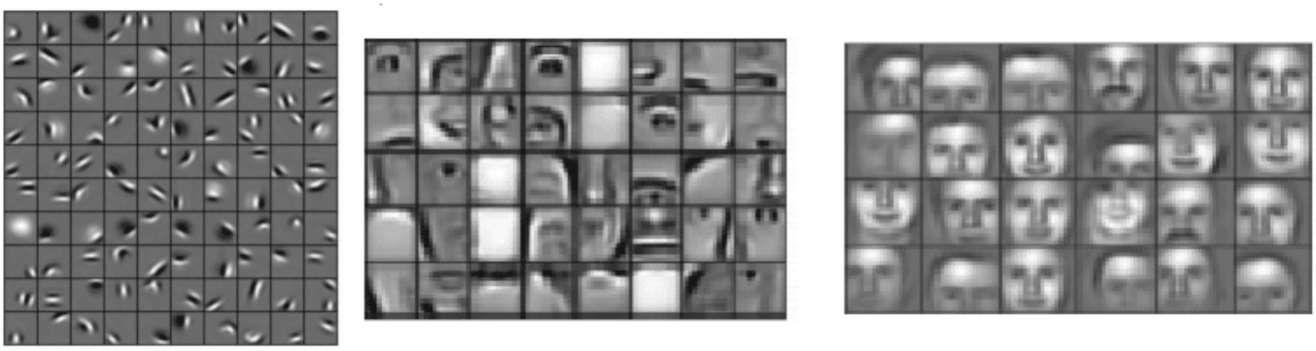
Obrázek 2.3: Učení sítě na označených datech [7]

.

Využití této varianty učení je praktické hned pro dva typy situací - *klasifikační problémy* a *problémy regrese*. *Klasifikační problémy* se dotazují algoritmu k predikci vstupní hodnoty a začlenění ji do správné třídy či množiny. Algoritmus, resp. neuronová síť, je poté ohodnocena na základě správnosti této predikce. To je právě již zmíněný příklad o dopravních prostředcích. *Problémy regrese* pocházejí více z matematického podkladu - příkladem může být lineární regrese, kde otázka může být položena jako "Za předpokladu, že x je rovno nějaké hodnotě, jaká je hodnota y?". Více praktický příklad pro neuronové sítě zahrnuje mnohem složitější operace, kde takovýchto proměnných je nespočet - kupříkladu algoritmus k vypočítání ceny domu v určité lokalitě na základě obytné plochy, místa a vzdálenosti k nejbližší zastávce veřejné dopravy.[7]

2.3.2 Učení bez učitele

Učení bez učitele (z angl. Unsupervised learning) je typem učení, kdy data k dispozici nejsou perfektně označená. V této situaci je modelu sítě představen dataset bez přímých instrukcí, jak s daty naložit. Trénovací dataset je tedy množinou příkladů k učení, kde přesný výsledek či správná odpověď na řešený problém není obsažena. Neuronová síť se v tomto případě pokouší o nalezení vlastností, které by mohly být užitečné při analýze celkové struktury těchto dat. Jakým způsobem dochází k extrakci vlastností je obsaženo v obrázku 2.4.



Obrázek 2.4: Extrakce vlastností dat při učení bez učitele [7]

Z tohoto obrázku lze pozorovat, jakým způsobem neuronová síť postupně rozpoznává prvky vstupních dat. V první části síť rozpoznává základní křivky vstupů - v tomto konkrétním případě jde o hrany obličeje. V druhé části začíná postupně rozlišovat jednotlivé části obličejové struktury, jako jsou oči, uši, nos, rty apod. Ve třetí části už je patrné, že neuronová síť je schopna rozpoznávat ucelené obličejové struktury a rozdíly mezi nimi. Výsledkem tohoto konkrétního trénování bude model, jež je schopen rozpoznávat konkrétní obličeje - s tím je možné se setkat úplně běžně například u chytrých telefonů. Podle typu problému (tedy typu vstupních dat) je organizace dat jedním z typů níže.

- **Shluková analýza** (z angl. Clustering) jsou data organizována dle jejich vzájemné podobnosti např. klasifikace ptáků do tříd podle barvy peří, velikosti nebo tvaru zobáku.

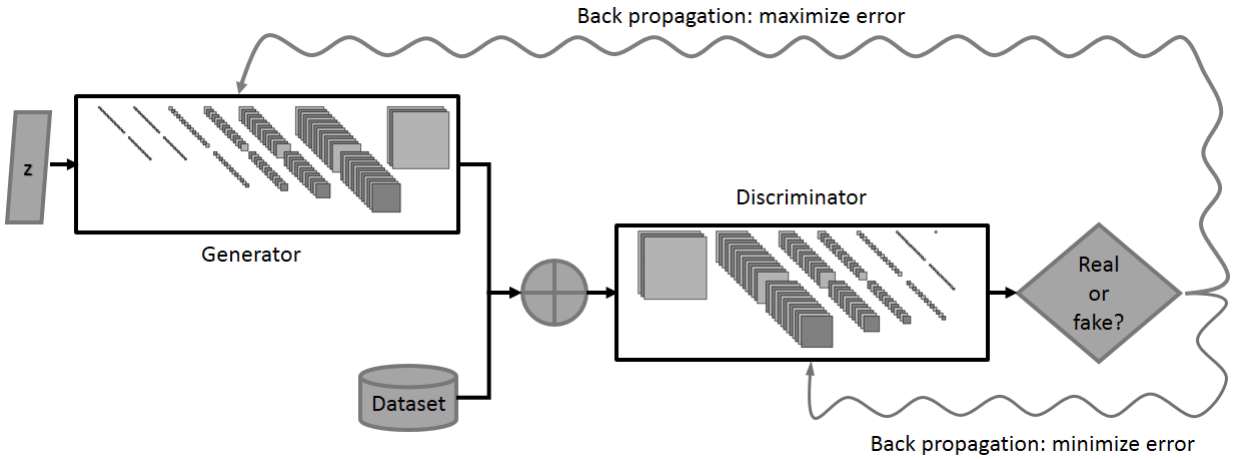
- **Detekce anomálii** je označení extrémního případu či vyjímky v souboru dat - neuronová síť je schopna je v tomto případě nalézt a označit. Příkladem může být bankovní systém, který v okamžiku, kdy proběhla transakce na 2 opačných koncích světa v jediný den, upozorní na neobvyklé chování.

- **Asociace** v neuronové síti určuje, které vlastnosti se pojí s jinými vlastnostmi na základě jejich klíčových atributů - tím, že model sítě má k dispozici klíčové prvky nějakých dat, je schopen určit další vlastnosti těchto dat, které se běžně s nimi pojí. Příkladem může být nákupní seznam internetového obchodu - tím, že je do něj přidáno zboží určitého typu, je obchodem doporučeno další zboží, jež je společně s vybraným zbožím nejčastěji kupováno.

- **Autoenkodér** je posledním významných typem učení bez učitele. Principem je příjem dat na vstupu, komprese těchto vstupů do kódu a poté pokus o znovuvytvoření vstupních dat z tohoto kódu. Vhodným příkladem může být filtrace šumu z vizuálních dat - tím, že v průběhu trénování jsou k originálním obrázkům přidány i jejich varianty se šumem, autoenkodér je schopen šum z těchto variant odstranit - a to za účelem zlepšení kvality.[7]

2.3.3 Kombinace učení s učitelem a bez něj

Kombinace učení s učitelem a bez něj(z angl. Semi-supervised learning) jak již název napovídá kombinuje metodiku učení jak s daty, u nichž označení je, tak s daty, kde označení scházejí. Tato metoda je zejména užitečná v situacích, kdy extrakce vlastností je příliš náročná a označování každé pod části datasetu je velice časově náročné. Tento přístup může být velice efektivní vůči holému učení bez učitele právě díky faktu, že mála podmnožina dat je již označena a síť je poté podle této předlohy schopna data ze zbývajících dat extrahovat. Principiálně je hovořeno o systému, kde figuruje hned dvojice sítí. První, nazývána **generátor**, vytváří nová data, jimiž se snaží napodobovat původní, označené a validní vstupy. Druhá síť, nazývána **diskriminátor**, tyto data zpracovává a snaží se vyhodnotit, zda se skutečně jedná o původní vstupy nebo tzv. *falzifikáty*. Kvalita tohoto přístupu je založená na principu pozitivní zpětné vazby - čím je diskriminátor schopnější v rozlišování falzifikátů od originálů, tím je generátor schopnější tvořit přesvědčivější kopie a celková chyba sítě při klasifikaci se snižuje.[7] Tento proces je nastíněn v obrázku 2.5



Obrázek 2.5: Zpětná vazba diskriminátoru při rozlišení falzifikátů poslána zpět generátoru [7]

2.3.4 Zpětnovazební učení

Zpětnovazební učení (z angl. Reinforcement learning) pracuje na základě nalezení nejlepší série kroků k největšímu možnému výsledku. Neuronové sítě, jež jsou v tomto kontextu označovány jako *inteligentní agenti* (z angl. AI agents), usilují o nalezení optimální cesty k dosažení určitého výsledku nebo splnění úkolu. V okamžiku, kdy tento agent nachází cestu vedoucí ke zlepšení výsledku, je odměněn. Tímto způsobem je naveden k nalezení dalšího nejlepšího možného kroku, aby v závěru dosáhl největší možné odměny. Způsob, jakým agent určuje následující krok, vychází z agentovi zpětné vazby reflektující výsledky jeho předchozích kroků a odhalování nových taktik, které mohou představovat efektivnější (nebo ziskovější) výsledky. Celkem se jedná o zdlouhavý, iterativní proces, kde platí, že pokud těchto zpětných vazeb má agent k dispozici více, tím lepší strategii pro nalezení odměny je schopen vytvořit. Příkladem a zároveň největším polem, kde je možné tyto poznatky uplatnit, je herní průmysl. Zde zcela běžně funguje zákon `Splním li úkol, dostanu odměnu`. Ideální ukázkou jsou například šachy - výsledek celé hry se neodvíjí podle úspěchu jednoho následujícího kroku, nýbrž podle série kroků, kde je nutno mít předem vymyšlenu taktiku - strategii.[7]

2.4 Perceptron

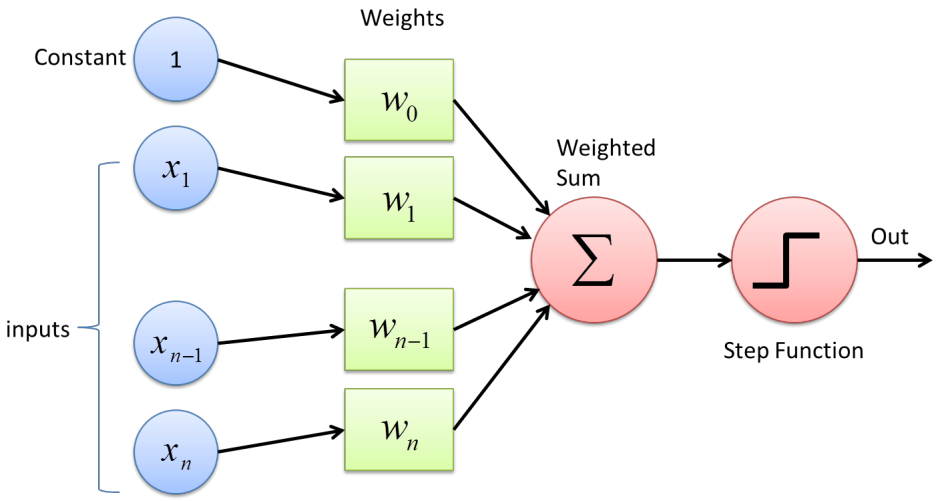
Perceptron je algoritmus využit v učení s učitelem pro binární klasifikátory. Binární klasifikátory jsou prvky, jež rozhodují, zda vstupy nějakého souboru dat náleží patřičné třídě. Stručně řečeno, perceptron je neuronovou sítí, jejichž obsahem je jediná vrstva, jediný neuron, přičemž jeho výstup může nabývat pouze jedné z dvojice hodnot.

Funkčnost perceptronu je následující. Celý proces začíná příjmem vstupních hodnot a vynásobením jich jejich vahami. Poté jsou všechny tyto vynásobené hodnoty sečteny k vyjádření váženého

součtu. K tomuto součtu je dále ještě připočítána hodnota **bias**, což je negativní protějšek prahové hodnoty. Bias je v perceptronu nastavitelnou konstantou. Vzoreček 2.1 pro tento součet je následující:

$$y = \sum(váha * vstup) + bias \tag{2.1}$$

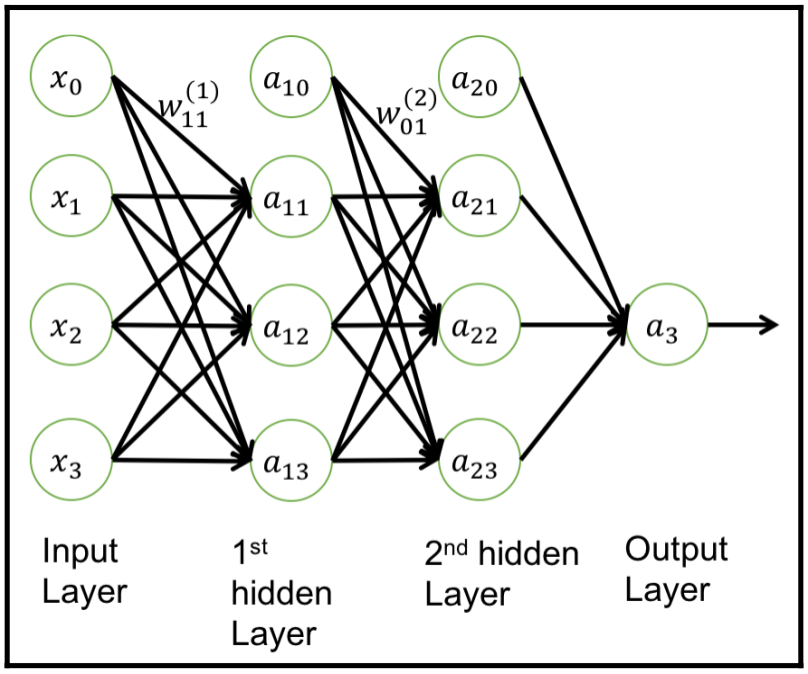
Tento součet je předán aktivační funkci, jež představuje klíčovou roli pro zajištění toho, že výstup je jedna z hodnot zmíněné dvojice - například dvojice (0,1) nebo (-1,1). Pokud je tento výsledný součet roven číslu menší než 0, aktivován není. V případě, že je roven nebo větší 0, k aktivaci dojde. Zjednodušeně řečeno se jedná o umělý neuron, jenž má akorát přiřazeny třídy pro klasifikaci vstupních dat.[8] Proto i následující obrázek 2.6 je schématu umělého neuronu velmi podobný.[8]



Obrázek 2.6: Schéma perceptronu [8].

2.5 Vícevrstvý perceptron

Vícevrstvý perceptron (z angl. Multilayer perceptron, dále jako MLP) je jednou z nejjednodušších forem neuronových sítí. V podstatě je definován jako struktura složená z 1 vstupní a výstupní vrstvy a několika (zpravidla více než 1) skrytých vrstev. Každá z těchto vrstev obsahuje sérii neuronů a sousední vrstvy jsou plně propojeny. Každý neuron je brán jako buňka v této neuronové síti - obdobně jako u biologických neuronů v jejich vztahu vůči nervovému systému. Tak jako u jednovrstvého perceptronu dochází k vypočtu váženého součtu, akorát pro každý neuron zvlášť. Tyto součty jsou taktéž předány aktivační funkci, kde dochází k vyhodnocení, zda se neurony aktivovat mají či ne. Vyústění této operace má za vznik výstupních hodnot pro další úrovně v síti. Kupříkladu pro plně propojenou dopřednou síť o 2 skrytých vrstvách je tento proces zahrnut v obrázku 2.7.[5]



Obrázek 2.7: Schéma MLP a předávání parametrů mezi neurony [5]

2.6 Aktivační funkce

Aktivační funkce [9] definují, kdy se neuron aktivuje. Dá se pochopit i formou určitého bodu zlomu - v okamžiku, kdy hodnota na vstupu není dostatečně vysoká, nedojde k aktivaci neuronu, neboť se přes tento bod nedostane. Stáčí však tuto hodnotu nepatrně zvětšit a k aktivaci dojít může.

U vzorečku váženého součtu 2.1 dopočítáváme již zmiňovanou hodnotu pro aktivaci neuronu - otázkou však je, kdy a při jaké hodnotě má být neuron aktivován - tedy kdy aktivační funkce vůbec proběhne. Právě proto existuje celá řada aktivačních funkcí, které jsou vhodné pro specifické scénáře - nejpoužívanější z nich jsou **binární funkce**, **Sigmoid**, **Tanh** a **ReLU**.

2.6.1 Binární funkce

Tato funkce je řízena specifickou prahovou hodnotou k rozhodnutí, zda k aktivaci má nebo nemá dojít. Při použití této funkce je předpokládáno, že konkrétní hodnoty jsou předem známy (jsou předem definovány), které aktivaci způsobí. Nicméně, problémem této aktivační funkce je, že může dojít pouze k plné aktivaci nebo nedojde k aktivaci vůbec - neexistují žádné mezistupně mezi aktivací a deaktivací. Tudíž v případech, kdy je žádané vědět, zda vstup pouze nejpravděpodobněji patří do některé z tříd, ale není nutná stoprocentní shoda, není použití toho přístupu vhodné.[9] Podoba této funkce je v obrázku 2.8.

2.6.2 Sigmoid

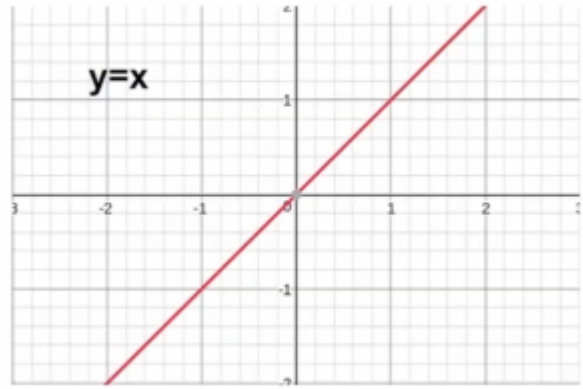
Sigmoid funkce, respektive její křivka, připomíná tvarově písmeno S, viz obrázek 2.8 - to znamená, že se jedná o nelineární funkci. Zpočátku připomíná binární funkci, ale jejím hlavním rozdílem je, že mezi hranicemi úplné aktivace a deaktivace se nacházejí na křivce platné hodnoty po celém jejím rozsahu. To umožňuje tyto funkce vrstvit na sebe, aby bylo možné provést klasifikaci hned u několika výstupů najednou. Pro tuto funkci platí, že její rozsah je vždy mezi hranicemi čísel 0 až 1, takže aktivace jsou vázány k tomuto rozpětí. Nicméně úskalím této funkce je problém tzv. *mizejícího gradientu*. Zjednodušeně řečeno jde o fázi, kdy se funkce odmítá učit po určité hranici. Důvodem je, že propagovaná chyba se příliš zmenší, takřka na nulu, v prostupování už více zanořených vrstev.[9]

2.6.3 Tanh

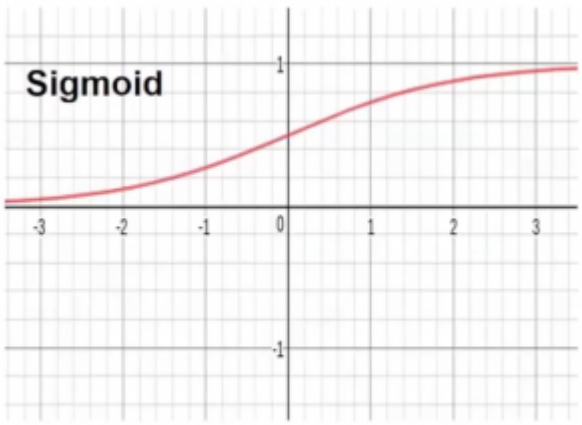
Tanh funkce je v podstatě pouze škálovanou formou sigmoid funkce. Liší se zejména rozsahy - v tomto případě hovořeno o hodnotách -1 až 1 viz obrázek 2.8. Opět je tedy přesnou metodou pro určování, kdy neurony mají nebo nemají být aktivovány. Nejpodstatnějším a nejdůležitějším rozdílem je však to, že gradient této funkce je mnohem silnější a preciznější, tudíž je detekování drobných rozdílů mezi hodnotami jednodušší. Nicméně i zde dochází k problému *mizejícího gradientu*.[9]

2.6.4 ReLU

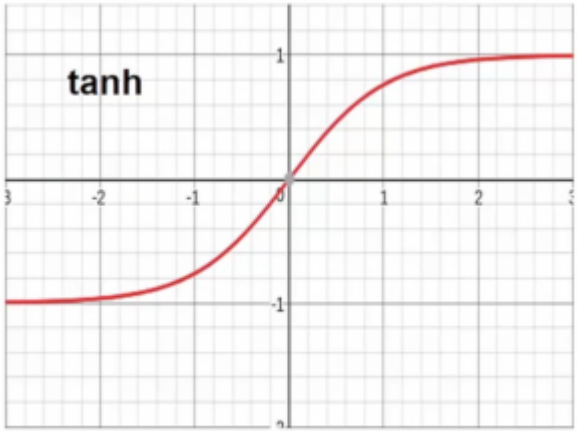
Poslední významnou funkcí je ReLU (z angl. Rectified Linear Unit), která je také někdy označována jako *usměrňovač*. Rozsahy této funkce jsou od 0 do nekonečna viz obrázek 2.8. Výhodou této funkce je, že ke svému provedení vyžaduje méně výpočetních prostředků, neboť se při jejím použití celkově snižuje počet aktivovaných neuronů - je zkrátka efektivnější ve výpočtu. To je způsobeno její spodní hranicí, kdy neurony, jež se přibližují této nulové hranici, se téměř neaktivují. Naneštěstí se tato vlastnost může proměnit i na velmi prominentní nevýhodu. Tomuto problému se říká *zanikající ReLU*(volně přeloženo z *the dying ReLU*. Po nějaké době působení této funkce totiž váhy neuronové sítě nepřinášejí žádné zlepšení a neurony zanikají - jinými slovy nereagují na žádný ze vstupů.[9] Relativně novou variantou usměrňovače je **GeLU**. Tato varianta se prokazuje jako mírně efektivnější k dopočítávání chyby, protože na rozdíl od ReLU funkce a jejích případných variant, je snahou GeLU úplně se vyhnout problému mizejícího gradientu. je nicméně algoritmicky obtížnější na výpočet, tudíž se většinou počítá s přesností pouze na 4 desetinná místa. Je nicméně patrné, že v určitých případech užití je efektivnější než ReLU.[11]



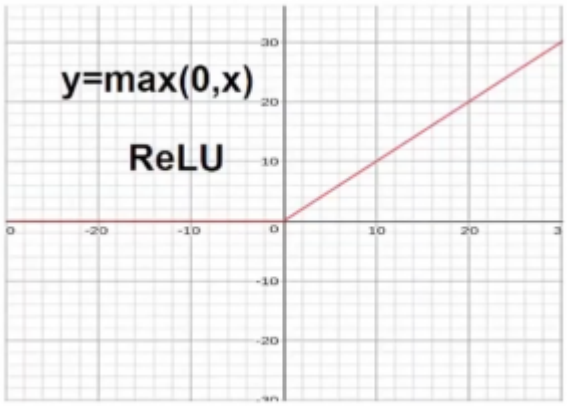
(a) Graf binární funkce [10]



(b) Graf Sigmoid funkce [10]



(c) Graf Tanh funkce [10]



(d) Graf ReLU funkce [10]

Obrázek 2.8: Grafy aktivačních funkcí

2.7 Algoritmus zpětného šíření chyby

Algoritmus zpětného šíření chyby (z angl. Backpropagation) je algoritmus neuronových sítí pro učení s učitelem za pomocí gradientního sestupu. Byl vymyšlen v 70. letech jako způsob obecné optimalizace pro komplexní diferenciální funkce. Nicméně jeho důležitosti významně přispěla až trojice vědců David E. Rumelhart, Geoffrey E. Hinton a Ronald J. Williams, kdy ve své práci *Learning Representation by Back-Propagating Errors* tento algoritmus vyzdvihly a komunita strojového učení začala této metodiky ze široka využívat. Tento algoritmus totiž jako jeden z prvních metod využívaných ve strojovém učení byl schopen demonstrovat, že umělé neuronové sítě jsou schopny učit se netriviálním příznakům na datech. [12]

V zásadě algoritmus funguje tak, že v případě, kdy máme dánu neuronovou síť společně se spe-

cifickou funkcí k výpočtu chyb v síti (označována někdy jako účelová či cenová funkce), tato metoda dopočítává gradient těchto chyb s ohledem na jednotlivé váhy v síti. Označení zpětné šíření chyby vychází ze způsobu výpočtu zmíněného gradientu v sítí směrem od konce na začátek - jinými slovy je gradient poslední vrstvy s vahami vypočítán jako první a gradient první vrstvy jako poslední.

Takto jsou mezivýsledky jedné vrstvy znovu využity k získání výsledků její předchozí vrstvy. Tento přístup k výpočtům umožňuje efektivní a účinný výpočet gradientů všech vrstev oproti poměrně výpočetně hrubému počítání gradientů v každé z vrstev zvlášť.[12]

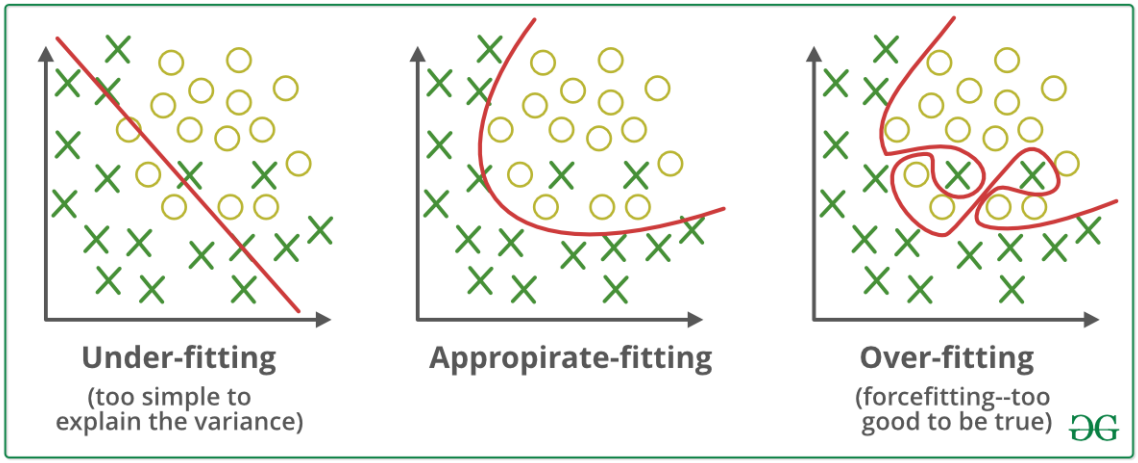
2.8 Přeučování, nedoučování a jejich prevence

Přeučování (z angl. overfitting) ve strojovém učení je konceptem, kdy model sítě je vůči svým trénovacím datům příliš konkrétně naučen. Když k tomuto dojde, model není efektivně schopen korektně predikovat jemu prezentovaná data, které pro něj byla v době tréninku skryta, čímž se absolutně ztrácí jeho pointa a proč byl vůbec trénován. Generalizace tohoto modelu k novým datům je předně to, co ze strojového učení dělá efektivní nástroj pro predikci a klasifikaci dat v běžné praxi.

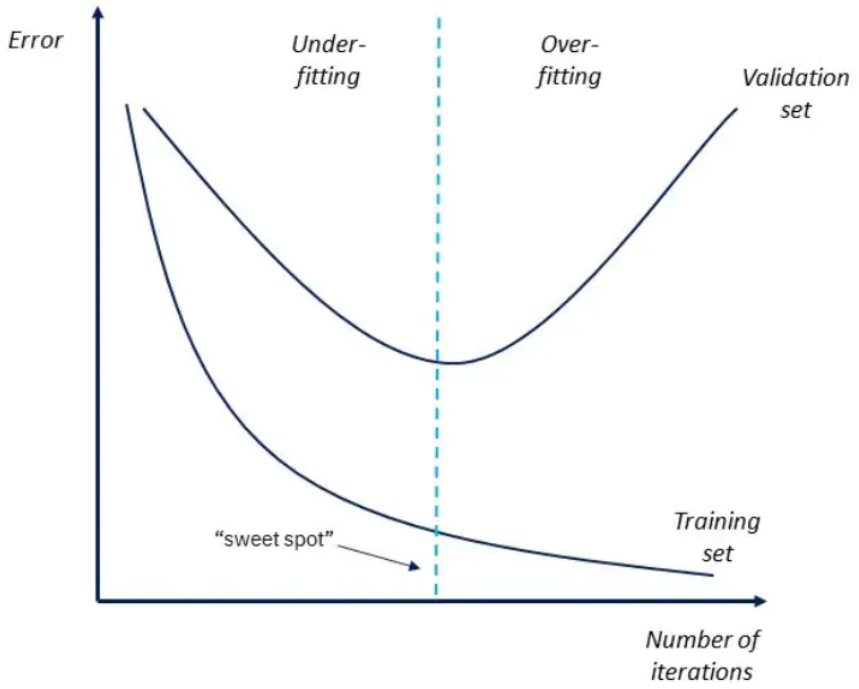
Celý proces přeučování nastane tehdy, kdy modelu jsou předloženy data určené k tréninku a model se příliš konkrétně naučí vlastnosti těchto dat - kupříkladu začne brát více zřetel na šum v obsažených datech nebo na informace, které pro budoucí predikce nejsou relevantní. Když se model místo skutečně užitečných dat v souboru datasetu naučí rozpoznávat tento šum, je označován právě jako přeučený model (z angl. overfit model).

Opačným problémem přeučování je naopak **nedoučování** či **nedostatečné učení** (volně přeloženo z angl. underfitting). Dochází k němu, když je trénování naopak zastaveno předčasně a model se naučit důležité vlastnosti nestačil a nebo když jsou parametry či vstupní data sítě nedostatečně odladěny pro vytvoření vazeb pro správnou predikci - typicky vysoká/nízká míra učení nebo nevhodně upravená data.

V obou případech však dochází k nesprávnému trénování modelu, který je pro nově předložená data neúčinný - zpravidla je přesnost velice nízká nebo přímo mizivá. Proto je v problematice učení jako takového kladen dosti vysoký důraz na různé **prevence přeučení**, neboť ruční kontrola trénování je extrémně časově náročná. Způsoby, kterými lze pozorovat, že u modelu dochází k přeučení nebo nedostatečnému učení, jsou křivky vyjadřující buďto chybovost modelu nebo vývoj ztrátové funkce (z angl. loss function), jejichž hodnoty jsou iterativně počítány v průběhu trénování modelu - tj. zároveň při trénování dochází i k průběžné validaci predikce modelu v jednotlivých epochách.[13] Takováto křivka je k nahlednutí v obrázku 2.10, kde je vyobrazen vývoj chybovosti modelu. Porovnání nedotrénováného, optimálního a přetrénovaného modelu je v obrázku 2.9. Takto lze zhodnotit i zmíněnou účelovou funkci, neboť by se i její křivka měla vyvíjet obdobným způsobem.



Obrázek 2.9: Příklady nedotrénováného, optimálního a přetrénovaného modelu [13].



Obrázek 2.10: Křivka chybovosti modelu [13].
Na této křivce jsou vyobrazeny i oblasti, kde je model přeučen nebo nedoučen, společně i s místem, kdy by se ideálně mělo trénování přerušit

2.8.1 Prevence přeučení

Pro prevenci těchto dvou zmíněných stavů existuje hned několik technik, kterých je využito při klasifikaci úloh reálného světa. Několik z nich je níže uvedeno společně s vysvětlivkami, proč jsou používány a jak fungují. Je nutno zmínit, že těchto technik existuje celá řada a proto jich je uvedeno pouze několik - zejména ty, co jsou nejvíce používány.

Předčasné zastavení (z angl. Early Stopping) je metodou, jež model automaticky zastaví ještě před tím, než se stačí naučit i nechtěné prvky vstupních dat - typicky zmíněný šum nebo nechtěná/irelevantní data. Tento proces samozřejmě nese sebou i určitá rizika, neboť může způsobit, že model bude zastaven příliš brzo a dojde místo k přeučení naopak k jeho protějšku - model bude nedostatečně naučen. Proto je nutné při použití této metody najít to pravé místo, kdy je vhodné model v učení včasně zarazit.[13]

Trénink s více daty. Tento přistup, jak jíž označení napovídá, je velice přímočarý - pokud je k dispozici větší objem dat, na kterých se model může naučit patřičné vlastnosti ke správné predikci neviděných dat, je přínosné je použít. Nicméně toto platí pouze pro případy, kdy je předem o datech jisté, že jsou nezávadná - tím myšleno, že se v datech nevyskytují příliš časté chyby nebo invalidní záznamy. Pokud totiž takovéto záznamy jsou do dat zahrnuty, je možné, že se tímto komplexita modelu navýší, což opět vede k přeučování.[13]

Umělé zvětšování dat je technikou, kdy data v rámci datasetu jsou omezena a čerpat z dalších validních a nových dat její variantou, neboť nejsou k dispozici. Principiálně jde o přidání dat do datasetu, u nichž je záměrně přidružen šum. Nicméně je nutno tuto metodu používat opatrně, neboť je velmi choulostivá na validitu dat, jež jsou modelu předávána. Taktéž je poměrně obtížné volit jak a jak moc budou data touto technikou v závěru ovlivněny - pokud je totiž zavedený šum příliš výrazný, model nebude schopen se efektivně naučit vlastnosti jemu předložených dat a opět dojde k přeučení.[13]

Regularizace je velice oblíbenou metodou z hlediska zamezení přeučování modelů. Její podstatou je zavedení určité formy penalizace na vstupní parametry modelu, což efektivně eliminuje množství nepotřebného šumu v datech.[13] V kontextu neuronových sítí existuje již několik variant této metody - jako například **Dropout**, což je technika, jež náhodně vylučuje část neuronů při trénování.

Z hlediska strojového učení, konkrétně zejména v rámci hlubokého učení, je naprosto běžné tyto metody kombinovat, neboť jejich korektní využití může výrazně zlepšit predikční schopnosti trénovaného modelu. Ve většině případů je však nutné všechny tyto techniky pečlivě zkoušet a odlaďovat, aby přínos neměl za následek degradaci modelu. Ve většině moderních architektur komplexních neuronových sítí je vždy nějaké takovéto techniky využito a proto se považuje takřka jako samozřejmé, aby byla součástí každé neuronové sítě.

Kapitola 3

Konvoluční neuronové sítě a jejich podstata

Konvoluční neuronové sítě (dále jako **CNN**) jsou varianty biologicky-inspirované neuronové sítě k rozpoznávání a vyhodnocení obrazových dat – v přírodě je možné tento princip přirovnat k biologickému neuronu, jen na více vrstvách/úrovních. Ve skutečnosti je vesměs možné hovořit o standardních neuronových sítích, u kterých došlo pouze k obohacení o několik unikátních vrstev, které CNN od ostatních značně odlišují. A právě tento typ sítě se ukázal jako velice efektivní v oblastech rozpoznávání obrazu a jejich klasifikace. CNN již byly proto úspěšně aplikovány nejen v systémech pro rozpoznávání obličejů, objektů, pohybu a dopravních značek, ale i pro zlepšení vnímání okolí a strojového vidění – například u samořídících automobilů či bezpečnostních kamer.

Nejprominentnější unikátní vrstvou je vrstva konvoluční. Stěžejní vlastností této vrstvy je především specifický výběr příznaků rozpoznávaného objektu – například obrázku – a propagace tohoto výběru následující vrstvě. Tato propagace je zvlášť efektivní kvůli upřednostnění význačných prvků tohoto výběru a potlačení jeho nevýznamných částí – to v závěru znamená, že pro každou následující vrstvu CNN, tedy v podstatě každý neuron, namísto přijetí informace pouze jediné předchozí vrstvy, jsou přijaty informace hned z několika předchozích vrstev najednou [5] – tedy je vždy v následující jednotce snímáno širší pole dat než v jednotce předchozí. Tento princip je detailněji vysvětlen v podkapitole **3.1.1 Konvoluce a konvoluční vrstvy**.

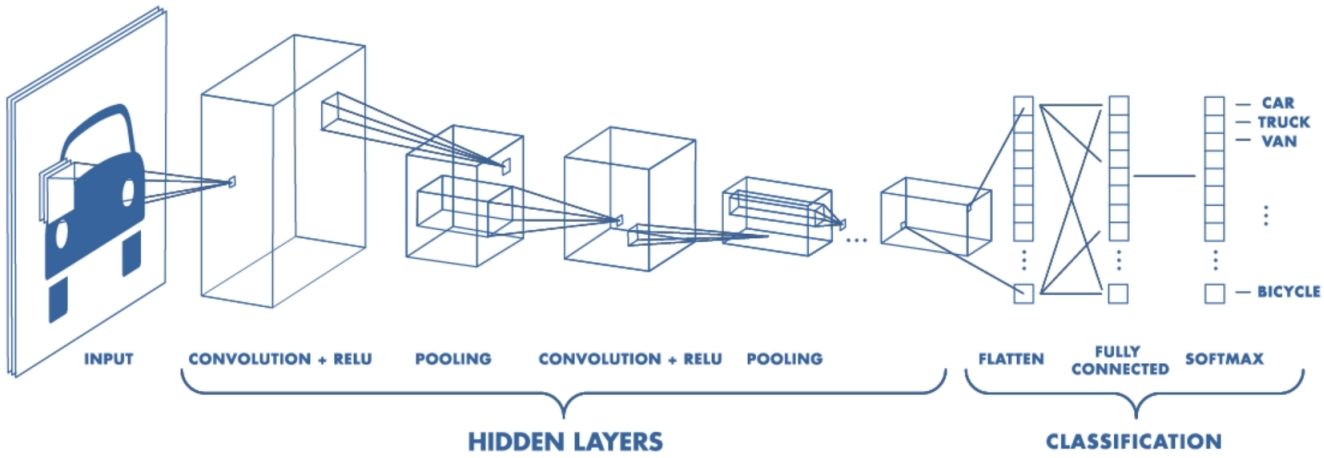
Postupně ve vyšších vrstvách je kombinace těchto polí neurony přebírána a každá taková následující vrstva je čím dál tím více schopná učit se abstraktní vlastnosti daného obrázku – od stínů a barev, přes rohy a hrany, až po kontury obličeje a konkrétní tvary. Díky tomuto aspektu je tedy dáno, že k největšímu výpočetnímu úkonu dochází právě v této vrstvě, což přímo definuje příčiny vzniku konvolučních sítí.

3.1 Struktura CNN

CNN jsou tvořeny zejména dvěma částmi - jednak souhrnem **skrytých vrstev s funkčními částmi** na výběr vlastností ze vstupních dat a **klasifikační části**. První zmíněná část má za úkol provést sérii konvolučních a sdružovacích operací (z angl. Convolutions and Pooling) při kterých jsou nalezeny požadované vlastnosti. Tento prvek, jež tyto konvoluce provádí , se právě nazývá **konvoluční vrstva**. Příkladem může být právě automobil - v tomto kroku je síť schopna extrahovat prvky jako jsou kola, dveře, světla, okna nebo i barvu či značku.

Po každé takovéto konvoluční vrstvě následuje vrstva **škálovací**. To je především dáno tím, že výsledkem konvoluce jsou data, jež z hlediska rozměru narůstají. Je tedy nutné tento rozměr průběžně redukovat kvůli algoritmické náročnosti.

Druhou částí je zmíněná **klasifikace**, resp. **klasifikační část**. Zde plně propojené vrstvy fungují jako klasifikační prvky nad extrahovanými vlastnostmi, kde dochází k přiřazení pravděpodobnosti rozpoznávanému objektu, zda je skutečně tím, co algoritmus předpokládá že je viz obrázek 3.1.[14]

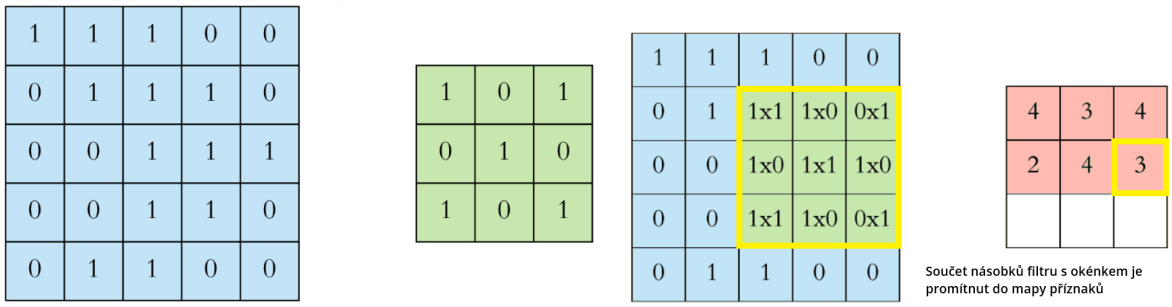


Obrázek 3.1: Zjednodušený nákres procesu rozpoznávání objektu v CNN [15]

3.1.1 Konvoluce a konvoluční vrstva

Konvoluce jako taková v matematickém kontextu odkazuje na kombinaci 2 funkcí, jejichž výsledkem je jiná třetí funkce. V případě CNN jde o aplikace **filtru**, také označovaném jako kernel, na **vstupních datech** k vytvoření **příznakové mapy** (volně přeloženo z Feature Map). Jak je v obrázku 3.1 vyobrazeno, nejprve jsou vstupní data předána **konvoluční vrstvě**. Samotná konvoluce probíhá krokově - a to tak, že zmíněný filtr je posouván skrze celými vstupními daty tzn. celým objektem. V každém místě, kde je filtr aplikován, dochází k násobení matic (vstup a filtr) a výsledky jsou ukládány do mapy. V tomto procesu o filtru hovoříme jako o poli vnímání, jež z hlediska terminologie vychází z biologického předobrazu - neuronových buněk. V části *a* obrázku 3.2

je znázorněn **filtr** - zelený čtverec se **vstupními daty** - modrý čtverec. V části *b* je poté znázorněn postup, jakým probíhá extrakce příznaků ze vstupních dat - filtr je postupně posouván nad vstupními daty a je tvořena zmíněná **příznaková mapa**, jež je v reprezentována červeným čtvercem. [14, 16]

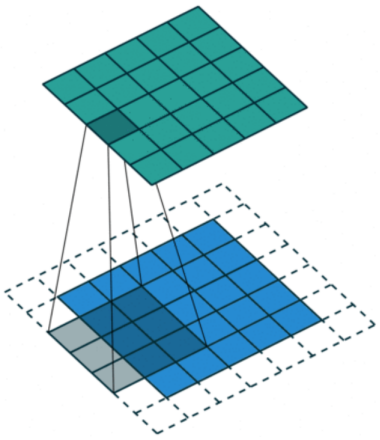


(a) Vstup s filtrem pro konvoluci

(b) Proces konvoluce nad částí vstupních dat

Obrázek 3.2: Znázornění příznakové mapy získané z násobení vstupu s filtrem [16]

V části *b* je filtr vždy posouván o stejnou hodnotu. Tomuto posunu se obecně říká **krok** (z angl. Stride). Ve většině případů je hodnota tohoto kroku rovna 1, aby filtr procházel obrázek pixel po pixelu. Proto je dáno, že pokud je tento krok zvětšován, překryv při procházení obrázku je řidší. To má efektivně za následek zmenšování příznakové mapy, což není žádoucí vlastností. Proto je uplatňováno techniky **vyplňování** (z angl. Padding). Vyplňování je proces, kdy dojde k obklopení celého obrázku pixely převážně s nulovou hodnotou - existují i situace, kdy je přínosné si hodnotu výplně zvolit. Takto zůstává nejen mapa prostorově stále stejně velká, ale také to zaručuje, že filtr v kombinaci s krokem nebude nikdy větší, než vstupní obrázek tzn. vždy se do vstupního obrázku vejde viz ilustrace 3.3.[14]

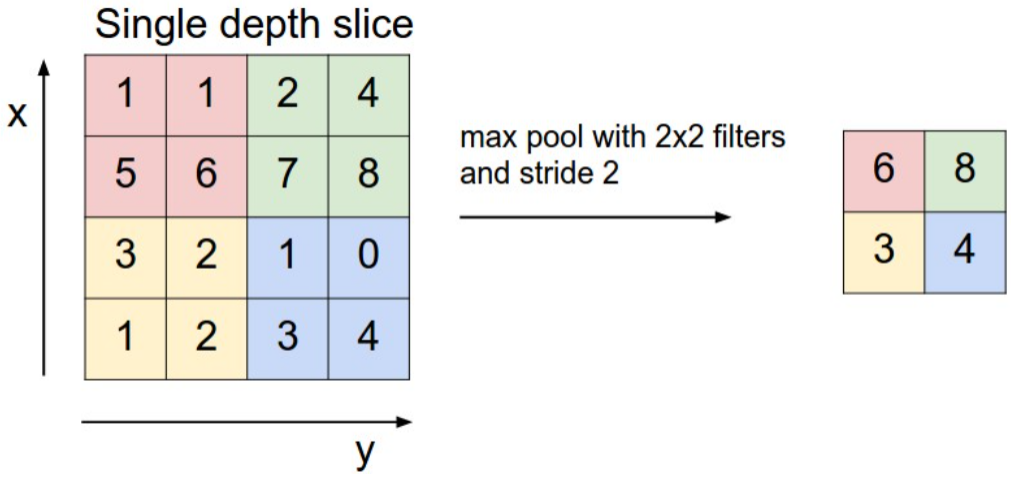


Obrázek 3.3: Průchod filtru s výplní = 1

Ve skutečnosti však konvoluce probíhá trojrozměrně, neboť je vstup (typicky obrázek) reprezentován šířkou, výškou a hloubkou - hloubka je počet barevných filtrů červená, zelená, modrá. Je tedy použito na vstupu hned několika konvolucí, kde každé z nich náleží vlastní filtr. To má za následek vytvoření sady příznakových map, které jsou v závěru sečteny a jejich hodnoty reprezentovány jako finální výsledek (= výstup) konvoluční vrstvy. Jako každá neuronová síť, i CNN obsahuje aktivační funkci - ta může být kupříkladu typu ReLU viz 2.6.4.

3.1.2 Škálovací vrstva

Po konvoluční vrstvě následuje vrstva škálovácí (volně přeloženo z angl. Pooling layer). Ta zpravidla není v modelu celé sítě obsažena pouze jedna, nýbrž jsou jí prokládány všechny konvoluční vrstvy v pořadí konvoluce->škálování. Funkce této vrstvy je průběžné zmenšování rozměru vstupního obrázku za účelem redukce parametrů a výpočtů v sítí. Díky tomuto je trénovací čas zkracován a v jistém smyslu i prezentuje jednu z možností kontroly sítě z hlediska přeučování. Nejfrekventovanější typ využívaného škálování je dle škálování dle maximální hodnoty (volně přeloženo z angl. max pooling). V principu jde o výběr nejvyšší hodnoty každého prostupu filtru konvoluční vrstvy - velikost tohoto filtru však musí být pro tuto vrstvu předem známá. Tento celý proces, jak již bylo zmíněno, zmenšuje příznakovou mapu, přičemž zároveň udržuje důležité informace získané konvolucí. Náhled funkcionality této vrstvy je v obrázku 3.4 [14]



Obrázek 3.4: Průchod filtru o velikosti 2x2 s krokem = 2 [17]

3.1.3 Klasifikační část

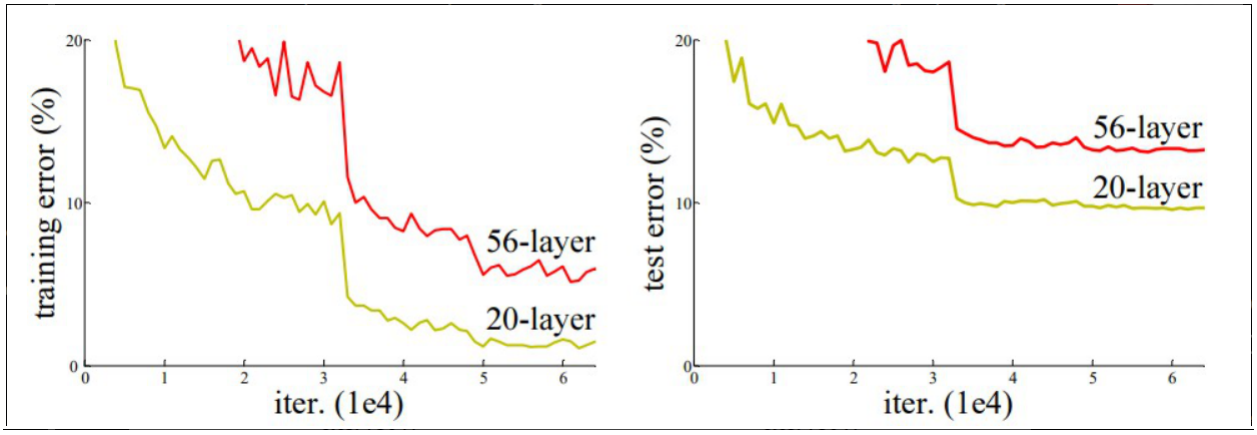
Po proběhnutí konvoluce a škálování se mapy dostávají do klasifikační části, jež se skládá ze série funkcí a plně propojených vrstev. Vždy je totiž nutné tyto data převést do takové podoby, aby byly těmito vrstvami zpracovatelné - k tomu je využito právě funkce `Flatten`, což je možné

volně přeložit jako *zploštění*, kdy vstupní data v trojrozměrném formátu jsou převedeny na data ve formátu jednorozměrného vektoru. Ty jsou v závěru předány posledním vrstvám CNN, které jsou složeny z plně propojených vrstev, které nejsou nic jiného, než dopředné neuronové sítě - jinými slovy jde o tradiční případ vícevrstvého perceptronu. V závěru klasifikace jsou finální data předány druhé z funkcí **softmax**, která je využita na závěrečnou klasifikaci vstupních dat v situacích, kdy existují více než 2 klasifikační třídy - což je u obrazových dat v drtivé většině případů naprosto běžné.[14, 16]

Kapitola 4

Reziduální model

Po uvedení CNN architektury AlexNet, která jako první roku 2012 vyhrála v soutěži ImageNet, se každá následující vítězná síť pokoušela o redukci chybovosti a tedy zlepšení přesnosti modelu přidáním většího množství vrstev. Tohle do jisté míry fungovalo, nicméně jen v rámci modelů, jež neoplývali velkým počtem těchto vrstev - jinými slovy nebyly příliš hluboké. Ono v okamžiku, kdy je těchto vrstev navrstveno na sobě příliš velké množství, dochází poměrně k častému problému v oblasti hlubokého učení, a tím je buďto problém mizejícího nebo výbušného gradientu (volně přeloženo z Vanishing/Exploding gradient). V mizejícím případě je hodnota gradientu takřka sražena na nulu, ve výbušném případě naopak gradient extrémně vzroste. Tudíž pokud je neustále přidáváno dalších vrstev, nedochází k poklesu, ale naopak ke zvyšování chybovosti modelů, což přirozeně není žádoucím účinkem. Tento problém je nastíněn v obrázku 4.1, kde je v grafu patrný rozdíl mezi 20-ti vrstvou a 56-ti vrstvou architekturou a jejich mírou chybovosti. Po podrobné analýze příčině nárůstu chybovosti bylo usouzeno, že tento případ způsoben přeučením není a že skutečně jde o problémy gradientů.[18, 19]



Obrázek 4.1: Porovnání chybovostí 20-ti vrstvé a 56-ti vrstvé architektury [20]

To dalo za vznik novému typu architektur, jež jsou všeobecně označovány jako **reziduální modely**, zkráceně ResNet, jež vznikly pod záštitou pobočky Microsoft Research v roce 2015. Forma řešení, jež ResNet představoval, bylo vytvoření tzv. reziduálních bloků.

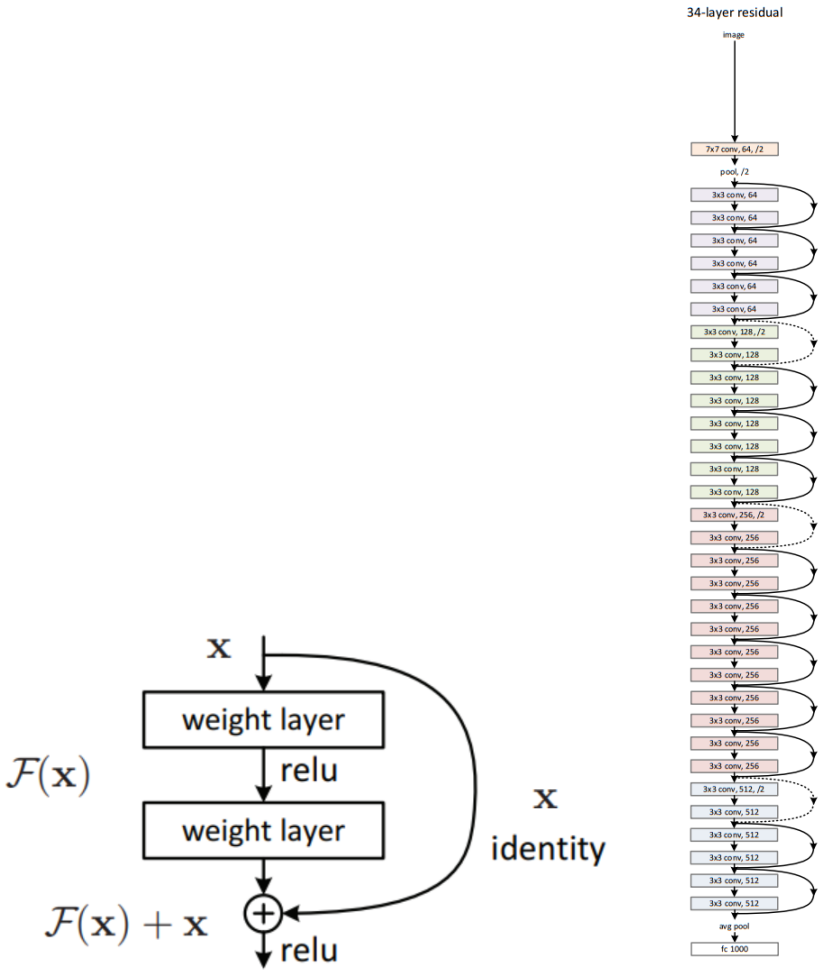
Reziduální blok je řešení jak degradačního problému hlubokých architektur, tak i mizejícího a výbušného gradientu použitím techniky *skokového spojení* (volně přeloženo z angl. Skip Connection). Principem této techniky je, jak již název napovídá, přeskočení několika vrstev v trénování za účelem spojení se přímo s výstupy těchto přeskočených vrstev. To tedy zamezuje vzniku mizejících nebo výbušných gradientů ve velmi hlubokých sítích, neboť je gradient mezi vrstvami předáván, aniž by docházelo k jeho graduálnímu zmenšování či zvětšování - pokud je hodnota gradientu opakovaně násobena hodnotou v rozsahu 0 až 1, dochází k jeho mizení, pokud větší než 1, dochází k jeho nárůstu (explozi). Výhodou tohoto přístupu je také přetřídění celé architektury - v případě, kdy nějaká z vrstev představuje negativní dopad pro síť a není tedy pro síť prospěšná, je přeskočena díky regularizaci.[18, 19] Schéma reziduálního bloku a jeho zavedení v reziduální sítí o 34 vrstvách je vyobrazen v obrázku 4.2 níže. Zvětšený obrázek sítě B.2 je obsažen v příloze B.

Jak je možné pozorovat v obrázku 4.2, v rámci skoku je provedena lineární aktivační funkce, též označována jako **identita**. Tato identita má za úkol uchovávat gradient a také případně upravovat rozměry vrstev, které jsou tímto skokem přeneseny na vstup dalšího reziduálního bloku. V okamžiku, kdy se výstup předchozí vrstvy parametry nerovná vstupu nadcházející vrstvy (ať už v rámci rozměrů samotného vstupu nebo z hlediska dimensionality), je nutné parametry tohoto výstupu upravit, aby byly s parametry vstupu ekvivalentní. Pokud je výstup neekvivalentní z hlediska rozměrů samotných dat, je do následující vrstvy vyplněn nulovými hodnotami do požadovaných rozměrů v rámci identity - zkrátka je provedena funkce *výplň* (Padding). V případě, kdy se liší výstup a vstup z hlediska dimensionality (tj. počet kanálů), je v rámci identity provedena konvoluce s filtrem o rozměrech 1x1 - ta rozměrově data nijak neupraví, neboť filtr o rozměrech 1x1 prochází data pixel po pixelu. Nicméně je tato konvoluce schopna sloučit kanály na požadovanou hodnotu - takovou, jakou následující vrstva na vstupu očekává. Samozřejmě je také možné, že může být výstup odlišný oběma způsoby - pak je akorát provedena kombinace obou zmíněných řešení.

Hodnota, která by měla být předána na vstup následující vahové vrstvy dalšího reziduálního bloku, je součet výstupu předchozího reziduálního bloku a původního vstupu tohoto reziduálního bloku - ten je předán v identitě. Tato hodnota je v obrázku 4.2 definována jako $F(x) + x$. Pro usnadnění bude výpočet této hodnoty zadefinován jako funkce $H(x) = F(x) + x$. Vzhledem k tomu, že je vztah pro tuto funkci znám, je tedy patrné, že funkce $F(x)$ musí být rovna $H(x) - x$, neboť se jedná o mapování o jeden krok zpět. Tudíž tato funkce $H(x)$ slouží jako reference pro funkci $F(x)$ a proto má tato funkce určitý předpoklad pro to, jaké by měla nabývat hodnoty - v tomto případě se jedná o aktivační funkci **ReLU**. Tato reference je označována jako **reziduum**, proto také název reziduální blok/síť. V závěru je tímto přístupem dosaženo řešení degradačního problému a snížené algoritmické náročnosti. Celý tento proces se nazývá **reziduální mapování**. [29, 18, 19]

Reziduální bloky však nejsou jedinou výhodou, kterou ResNet představuje. Ačkoliv je pravdou, že je

to jeho předností, ResNet je v dnešní době preferován i z dalších jiných klíčových důvodů. Zejména tedy kvůli jeho samotné podpoře z hlediska externích frameworků, přičemž není řeč pouze o Tensorflow - dalšími vhodnými příklady mohou být frameworky *Caffe* či *PyTorch*, jež ResNet poskytují taktéž. Další výrazně prominentní výhodou je možnost využití ResNetu jakožto před trénovaného modelu. V tomto kontextu je řeč o **přenosovém učení**, jež poskytuje naučené vlastnosti již natrénovaného modelu, modelu nenatrénovanému. A právě tato vlastnost poskytuje uživateli velice široké možnosti jak a na kterých datech model trénovat - a to i za pomoci dalších **přídavných vrstev a metod**, jež může vývojář zařazovat do modelu dle libosti. Velice populární volbou pro vstup do prostředí přenosového učení je ResNet50, neboť kromě všech již zmíněných vlastností je také jedním z prvních reziduálních modelů, které již byly považovány z hlediska vrstev za velmi hluboké - právě proto je jej využito i v této práci.

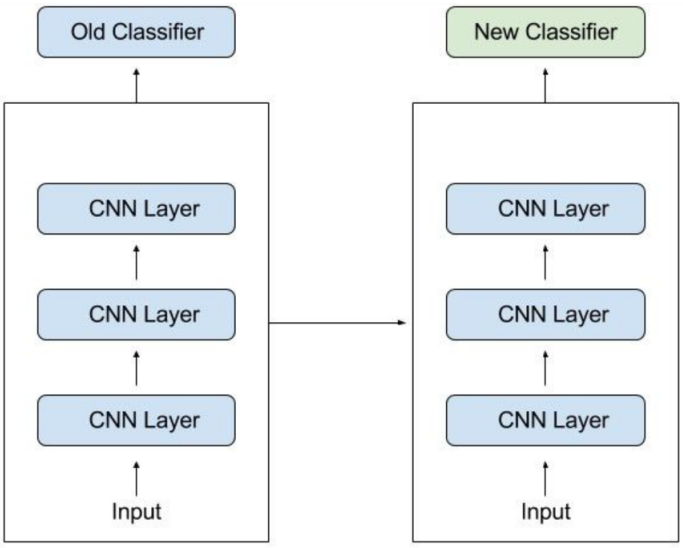


Obrázek 4.2: Reziduální blok a ResNet34 [20]

4.1 Přenosové učení

Přenosové učení je metodou předávání naučených struktur jedné sítě síti druhé. Toho je dosaženo sdílením naučených vah prvních vrstev původního modelu. Tyto váhy jsou novému modelu předány a dochází k učení a zpětnému šíření pouze u posledních plně propojených vrstev. V podstatě jde o formu recyklace naučených poznatků, kterých první síť nabyla v rámci učení, aby druhá síť mohla tyto poznatky zužitkovat. Tento přístup je především praktikován v případech, kdy je k dispozici velice omezené množství dat a původní síť byla trénována na podobných datech, nebo když je žádané převzít nějaký z natrénovaných modelů a pouze jej v rámci několika vrstev přeučit na nových datech - takovým typickým případem jsou právě převzaté modely ResNet v rámci frameworku Tensorflow, jež byly už předem natrénovány.

Metodika přenosového učení však není vázána pouze na data obrazového charakteru - právě přenosové učení umožňuje velice efektivně vytvářet modely schopné přesně rozpoznávat prvky přirozené řeči - aplikace tohoto druhu jsou v dnešní době poměrně běžné. Příkladem může být hlasové rozpoznávání nebo ovládání elektronických zařízení hlasem. Zjednodušené schéma této metody je v obrázku 4.3. Na tomto obrázku je možné pozorovat, že naučené prvky staré sítě jsou předány nové síti a ta je začleňuje mezi své naučené příznaky. [21]

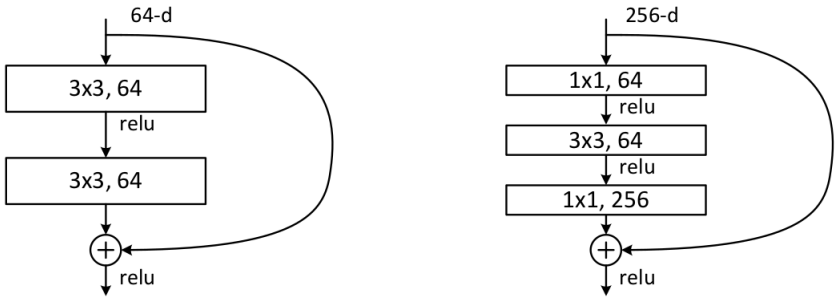


Obrázek 4.3: Schéma přenosového učení

4.2 Popis modelu ResNet50

Jak bylo výše zmíněno, v této práci je využito residuálního modelu ResNet, konkrétně ResNet50. Jedinými rozdíly od předchozích modelů ResNet je vyšší počet vrstev (v tomto případě je vrstev

50) a vylepšení reziduální blok. Namísto skoku přes 2 vrstvy, reziduální blok ResNetu50 přeskakuje vrstvy 3 viz obrázek 4.4.[19] Nastínění architektury tohoto modelu je popsáno v obrázku 4.5 níže.



Obrázek 4.4: Porovnání reziduálního bloku předchozích ResNetů (vlevo) s ResNet50 (vpravo) [19]

ResNet50

layer name	output size	50-layer
conv1	112×112	7×7, 64, stride 2
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times3$
conv3_x	28×28	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times4$
conv4_x	14×14	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times6$
conv5_x	7×7	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times3$
	1×1	average pool, 1000-d fc, softmax

Obrázek 4.5: Síť ResNet50

Ve třetím sloupci, přesněji u vrstev *conv2-x* až *conv5-x*, jsou hranatými závorkami prezentovány tzv. *reziduálními bloky* - nejedná se o nic jiného, než o seskupení sady konvolučních vrstev s parametry takovými, jaké jsou v tabulce.

Z tabulky je tedy patrné, jak je model konstruován - vrstvy typu typu max/average pooling či aktivační funkce do samotného součtu vrstev modelu nejsou započítávány. Tudíž je výčet vrstev následující:

- První vrstvou je conv1, tedy konvoluční vrstva, s rozměry posuvného okénka 7x7 s počtem filtrů 64 a posunem o hodnotě 2 - zde se tedy nachází **1** vrstva
- Poté příchází první sada inception bloků conv2_x - zde tedy jsou konvoluční vrstvy s parametry okének 1x1, 3x3 a 1x1. Počet filtrů je 64, 64, 256. Tyto 3 vrstvy se zde nacházejí 3x - celkem **9** vrstev.
- Dále nastupuje sada conv3_x s parametry 1x1, 3x3, 1x1 a 128, 128, 512. Každá z vrstev 4x, celkem tedy dalších **12** vrstev.
- V následující sadě conv4_x jsou přítomny konvoluční vrstvy s parametry 1x1, 3x3, 1x1 a 256, 256, 1024, každá z nich 6x - celkem **18** vrstev.
- A v poslední sadě conv5_x se nacházejí vrstvy s parametry 1x1, 3x3, 1x1 a 512, 512, 2048 - zastoupeny 3x každá - celkem **9** vrstev
- Poslední vrstvou zakončující celou sérii vrstev v modelu je plně propojená vrstva čítající 1000 uzlů (nebo-li neuronů) společně se softmax funkcí v závěru - **1** vrstva

4.3 Přídavné vrstvy a metody

V modelu ResNet je však používáno i celé palety přídavných funkcí či vrstev, které například snižují celkovou chybovost modelu, zamezují modelu v přeučování nebo transformují data mezi vrstvami - ty jsou popsány níže. Je však nutno podotknout, že výčet níže uvedených metod a vrstev není úplný, neboť jsou zde vyzdviženy zejména ty prvky, jež byly využity v této práci.[22]

ReduceLROnPlateau funkce má na starosti dynamickou úpravu míry učení ve vztahu k hodnotě *účelové funkce* a je velice úzce spjatá s funkcí EarlyStopping. Běžně se tyto funkce používají v kombinaci, kdy je u obou sledována stejná metrika - hodnota zmíněné účelové funkce `loss`. Pokud tato hodnota klesne pod či vzroste nad určitou hranici, je parametr sítě `míra učení` penalizován. Je však možné touto funkcí i hlídat kolísavost trénování. Pokud se síť stává příliš nestabilní, může být míra učení penalizována také. Tímto způsobem je po určitém množství proběhnutých epoch možné síť jemněji trénovat, aniž by docházelo k přeučování.[22]

BatchNormalization je vrstvou, jež má na starost normalizaci vstupních dat. Podstatou její funkčnosti je přeškálování dat na hodnoty o průměru blízko `0` se standardní odchylkou `1` v obou číselných osách. Standardně je používána mezi jednotlivými vrstvami, přičemž jejím vstupem jsou

právě výstupní hodnoty předchozí vrstvy - zejména z důvodu, aby učení v rámci každé z nich probíhalo více nezávisle a všechna data byla škálována do stejného rozsahu. Je možné jí však využít i jako krok k normalizaci samotných vstupních dat sítě, pokud je zařazena ještě před první konvoluční vrstvu - této metody využívá zmíněný ResNet. Do jisté míry i nahrazuje vrstvu Dropout.[22]

EarlyStopping je funkcí, jejímž úkolem je síť zastavit v trénování, pokud pozorovaná metrika klesne (nebo přeroste) přes určitou hranici. Standardně je tato funkce párována s hodnotou účelové funkce *loss*, kde je sledováno její minimum. Pokud se hodnota této funkce na minimum dostane a poté se nijak výrazně v následujících epochách trénování nezlepšuje (nebo dokonce začíná narůstat), je trénování zastaveno. Tímto způsobem je možné síti zamezit v přeučování nad trénovacími daty, aniž by bylo nutné přesně nastavovat počet epoch tréninku.[22]

Dense vrstva je označení pro plně propojenou vrstvu - ta, jak bylo zmíněno, transformuje výstupní data předchozích vrstev na `m` rozměrné vektory - je tedy využívána ke transformaci dat. Může však také provádět operace jako škálování, otočení či vektorové posunutí.[22]

Dropout je vrstvou, jenž náhodně vybírá část neuronů a ty jsou deaktivovány. Tento proces však probíhá proměnlivě - v každém kroku trénování je výběr opakován a jsou deaktivovány další, náhodné neurony. Tudíž tento výběr není proveden pouze jednou, kde je vyloučena jedna podmnožina neuronů trvale, ale dynamicky při trénování sítě, kdy je podmnožina těchto neuronů volena náhodně v každém kroku. Frekvence těchto deaktivací je standardně předem definována - nejčastěji je deaktivováno kolem 20% neuronů, to se však může případ od případu lišit. Pointou této vrstvy je zamezení přeučování tím, že se záměrně síť zužuje z hlediska kapacity, tzn. počtu neuronů, při trénování a síť se proto tak velmi úzce neváže na konkrétní příznaky vstupních dat. Tím, že jsou některé tyto příznaky vyloučeny, je využito všech vstupních dat a k přeučování nedochází (nebo je alespoň významně omezeno). [22]

Kapitola 5

Implementace neuronové sítě

K implementaci samotného modelu sítě byl zvolen programovací jazyk Python - jeho upřednostnění nad ostatními programovacími jazyky byla především jeho jednoduchost, přehlednost a přímočarost psaného kódu. Dále stojí za zmínku i jeho rozsáhlá podpora s ním kompatibilními moduly a frameworky. nehledě na jeho oblíbenost v rámci komunity strojového učení a hlubokého učení. V neposlední řadě je také jeho velmi přednostní výhodou podpora frameworků a knihoven, jež byly využity pro vznik této práce.

Samotnými frameworky k implementaci modelu a práci s daty jsou Keras a Tensorflow. Pro trénování a následnou validaci dat byl zvolen model reziduální konvoluční neuronové sítě ResNet50 4.2. Tento model byl převzat z přímé implementace frameworku Keras a následně doupraven tak, aby byl bezchybně spustitelný na lokálním zařízení - z hlediska úprav se jedná zejména o doplnění správných závislostí k frameworku Tensorflow a správného volání jednotlivých vrstev, podmodulů a funkcí. Ze stránky původní architektury modelu k žádným zásadním změnám nedošlo.

Pro samotné trénování a validaci bylo použito již zmíněných frameworků Keras a Tensorflow, které k těmto úkonům využívají výpočetní síly GPU a její možnosti masivní paralelizace. K práci s daty bylo využito zejména výchozích možností jazyka Python a nativních modulů, jako je kupříkladu modul Decimal, v kombinaci s externími moduly či knihovnami umožňující pokročilejší a efektivnější matematické operace s daty či grafické vykreslování. Takovou knihovnou je například Numpy.

Modul Decimal byl použit z důvodu zachování přesnosti při výpočtech s daty, knihovna Numpy byla použita z důvodu usnadnění výpočtů a manipulace pro výslednou formu dat k tréninku i validaci. Dále bylo také využito knihovny OpenCV k úpravě výsledných dat do požadovaného formátu a k vizualizaci vzorků dat k prezentačním účelům. Zvolené datasety, na kterých byl experiment praktikován, byly MSR Action 3D a KARD. Trénování i validace byla provedena na osobním zařízení s parametry:

- **Model** - ASUS ROG G752VSK

- **CPU** - Intel Core i7-7700HQ @ 2.80 GHz
- **GPU** - NVidia GTX 1070 s 8GB VRAM
- **RAM** - 2x8 GB o frekvenci 2333MHz
- **Disk** - HGST, 1TB, 7200 RPM
- **OS** - Windows 10 Home

Z hlediska organizace projektu byla funkcionalita celé práce rozdělena na ucelené části, které se nacházejí v kořenovém adresáři projektu. Samotná data - jak výchozí, tak předzpracovaná - jsou dělena do příslušných složek opět v rámci kořenového adresáře. Strukturalizace a dělení dat na příslušné kategorie probíhá dynamicky tj. při spuštění skriptu `dataset_computation.py` je strom adresářů pro další kroky výpočtu vždy vygenerován. Podrobnější popis struktury projektu je zahrnut v příloze A.

5.1 Popis použitého modelu

Model ResNet50 byl pro trénink přiřazen do sekvenčního modelu, který v tomto případě slouží pouze jako *kontejner* pro před-zhotovený model. Váhy již před natrénovaného ResNetu byly zmrazeny, aby nedošlo k jejich přeučení. K tomuto výslednému modelu byly v závěru navíc přidány plně propojené vrstvy a funkce, jež jsou zmíněny v podkapitole **4.3**. V pořadí jsou přidány vrstvy **Flatten->Dense(512)->Dropout(0.2)->Dense(počet validačních tříd)**. Jejich účelem je zpřesnění výsledků a korektní klasifikace výsledků do přesného počtu klasifikačních tříd. Jako optimizér byl zvolen **Adam** - zejména z toho důvodu, že při experimentování byly výsledky za použití tohoto optimizéru nejpřesnější a trénování bylo nejstabilnější - celkem byly odzkoušeny optimizéry Adam, Adagrad a SGD, z nichž právě Adam vycházel nejlépe. Poté jé k modelu dále přiřazena dvojice funkcí zpětného volání **EarlyStopping** a **ReduceLROnPlateau**.

5.2 Použité nástroje

Keras [22] představuje moderní API vytvořené za účelem jednoduchosti, flexibility a rozšiřitelnosti práce pro lidi, nikoliv pro stroje v Pythonu. Důležitým faktem pro samotný Keras je co nejvíce možná optimalizace, tedy v tomto případě redukce, zátěže jak paměti zařízení, tak 'paměti' uživatele - tj. pro běžné případy užití Keras nabízi velice jednoduchá řešení a přístup, jež značně omezí potřebné zásahy k implementaci řešení ze strany uživatele. Výhodou je i samozřejmě velmi podrobná, ale zároveň přehledná, dokumentace a návody inspirované reálnými řešeními zkušenějších uživatelů.

Jádrem samotného API je jeho implementace na vrstvě strojového učení v podání Tensorflow. To znamená, že Keras řešení pro implementaci neposkytuje a ke svému chodu je zapotřebí další, již pro Keras navržený, framework - v případě této práce Tensorflow. **Tensorflow** [23] je kompletní open source framework určen pro aplikace v okruhu strojového učení. Primární vlastností tohoto frameworku je ekosystém vytvořený konkrétně k tvorbě nejpokročilejších a tedy nejmodernějších aplikací strojového učení.

Důvodem chvalné oblíbenosti tohoto nástroje je jeho abstraktní přístup k vývoji aplikací strojového učení - především je nutno zmínit, že framework je schopen plně využít výkon GPU. V krajních případech je možné tento framework využít i se samotným CPU, nicméně se prozatím jedná v porovnání s výkonem grafické karty o velice neefektivní a zdlouhavý proces.

Z hlediska podpory je vývoj v Tensorflow možný v jazycích Python, C++, JavaScript i Javě - nicméně pro Javu Tensorflow negarantuje záruku stabilního chodu. Ačkoliv se zdají být možnosti profilace jazykové podpory poněkud úzké, právě z důvodu oblíbenosti profesionální komunitou jsou tvořeny různá propojení pro ostatní jazyky a platformy (jako jsou například MATLAB, C# nebo i Haskell).

Numpy [24] je základním balíčkem přednostně pro vědecké práce v rámci Pythonu. Stručně řečeno jde o knihovnu poskytující podporu z hlediska vícerozměrných polí, nad kterými otevírá možnosti svižných výpočetních metod, ať už matematického, logického či tvarového charakteru. Taktéž poskytuje moderní přístup k třídění, selekci dat, operacím týkající se I/O, základy statistiky i lineární algebry a mnoho dalšího. Celou tuto škálu možností sjednocuje klíčový objekt unikátní pro Numpy a tím je *ndarray object*. Díky tomu je možné s jeho prací prakticky naprosto volně operovat s daty v ekosystému Numpy.

OpenCV [25] je knihovnou, shrnující zejména metody využitelné v oblasti počítačového vidění a strojového učení. Účelem jejího vzniku bylo poskytnutí snadno použitelné, byť komplexní, infrastruktury pro zefektivnění vývoje v aplikacích, jež se na strojové učení a počítačové vidění zaměřují.

Podpora této knihovny zahrnuje rozhraní pro C++, Python, Javu a MATLAB, přičemž samotná knihovna je původně psána v C++ - pro ostatní jazyky je funkční skrze mapování - tj. vytvoření programového rozhraní, jež umožňuje užití původní funkcionality knihovny i v jiných jazycích, než ve kterém byla napsána.

5.3 Datasety

K otestování přesností použité sítě byly vybrány 2 datasety - *MSR Action3D* a *KARD*. **KARD** dataset byl pořízen S. Gaglio et al. v roce 2014 za pomocí pohybového senzoru Microsoft Kinect. Jednotlivé záznamy jsou třízeny celkem do 18 aktivit. Těmito aktivitami jsou: *horizontální mávnutí rukou, vysoké mávnutí rukou, mávnutí rucemi, chycení čepice, vysoký hod, kreslení X, kreslení fajfky, vržení papíru, kop dopředu, boční kop, vezmutí deštníku, ohnutí, tlesknutí, chůze, telefonní*

hovor, napití se, sednutí, stoupnutí. Každá z těchto aktivit je vykonána **vždy** 3 krát, přičemž každou tuto aktivitu vykonalo dohromady 10 subjektů. Celkem se tedy sada akcí skládá z 540 záznamů. Každá tato sada je v datasetu prezentována ve variantě souřadnic kloubů kostry, mp4 videoklipů, hloubkových map a snímků sestávajících z kombinace souřadnic kloubů a hloubkových map. Celkem tedy dataset tvoří 2160 záznamů. Pořadí ucelených sekvencí tj. *počet subjektů * počet opakování* je organizováno tak, jak byly sekvence pořízeny.

Pro účely této práce bylo použito varianty *souřadnice kloubů kostry*. Celkový počet kloubů činí 15, kde každý kloub je složen z trojice hodnot - souřadnice X, Y a hloubka. V každém takovémto souboru je každý kloub reprezentován zmíněnou trojicí hodnot, přičemž ucelený snímek tedy tvoří 15 těchto trojic. Snímků je vždy v každém souboru různé množství, tudíž není naprosto triviální stanovit, z kolika snímků je celý dataset složen. Příklad vykreslené kostry je zahrnut v příloze B.

MSR dataset byl pořízen Dr Wanqing Li pod záštitou Microsoftu v roce 2010 taktéž za pomocí zařízení, jež má pohybový senzor srovnatelný s Microsoft Kinect a tedy se jedná o zařízení obdobného typu. V tomto případě je počet aktivit 20. Tyto aktivity jsou: *vysoké mávnutí rukou, horizontální mávnutí rukou, úder kladivem, chycení rukou, úder směrem kupředu, vysoký vrh, kreslení X, kreslení fajfky, kreslení kruhu, tlesknutí, mávnutí oběmi rucemi, boxování z profilu, ohnutí se, kop kupředu, kop z profilu, jogging, tenisové švih, tenisové podání, golfový švih, zvednutí a hod* . Každá z těchto aktivit je vykonána 2-3 krát, přičemž které akce v rámci opakování byly vynechány dopředu není známo. Každou tuto akci vykonalo 6-10 subjektů. Opět předem není známo, které a kolik subjektů bylo vynecháno v rámci opakování. Celkem se tato sada akcí skládá z 567 záznamů. Pořadí ucelených sekvencí tj. *počet subjektů * počet opakování* je organizováno tak, jak byly sekvence pořízeny.

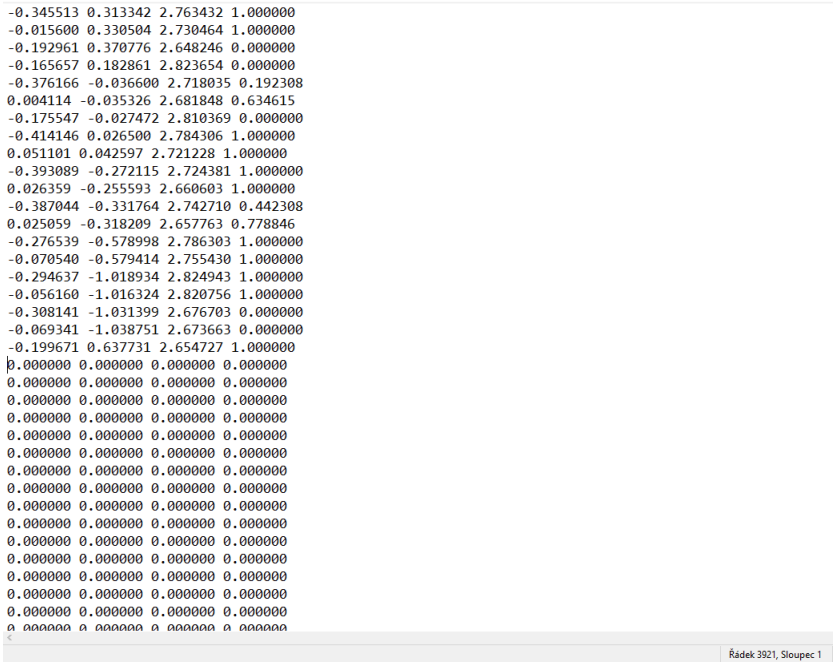
Celkový počet kloubů činí 20, kde každý kloub je složen z trojice hodnot - souřadnice X, Y a hloubka. V každém souboru je každý kloub reprezentován zmíněnou trojicí hodnot, přičemž ucelený snímek tedy tvoří 20 těchto trojic.

Formát jmenné konvence souborů je `a<akce>-s<subjekt>-e<epizoda>-skeleton3D/realworld.txt`, kde: `<akce>` je číslo akce, `<subjekt>` je číslo subjektu a `<epizoda>` je číslo opakování. Příkladem tedy soubor `a02-s04-e01-skeleton3D.txt` jsou data akce č. 2, jež vykonal 4. subjekt, prvního opakování.

5.4 Předzpracování dat

Pro každý z datasetů byla implementována jednotná funkce, jež má na starost správné načítání - v rámci tohoto načítání bylo nutno také počítat s chybovostí dat, kterou je zejména dataset MSR nechvalně známý. Přesněji řečeno se jedná o chybové záznamy, které se v datech projevovaly jako série nulových hodnot napříč značným množstvím řádků - těmito řádky jsou myšleny

opět trojice hodnot tj. souřadnice *X,Y a hloubka*. Jeden z takovýchto souborů je například soubor `a13_s09_e03_skeleton3D`, kde tyto nulové řádky drtivě převažují ty validní viz obrázek 5.1.



Obrázek 5.1: Řádky s nulovými hodnotami datasetu MSR.

Tyto řádky byly náležitě odfiltrovány a tedy z finální podoby textových dat připravených ke zpracování také odstraněny. Nicméně, zvláště pro dataset MSR byla metoda načítání dále upravena, neboť samotným autorem tohoto datasetu byl vydán textový soubor, jehož náplní je seznam "validních", tedy použitelných, záznamů - jeho úlohou je především opět odfiltrovat vhodné záznamy od záznamů chybových. Některé ze záznamů jsou autorem záměrně vynechány, neboť podle autora [citace] *Kostry úplně chyběly nebo byly velice chybné pro použití.* Rozhodl jsem se tedy tyto invalidní soubory taktéž vynechat a zohlednit autorův přístup k experimentům. Jméno a popis tohoto souboru jsou zahrnuty v příloze A.

Dalším úskalím z hlediska datasetu MSR je jeho nepřesný, nebo spíše tedy nekonzistentní, počet subjektů a opakování každé z akcí. Proto byla tato metoda obohacena o další logickou vrstvu, jejíž intencí je tyto počty hlídat, aby skutečně docházelo ke správnému dělení vůči experimentům.

Po ošetřeném načítání je v podstatě forma zpracování pro oba datasety téměř shodná. Vzhledem k formě jmenné konvence jednotlivých souborů jsou data členěny do příslušných kategorii dle názvu. Toto rozdělení je v práci implementováno tak, že po zmíněném načtení dat jsou jednotlivé akce tříděny do nového adresáře, jež je pojmenován dle toho, s kterým datasetem je pracováno. Pokud je tedy zpracováván dataset MSR, adresář bude pojmenován `MSR-subsets`, pokud dataset KARD, tak `KARD-subsets`. V každém tomto adresáři je rozdělení vedeno dle toho, o kterou sadu akcí v rámci testování jde, zda jde tedy o trénink či testování a kterým akcím přísluší které záznamy. Detailnější

popis je zahrnut v příloze A.

Následně jsou pro oba datasety provedeny stejné úpravy dat. Při načtení je zvolen vždy 1. sada souřadnic, tedy první kloub, vůči kterému jsou vypočítány relativní pozice všech ostatních kloubů. U tohoto přístupu transformace dat není nutné volit centrální kloub, neboť tato operace je provedena sekvenčně pro všechny klouby, jež jsou v každém ze snímků vedeny. To znamená, že po provedení této matematické operace pro 1. kloub, je toto zopakováno pro každý další následující kloub, aby relativní pozice kloubů byly ve vztahu "každý s každým". Je samozřejmostí, že jakmile jsou vypočítány pozice jednoho kloubů vůči všem ostatním, v další iteraci, tedy v dalším kroku výpočtu, je již vypočítaný kloub vynechán. Tímto se v podstatě stává ze statických souřadnic jednotlivých kloubů sada směrových vektorů. Nezáleží tedy poté na tom, pod jakým úhlem je subjekt snímán a jak je v závěru kostra natočená - nezáleží ani na tom, zda se subjekt pohybuje zleva doprava či naopak. Tato úprava má nicméně za následek zvětšení celkového objemu dat, protože dochází ke generaci směrových vektorů v kombinaci bez opakování.

Výsledkem této operace je tedy hodnota, kterou se celkový počet trojic souřadnic vynásobil. Po dosazení tedy dostáváme $K = 7$ pro KARD a $K = 9{,}5$ pro MSR. Důvodem pro úpravu dat tohoto typu je minimalizace variance jednotlivých výsledků - tím, že jsou vektory vypočítány, se data stávají více abstraktními, neboť už pro data není přílišně směrodatná samotná výška (případně šířka a vzdálenost od senzoru) jednotlivých subjektů.

Další standardní úpravou pro data je zpravidla normalizace dat do určitého rozsahu - z hlediska numerických hodnot je tento rozsah v drtivé většině stanoven na interval -1,1. Nicméně vzhledem k reziduálnímu modelu popsanému v 4, tento krok není striktně vyžadován - součástí modelu ResNet je vrstva BatchNorm 3.1.1, která tuto úpravu vykonává dynamicky pro každý batch. Z hlediska urychlení výpočtu je samozřejmě možné tyto data upravit na stanovený rozsah, nicméně jediný skutečný účinek, jež by tento krok přinesl, by bylo urychlení tréninku v několika prvních epochách, než se data normalizují za pomocí zmíněné vrstvy BatchNorm.

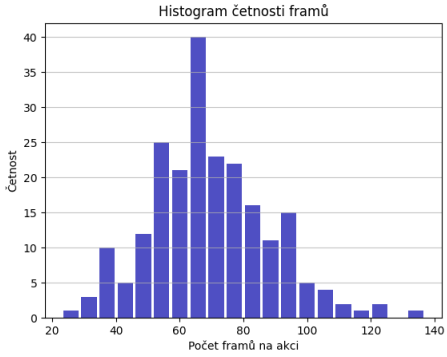
Aby model byl schopen se na těchto datech natrénovat a poté provést validaci, je každopádně nutné, aby všechna data (tj. jednotlivé záznamy) měla jednotný rozměr. Vzhledem k datům obou datasetů, kde data jsou reprezentovány sériemi kloubů s trojrozměrnými souřadnicemi *X,Y,hloubka*, je nutné tyto data převést do trojrozměrných polí *snímky*klouby*souřadnice*, kde :

- **snímky** jsou počet celkového počtu snímků v rámci záznamu - tento počet je vypočítán tak, že celkový počet trojic souřadnic kloubů je vydělen počtem těchto kloubů (15*7 pro KARD, 20*9,5 pro MSR)

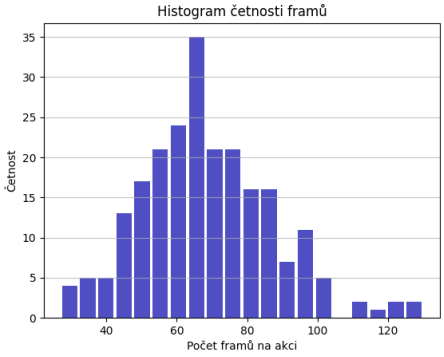
- **klouby** jsou počet kloubů v rámci dat datasetu

- **souřadnice** je trojice X,Y,hloubka

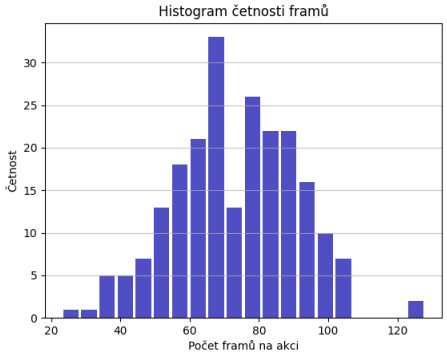
Ve výsledku jsou tedy data vždy ve formátu *snímky*105*3* pro KARD a *snímky*190*3* pro MSR. Hodnota *snímky* nebyla stanovena náhodně a byla taktéž vypočítána - pomocí histogramů. Vizualizace histogramů je k dispozici níže v obrázku 5.2 .



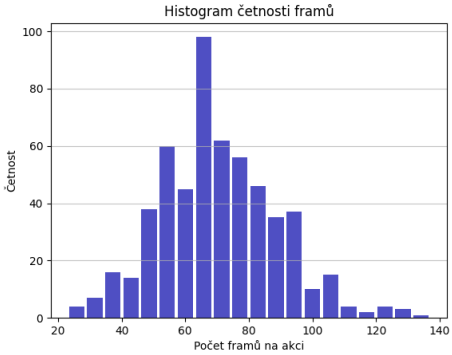
(a) **MSR histogram** - sada akcí AS1



(b) **MSR histogram** - sada akcí AS2



(c) **MSR histogram** - sada akcí AS3

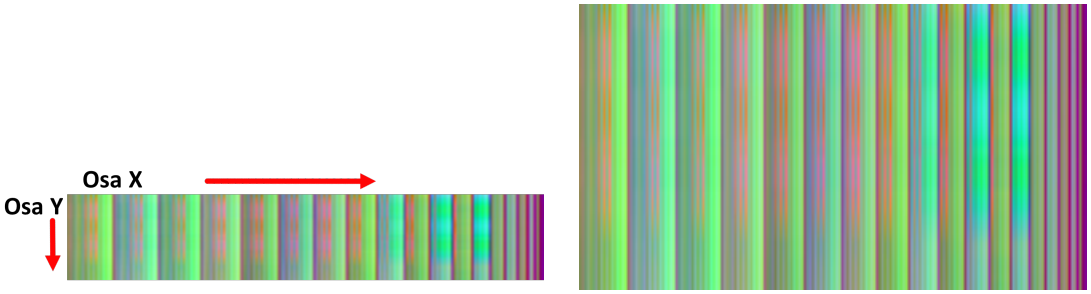


(d) **MSR histogram** - sada akcí AS4

Obrázek 5.2: Vizualizace všech 4 histogramů k datasetu MSR

Histogramy jsou v tomto případě využity tak, že napříč všemi záznamy byly nalezeny počty snímků všech akcí tak, aby mohla být dopočítána optimální hodnota pro *snímky*. Není totiž žádoucí ani záznamy příliš ořezat, tedy zkrátit o velké množství hodnot, ani roztáhnout, tedy vygenerovat velké množství hodnot, jež by poté ovlivnili hodnotu dat. Je tedy patrné, že tato hodnota nebyla zvolena pro každý dataset stejně a je výsledkem početní operace, která se odvíjí od typu zpracovávaného datasetu.

V závěru předzpracování byla tedy tyto data upravena za pomocí knihovny OpenCV - konkrétně pomocí metody *resize*, která všechny záznamy upravila do požadovaného, jednotného, tvaru. Zvolená metoda byla výchozí bilineární interpolace. Příklad této úpravy u jedné z akcí je zobrazen níže v obrázku 5.3. Je nutno podotknout, že tyto příkladová data byla normalizována v rámci hodnot v rozsahu 0-255, aby zobrazení bylo vhodně porovnatelné.



(a) **Před úpravami** (b) **Po úpravách**

Obrázek 5.3: Data před a po úpravě funkcí resize
RGB kanály tohoto obrázku reprezentují jednotlivé dimenze vektorového prostoru vlastností. Osa X představuje výčet směrových vektorů kloubů akce, osa Y je časovou osou - reprezentuje tedy počet snímků akce. Obrázky v porovnání jsou ve stejném poměru z hlediska velikosti - je tedy patrné, že data po úpravě jsou zvětšeny pouze v rámci počtu snímků.

Hodnoty výpočtů za pomocí histogramů jsou obsaženy v souboru `config.yaml`, který byl použit nejen k uchování hodnot z této operace - detailnější popis je obsažen v příloze A. Tyto operace bylo nutné spustit pouze jedenkrát a to formou spuštění sady hlavních skriptů - detailnější popis skriptů je taktéž obsažen v příloze A.

Data, jež byla takto upravena, byla následně uložena do složek původního rozdělení ve formátu `.npy` - tento formát byl zvolen z důvodu snížení celkové velikosti dat a také z důvodu zjednodušení převodu dat na tensory pro neuronovou síť. Originální textová data byla tedy nahrazena těmito upravenými daty ve vygenerované struktuře.

Samotná data nicméně nejsou z hlediska zastoupení vždy stejná - vzhledem ke zmíněné nekonzistenci dat (týkající se v tomto případě pouze datasetu MSR) je takřka běžné, že různé akce jsou reprezentovány různými počty záznamů - nejčastěji se jedná o výskyty, kdy záznamů je buďto 26, 27, 29 a nebo 30. V jednom z případů je záznamů i pouze 20 - konkrétně jde o akci *kop z profilu*.

5.5 Testování

Po dokončení předzpracování dat byla použita jednotná síť pro oba datasety, která se lišila pouze v nastavení hyperparametrů. Nicméně způsob otestování sítě, tedy zvolené metody či protokoly, byly pro oba datasety takřka totožné.

Pro dataset MSR byl použit původní přístup k otestování z hlediska rozdělení dat, jež je zmíněn v práci [26]. Toto konkrétní rozdělení je následující: Všechna data byla uspořádána do jednotlivých *podmnožin* (z angl. subset), které pro jednoduchost budou označovány jako AS1, AS2 a AS3. Tyto podmnožiny vždy sestávají přesně z 8 akcí, které byly právě do své podmnožiny přiřazeny podle

toho, zda se jedná o podobné, triviální akce (AS1 a AS2) či komplexní pohyby (AS3). Dále byly tyto data rozčleněny do *experimentů*, které se lišily způsobem rozdělení dat samotných mezi části určené k trénování a testování - a to dle počtu opakování nebo počtu samotných subjektů, jež akce vykonávaly. Celkem jsou tyto experimenty 3 - experimenty A, B a C. Experimenty A a B sestávají z rozdělení dat dle počtu opakování a to v poměrech 1:2 pro experiment A, kde menší část je určena pro trénování a větší pro testování, a 2:1, kde větší část je určena pro trénování a menší pro testování. Experiment C byl rozdělen způsobem *křížové validace* (z angl. cross-validation) - polovina subjektů každé akce byla určena pro trénování a polovina pro testování. Tento způsob je však méně přesný, neboť na rozdíl od experimentů A a B, je v trénování i testování pracováno vždy s neviděnými subjekty - tj. subjekt, jež byl obsažen v trénování, není obsažen v testování a je pro neuronovou síť nový. Každý subjekt totiž vykonává danou akci vždy trochu jiným způsobem (tzn. při vykonávání akce dělá jiné pohyby) než nějaký další, jiný subjekt. Nad rámec těchto původních rozdělení dle originální práce byl proveden ještě jeden způsob testování - AS4. Tato podmnožina je složena z objemu všech 20 akcí pro experimenty A,B i C.

Pro dataset KARD byl použit taktéž původní přístup k otestování dle originální práce [27], který je téměř totožný se způsobem dělení pro dataset MSR. Rozdílem je počet akcí, který u MSR činí 20, u KARD 18. Opět jsou však akce rozděleny do celků po 8 akcích, jež jsou sdruženy dle podobnosti či komplexity, a to do stejných experimentů A,B i C. Posledním rozdílem je pouze to, že na rozdíl od původní práce k MSR, byl autory proveden i test na podmnožině AS4, tedy je možné porovnat výsledky i z těchto experimentů. Rozdělení do jednotlivých podmnožin pro oba datasety je k dispozici v tabulkách 5.1 a 5.2 níže.

Pro účely testování byl trénink každého z experimentů každé podmnožiny (z angl. subset) proveden 10x a naměřená přesnost byla zprůměrována. Z hlediska epoch (tj. iterace) je počet pro každý experiment rozdílný a není stanoven jejich přesný počet - to, kdy má být trénování sítě zastaveno, je ošetřeno *zpětným voláním* EarlyStopping. Ta je nastavena tak, aby při náznaku sítě schylovat se k *přeučování* 2.8 bylo učení ukončeno. Je však stanoveno maximální množství epoch na hodnotu 1200, aby případné trénování netrvalo do nekonečna a bylo vždy ukončeno. Této hranice však žádný z experimentů nedosáhl. Přesné hodnoty pro tuto funkci v rámci každého datasetu jsou určeny experimentováním.

Aby průběh trénování byl stabilnější a bylo možné dosáhnout lepších výsledků, je využito taktéž zmíněné funkce ReduceLROnPlateau. Přesné hodnoty i pro tuto funkci jsou určeny experimentováním.

Tabulka 5.1: Rozdělení akcí datasetu MSR

Sada akcí AS1	**Sada akcí AS2**	**Sada akcí AS3**
Horizontální mávnutí rukou	Vysoké mávnutí rukou	Vysoký vrh
Úder kladivem	Chycení rukou	Kop kupředu
Úder směrem kupředu	Kreslení X	Kopnutí z profilu
Vysoký vrh	Kreslení fajfky	Jogging
Tlesknutí	Kreslení kruhu	Tenisový švih
Ohnutí se	Mávnutí oběmi rucemi	Tenisové podání
Tenisové podání	Boxování z profilu	Golfový švih
Zvednutí a hod	Kop kupředu	Zvednutí a hod

Tabulka 5.2: Rozdělení akcí datasetu KARD

Sada akcí AS1	**Sada akcí AS2**	**Sada akcí AS3**
Horizontální mávnutí rukou	Vysoké mávnutí rukou	Horizontální mávnutí rukou
Mávnutí oběmi rucemi	Chycení čepice	Vysoký hod
Kreslení X	Kreslení fajfky	Kreslení fajfky
Kop kupředu	Kop kupředu	Vržení papíru
Ohnutí se	Boční kop	Vezmutí deštníku
Chůze	Ohnutí se	Telefonní hovor
Telefonní hovor	Tlesknutí	Napití se
Stoupnutí	Sednutí	Sednutí

5.6 Výsledky testování a zhodnocení

Testování samotné již probíhalo dynamicky při trénování, přičemž výsledná přesnost byla poté zvlášť ještě určena po dokončení tréninku. Bylo tedy patrné z průběžných výsledků, zda síť se učí na datech správně a nedochází k zásadním chybám. V jiných ohledech testování nijak obtížné nebylo, pouze bylo nutné formou experimentování s hyperparametry přijít na vhodné hodnoty, při kterých si síť dokázala s daty poradit - tímto jsou myšleny zejména hodnoty typu jako je míra učení `LR` (z angl. Learning Rate), kde bylo nutné přijít na optimální rychlost učení sítě - pokud byla tato hodnota příliš nízká, trénování probíhalo extrémně zdlouhavě, pokud příliš vysoká, síť nebyla schopna učit se vlastnostem k rozpoznání akcí a přesnost byla mizivá. Finální hodnota pro `LR` byla tedy stanovena na číslo *0,0005*, neboť při této rychlosti učení docházelo k nejlepšímu trénování sítě.

Další takovou funkcí bylo de facto přímé zpomalování učení, tedy dynamické snižování míry učení. Konkrétně šlo o vymezení parametru `patience` a `factor`. Parametr `patience` určuje, kolik

epoch má při trénování vynechat, než začne opětovně měřit, jak efektivně se síť učí. Hodnota `factor` s tímto úzce souvisí a určuje, jak moc se má síť penalizovat v případě aktivace této funkce.

5.6.1 Testování s původními daty

První část testování je vykonána s původními daty - tj. žádná data nebyla dodatečně vygenerována pro trénování. Výsledky těchto testování jsou v tabulkách 5.3 a 5.4. Procentuální hodnoty jsou výsledkem výpočtu přesnosti Sparse Categorical Accuracy knihovny Tensorflow, jež dopočítává frekvenci korektních klasifikací na základě porovnání nejpravděpodobnější hodnoty s pravdivou hodnotou.

Tabulka 5.3: Výsledky testování MSR v procentech

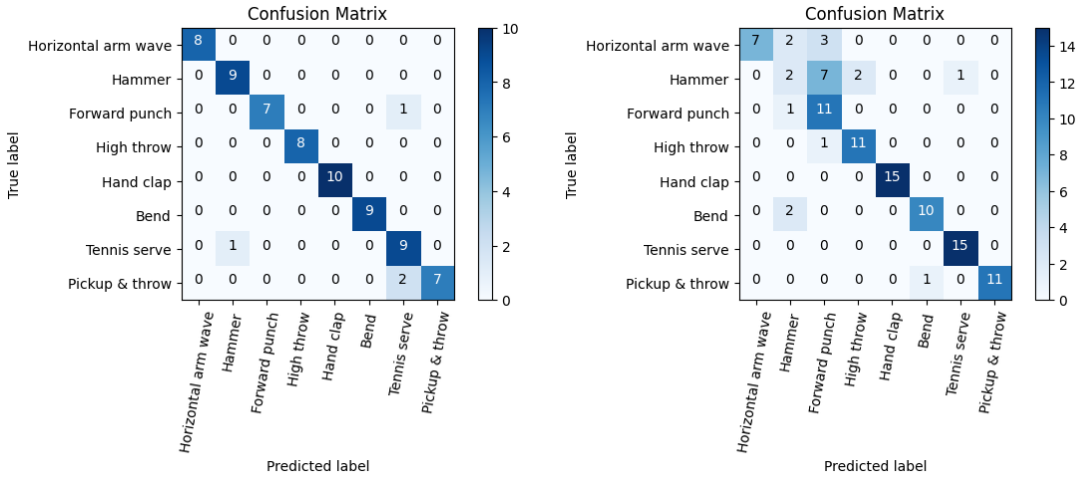
	Sada AS1	**Sada AS2**	**Sada AS3**	**Všechny akce AS4**
Experiment A	88,4	81,6	93,3	75,9
Experiment B	91,4	82,5	94,1	87,1
Experiment C	80,6	78,8	88,9	65,0

Tabulka 5.4: Výsledky testování KARD v procentech

	Sada AS1	**Sada AS2**	**Sada AS3**	**Všechny akce AS4**
Experiment A	99,9	99,6	96,5	97,8
Experiment B	100,0	100,0	96,9	98,8
Experiment C	98,9	99,3	88,5	89,4

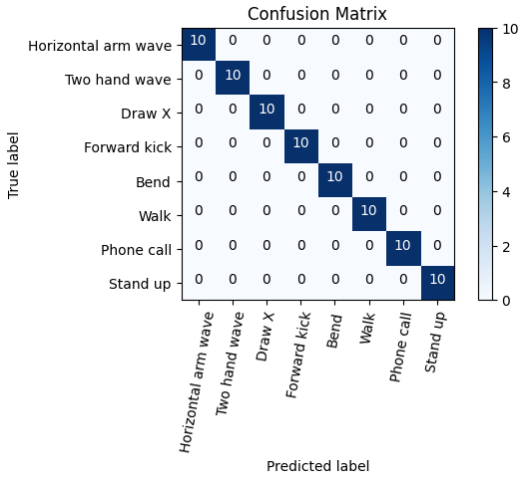
Z výsledků v tabulkách je tedy patrné, že v obou případech , tedy v obou datasetech, jsou zpravidla lepší výsledky v situacích, kdy pro trénování je přiřazeno větší množství dat (Experiment B). Jako očekávané lze i vzít výsledky křížové validace (Experiment C), neboť u nich je korektní validace mnohem vyzývavějším úkolem pro síť, než u předchozích experimentů, kde (v ideálním případě) je každá akce každého subjektu zastoupena jak v sadě pro trénování, tak testování.

Mezi samotnými datasety KARD a MSR je v mnohých případech schodek v přesnosti - to je nejspíše způsobeno tím, že MSR je, bohužel, nechvalně známý pro své chybové záznamy, které nejsou, kromě neprominentnějších chyb, nijak zvlášť filtrovány. Naopak KARD dataset je profiltrován již autorem, je tedy zcela pochopitelné, že výsledky v tomto případě jsou téměř stoprocentní u všech experimentů. Přesnosti jednotlivých akcí je možné vidět v obrázku 5.4. Je nutno také podotknout, že zmíněné matice pocházejí ze sady akcí AS1, zbytek těchto maticí pro ostatní sady a experimenty bude zahrnut v příloze C.

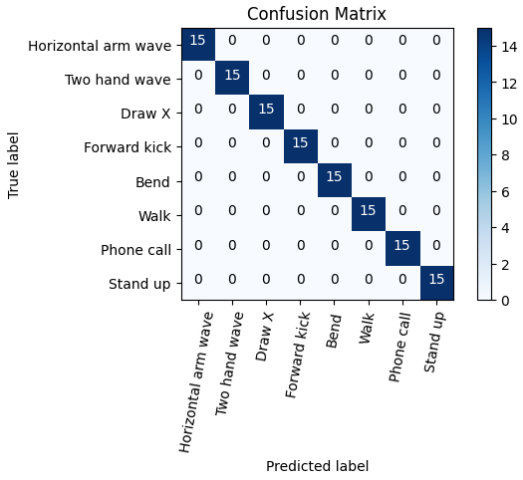


(a) **MSR** - experiment B

(b) **MSR** - experiment C



(c) **KARD** - experiment B

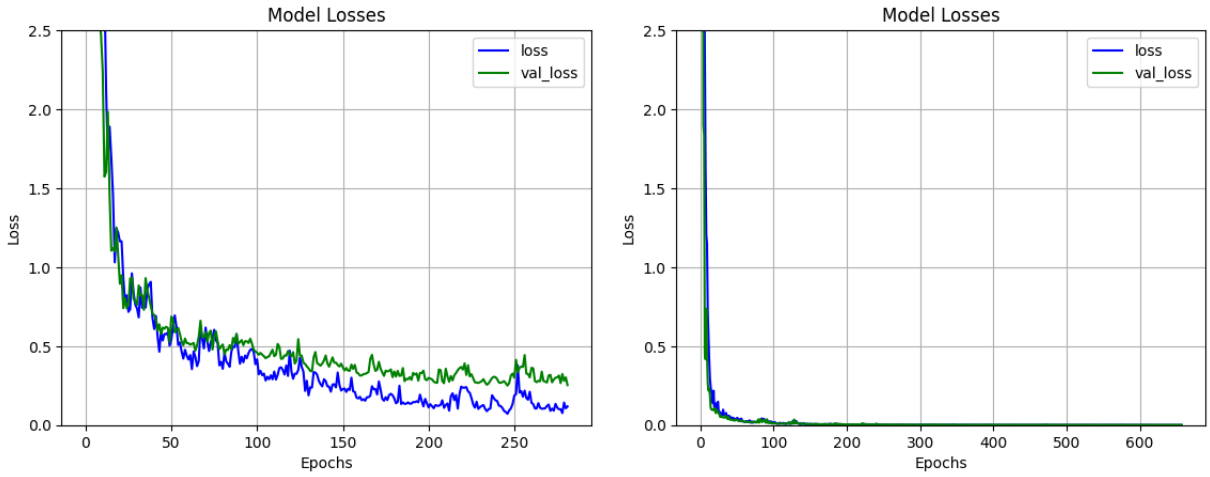


(d) **KARD** - experiment C

Obrázek 5.4: Porovnání experimentů B (2:1) ve prospěch trénování a experimentů C (cross-validation) pro datasety MSR a KARD

Lze tedy pozorovat, že pro dataset KARD byla úspěšnost velice dobrá u všech akcí. U MSR docházelo místy k invalidní validaci určitých akcí - příkladem záměna akce *Hammer*, za akce *Horizontální mávnutí rukou(Horizontal arm wave)*, *Úder směrem kupředu (Forward punch)* a *Vysoký vrh (High throw)*. To je dáno i tím, že tyto akce jsou ve všech případech vykonávány rukou (či rukami) a jsou tedy velmi podobného charakteru. Dalšími důvody je i to, že dataset MSR je vyzývavější, než-li dataset KARD a také fakt, že v MSR jsou obsaženy i již zmíněné chyby.

Z hlediska přeučení jsou v obrázku 5.5 doloženy i metriky účelových funkcí. Uvedené příklady pocházejí z trénování datasetu KARD i MSR, kdy v obou případech byla zvolena sada akcí AS1 při Experimentu B. Dle křivek je možné říci, že k přeučení nejpravděpodobněji nedocházelo. V příloze C je obsažena jedná celá sada těchto křivek pro sady akcí AS1, AS2, AS3 a AS4 obou datasetů.



(a) **MSR** - experiment B

(b) **KARD** - experiment B

Obrázek 5.5: Porovnání vývoje účelových funkcí pro datasety MSR a KARD

5.6.2 Testování s původními a vygenerovanými daty

Druhá část testování spočívala v tom, že k původním datům datasetu byly dodatečně vygenerovány data, jež se od původních odvíjely. Důvodem pro tento krok je pokud možno zamezit přílišné invariance sítě vůči samotným datům, aby byla síť schopna učit se obecněji a její využití se tím stalo univerzálnějším.

Data jsou vygenerována tak, že ke každé souřadnici každého kloubu je nezávisle vygenerován šum v určitém lineárním rozsahu. Čím větší je rozsah, tím větší je vygenerovaný šum a tím jsou data obecnější. Je nutné však brát také v potaz, že roztahovat toto číselné rozpětí do nekonečna není vhodné - mohlo by to způsobit, že data by byla až příliš ovlivněna šumem a přínos by byl naopak negativní. Experimentováním bylo zjištěno, že vhodným rozsahem pro generování šumu jsou hodnoty mezi 0 až 75 milimetry.

Dalším aspektem tohoto přístupu je možnost generování různého počtu datových sad - je tedy možné si zvolit, kolik těchto přídavných dat je požadováno. Nicméně i to má svá omezení a je nutné počítat s náročností dat z hlediska úložiště. Níže uvedené výsledky 5.5 a 5.6 vycházejí z *počtu = 3*:

Tabulka 5.5: Výsledky testování MSR v procentech, s přídavkem generovaných dat

	Sada AS1	**Sada AS2**	**Sada AS3**	**Všechny akce AS4**
Experiment A	92,1	85,8	94,8	85,9
Experiment B	92,7	86,3	95,3	88,7
Experiment C	83,0	76,8	89,2	76,3

Tabulka 5.6: Výsledky testování KARD v procentech, s přídavkem generovaných dat

	Sada AS1	**Sada AS2**	**Sada AS3**	**Všechny akce AS4**
Experiment A	99,9	99,9	96,5	98,5
Experiment B	100,0	100,0	99,0	98,7
Experiment C	98,5	99,8	87,4	92,5

Zde lze pozorovat, že v mnohých případech je generování dat prospěšné - nicméně se velmi úzce odvíjí od faktu, že pokud originální data ve svém objemu obsahují, ať už úmyslně či ne, chybová data, i vygenerovaná data se od nich odvíjejí. Zlepšení je možné pozorovat zejména u datasetu MSR, kde tato generace byla prospěšná a celkovou přesnost zlepšila. Stále však jsou v průměru tyto zlepšení poměrně minimální. Je však ale hodno vyzvednout fakt, že tyto modely, jež byly trénovány tedy i na vygenerovaných datech, by teoreticky měly být zlepšeny i z hlediska všestrannosti - jsou tedy obecnější a jejich uplatnění by mělo být širší. Konfuzní matice všech experimentů jsou zahrnuty v příloze C.

Kapitola 6

Závěr

Cílem této bakalářské práce bylo zejména seznámit se s neuronovými sítěmi, jejich principiální funkcionalitou a uplatněním - a to zejména v oblasti rozpoznávání lidských činností. V první části práce jsou popsány základní stavební kameny neuronových sítí, jejich struktura a také jejich předobraz v přírodě, na jehož principu vzdáleně fungují. Důraz je především kladen na CNN, které jsou popsány podrobněji, včetně jejich unikátních vrstev.

V následující kapitole byly detailně popsány reziduální modely, jejich přínos a výhody oproti standardním konvolučním sítím. Konkrétně pak bylo zaměřeno na model ResNet50. Taktéž zde byly vysvětleny některé přídavné vrstvy a funkce, jež byly využity v implementační části. V neposlední řadě byl popsán i princip přenosového učení. V další kapitole byl popsán způsob implementace neuronové sítě a byl taktéž kladen důraz na popis předpracování dat. Pro účely této práce byla zvolena síť ResNet50, především díky své popularitě z hlediska strojového učení a její spolehlivosti. Tato síť byla následně použita na trénování jak na datasetu MSR, tak datasetu KARD. Na těchto datasetech byla také vyzkoušena metoda generování dat, která měla sloužit jako přínos pro natrénované modely ve směru zobecnění.

V závěru práce bylo poté patrné, že výsledky trénování byly uspokojivé a vygenerovaná data se většinově projevila jako přínosná. Z těchto poznatků je však jisté, že do budoucí práce je nutné využít i dalších, přesnějších metod extrakce důležitých prvků z dat, jako jsou například hloubkové mapy. Také je nutné vzít jako klíčový aspekt k vylepšení i samotné generování dat, jež by mohlo být implementováno dynamicky za chodu sítě a ušetřilo se takto prostředků nutných k provozu těchto modelů.

Přínosem pro mne ze shrnutí této práce je hlavně uvedení do problematiky neuronových sítí, jejich pochopení, implementace a potenciální využití. Dále jsem se díky této práci mohl naučit i pokročilejším frameworkům, zejména tedy Keras a Tensorflow. Taktéž mne to přiblížilo ke zdokonalení z hlediska psaní kódu v Pythonu a používání cizích modulů a knihoven.

Literatura

1. EDUCATION, IBM Cloud. *What are neural networks?* [online]. 2020. [cit. 2022-04-21]. Dostupné z: `https://www.ibm.com/cloud/learn/neural-networks`.
2. RASHID, Tariq. *Make Your Own Neural Network - A Gentle Journey through the Mathematics of Neural Networks*. 1st ed. CreativeSpace Independent Publishing Platform, 2016. ISBN 978-1530826605.
3. MHATRE, Saurabh. *What Is The Relation Between Artificial And Biological Neuron?* [online]. 2020. [cit. 2022-04-20]. Dostupné z: `https://smhatre59.medium.com/what-is-the-relation-between-artificial-and-biological-neuron-18b05831036`.
4. BEAR, Mark F.; CONNORS, Barry W.; PARADISO, Michael A. *Neuroscience Exploring the Brain*. 4th ed. Ed. by JOYCE, Jonathan; FRANCIS, Linda G.; LOCHHAAS, Tom. Wolters Kluwer, 2019. ISBN 978-0-7817-7817-6.
5. DI, Wei; BHARDWAJ, Anurag; WEI, Jianing. *Deep Learning Essentials - Your hands-on guide to the fundamentals of deep learning and neural network modeling*. 1st ed. Ed. by PAGARE, Veena; SINGH, Aman; KOLTE, Snehal; NIKALJE, Sayli. Birmingham - Mumbai: Packt Publishing Ltd., 2018. ISBN 978-1-78588-036-0.
6. WIKIPEDIA, THE FREE ENCYCLOPEDIA. *Artificial neuron model* [online]. 2005. [cit. 2022-04-14]. Dostupné z: `https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png`.
7. SALIAN, Isha. *SuperVize Me: What's the Difference Between Supervised, Unsupervised, Semi-Supervised and Reinforcement Learning?* [online]. 2018. [cit. 2022-04-10]. Dostupné z: `https://blogs.nvidia.com/blog/2018/08/02/supervised-unsupervised-learning/`.
8. DEEPAI: THE FRONT PAGE OF A.I. *Artificial neuron model* [online]. 2019. [cit. 2022-04-17]. Dostupné z: `https://deepai.org/machine-learning-glossary-and-terms/perceptron`.
9. MUELLER, John Paul; MASSARON, Luca. *Deep Learning For Dummies*. 1st ed. Ed. by ALI, Mohammed Zafar. Hoboken, New Jersey: John Wiley and Sons, Inc, 2019. ISBN 978-1-119-54304-6.

10. SINGH, Himanshi. *Deep Learning 101: Beginners Guide to Neural Network* [online]. 2021. [cit. 2022-04-22]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/03/basics-of-neural-network/.

11. SHRIVASTAV, Aman. *Gaussian Error Linear Unit (GELU)* [online]. 2021. [cit. 2022-04-25]. Dostupné z: https://iq.opengenus.org/gaussian-error-linear-unit/.

12. MCGONAGLE, John; SHAIKOUSI, George; WILLIAMS, Christopher; HSU, Andrew; KHIM, Jimin; MILLER, Aaron. *Backpropagation* [online]. 2018. [cit. 2022-04-08]. Dostupné z: https://brilliant.org/wiki/backpropagation/.

13. EDUCATION, IBM Cloud. *Overfitting - What is overffiting?* [online]. 2021. [cit. 2022-04-11]. Dostupné z: https://www.ibm.com/cloud/learn/overfitting.

14. CORNELISSE, Daphne. *An intuitive guide to Convolutional Neural Networks* [online]. 2018. [cit. 2022-04-20]. Dostupné z: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/.

15. PINGEL, Johanna; PATEL, Shyamal. *Introduction to Deep Learning: What Are Convolutional Neural Networks?* [online]. 2017. [cit. 2022-04-08]. Dostupné z: https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html.

16. DERDAT, Arden. *Applied Deep Learning - Part 4: Convolutional Neural Networks* [online]. 2017. [cit. 2022-04-11]. Dostupné z: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

17. KARPATHY, Andrej. *Convolutional Neural Networks (CNNs / ConvNets)* [online]. [cit. 2022-04-08]. Dostupné z: https://cs231n.github.io/convolutional-networks/.

18. SAXENA, Pawan. *Residual Networks (ResNet) – Deep Learning* [online]. 2020. [cit. 2022-04-15]. Dostupné z: https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/.

19. KAUSHIK, Aakash. *Understanding ResNet50 architecture* [online]. 2020. [cit. 2022-04-23]. Dostupné z: https://iq.opengenus.org/resnet50-architecture/.

20. KAIMING, He; XIANGYU, Zhang; SHAOQING, Ren; JIAN, Sun. *Deep Residual Learning for Image Recognition* [online]. 2015. [cit. 2022-04-17]. Dostupné z: https://arxiv.org/abs/1512.03385.

21. SHARMA, Pranshu. *Understanding Transfer Learning for Deep Learning* [online]. 2021. [cit. 2022-04-22]. Dostupné z: https://www.analyticsvidhya.com/blog/2021/10/understanding-transfer-learning-for-deep-learning/#h2_6.

22. CHOLLET, François. *Keras* [online]. 2015. [cit. 2022-04-21]. Dostupné z: https://keras.io/.

23. ABADI, Martín. *TensorFlow: A System for Large-Scale Machine Learning* [online]. 2015. [cit. 2022-04-21]. Dostupné z: https://www.tensorflow.org/.

24. OLIPHANT, Travis. *NumPy* [online]. 2006. [cit. 2022-04-21]. Dostupné z: https://numpy.org.

25. BRADSKI, G. *The OpenCV Library* [online]. 2000. [cit. 2022-04-12]. Dostupné z: https://opencv.org/.

26. LI, Wanqing; ZHANG, Zhengyou; LIU, Zicheng. *Action recognition based on a bag of 3D points* [online]. IEEE Computer Society Conference on Computer Vision a Pattern Recognition - Workshops, 2010 [cit. 2022-03-28]. ISBN 978-1-4244-7030-3. Dostupné z DOI: 10.1109/CVPRW.2010.5543273.

27. GAGLIO, S.; RE, G. Lo; MORANA, M. *Human Activity Recognition Process Using 3-D Posture Data* [online]. EEE Transactions on Human-Machine Systems, 2014 [cit. 2022-03-25]. ISSN 2168-2305. Dostupné z DOI: 10.1109/THMS.2014.2377111.

28. NAUTIYAL, Dewang. *ML | Underfitting and Overfitting* [online]. 2021. [cit. 2022-04-25]. Dostupné z: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/.

29. ADALOGLOU, Nikolas. Intuitive Explanation of Skip Connections in Deep Learning. *https://theaisummer.com* [online]. 2020 [cit. 2022-04-28]. Dostupné z: https://theaisummer.com/skip-connections/.

Příloha A

Dokumentace kódu

Projekt je rozdělen v rámci kořenového adresáře na sérii skriptů, jež se pouštějí v pořadí samostatně dle jejich účelu. Tyto skripty jsou:

- `dataset_configuration.py` je skript, jež zpracovává vstupní dataset a převádí jej do předzpracované podoby, generuje složky do přesně dané hierarchie a do této hierarchie dataset třídí dle podmnožin (subsetů), experimentů a akcí.
- `histograms.py` generuje histogramy a optimální hodnoty pro interpolaci dat .
- `servitor.py` provádí interpolaci dat na požadovaný rozměr, upravuje označení akcí tak, aby byla budoucí práce s nimi snažší.
- `network_training.py` je skript, jež spouští trénování sítě.

Zmíněné skripty by měly být spuštěny v pořadí, jak byly výše uvedeny - tedy je nutné spustit`dataset_configuration.py` jako první a `network_training.py` jako poslední. Ovládání těchto skriptů v rámci volitelných parametrů probíhá skrze konfigurační soubor `config.yaml`, v němž jsou klíčové parametry pro všechny skripty - detailní popis je uveden níže v příloze. Dále se zde nachází soubor `runtime_config.yaml`, jež plní úlohy dynamického konfigurátoru a jsou do něj ukládány výsledky operací některých skriptů - například absolutní cesty podle umístění v rámci skriptu `dataset_computation.py`. Je tedy patrné, že bez této dvojice konfiguračních souborů nelze skripty spouštět.

Dále jsou součástí adresáře 2 skripty, které plní funkcí modulů a jsou využity pouze interně - nejsou tedy samostatně spustitelné:

- `resnet_implementation.py` je implementace sítě ResNet50, jež byla převzata z frameworku Keras a převedena do lokálního zdrojového kódu.
- `imagenet_utils.py` je doprovodným skriptem pro `resnet_implementation.py` - obsahuje kořenovou funkcionalitu pro práci s daty.

V kořenovém adresáři se dále nacházejí složky `KARD_actions` a `MSR_actions`, jež obsahují surová, nezpracovaná data datasetů v textovém formátu. Po spuštění skriptu `dataset_computation.py` jsou vytvořeny dále složky `KARD_coords` a `MSR_coords`, jež obsahují pouze .txt soubory se souřadnicovými daty, a `KARD_subsets` a `MSR_subsets`, v kterých jsou obsaženy již roztříděná data do podmnožin, experimentů a akcí.

Dále je v adresáři obsažen soubor `JiangExperimentFileList.txt`, jež je využit pro přetřídění dat MSR, jak je zmíněno v práci - strukturálně se skládá z validních záznamů, jež v trénování a testování jsou zahrnuty. Nevhodná data jsou tedy automaticky vyloučena. Po natrénování sítě jsou výsledky z trénování tříděny do vygenerovaných složek `Results_název_datasetu` - název datasetu se odvíjí od aktivního datasetu specifikovanám v `config.yaml`.

Soubor **`config.yaml`** obsahuje sérii parametrů, které jsou uživatelem ručně nastavovány. Tyto parametry jsou:

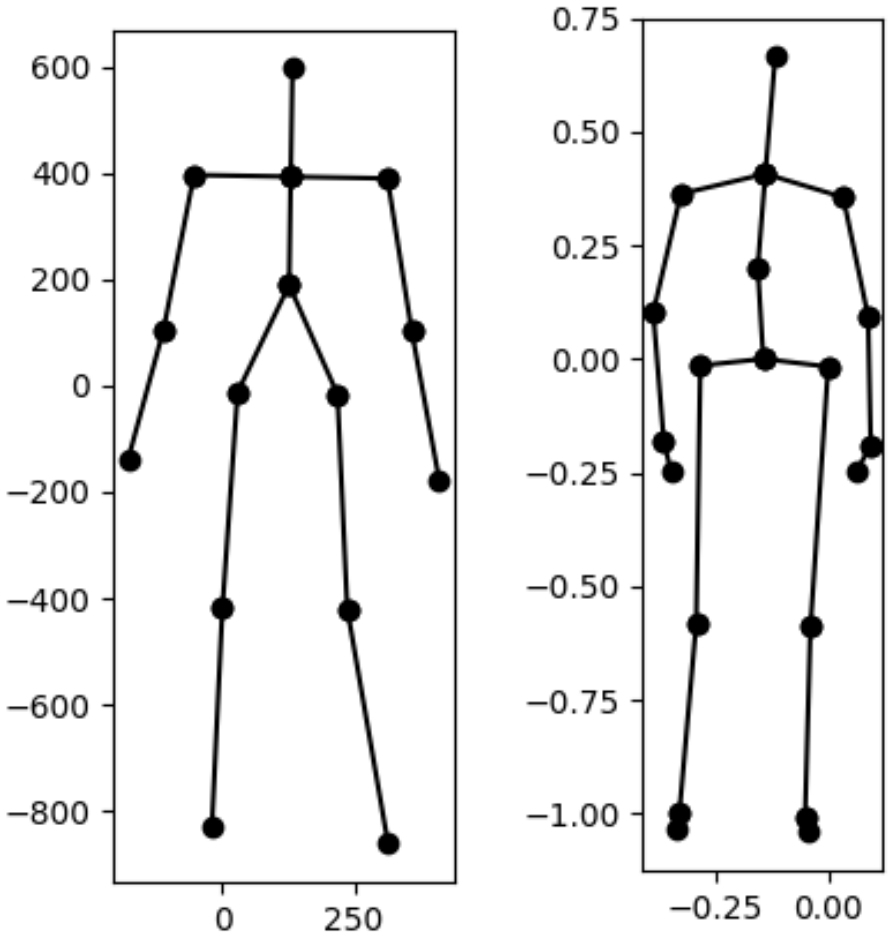
- **active_ds** - nastavení aktivního datasetu
- **frame_count** - počet snímků pro dataset
- **generate** - indikátor, zda se mají generovat data pro trénovací set
- **joint_count** - počet kloubů datasetu
- **generation_count** - počet vygenerovaných sérií přídavných dat
- **end** - stropní hodnota rozsahu pro generaci šumu v milimetrech
- **start** - počáteční hodnota rozsahu pro generaci šumu v milimetrech

Poté se zde nacházejí jednotlivé názvy akcí obou datasetů i s příslušnými označeními - tyto parametry upravovány nikdy nejsou. Soubor **`runtime_config.yaml`** obsahuje několik dynamických parametrů - tj. tyto parametry jsou nastaveny při spuštění některých ze skriptů. Tyto parametry jsou:

- **root** - absolutní adresa kořenového adresáře projektu
- **hodnoty pro rozměry x, y a z** každého z datasetů - tyto hodnoty uchovávají informace pro interpolaci dat

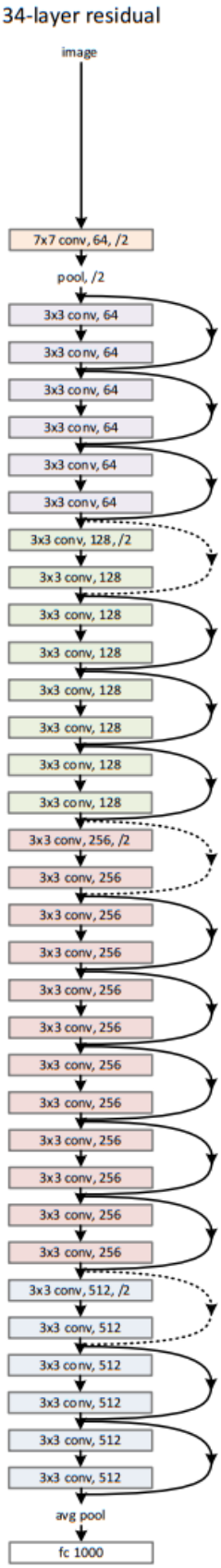
Příloha B

Doprovodné ilustrace ve větším rozlišení



(a) Kostra člověka datasetu KARD. (b) Kostra člověka datasetu MSR.

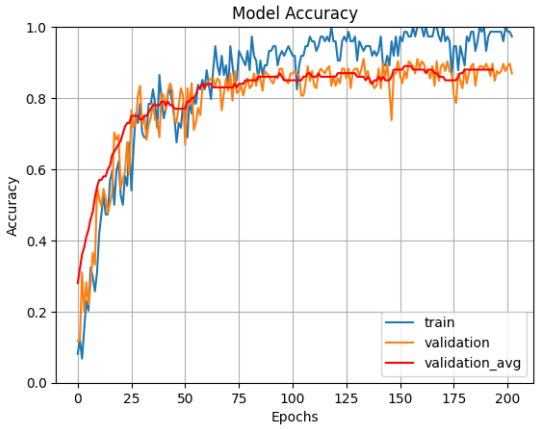
Obrázek B.1: Kostry datasetů MSR a KARD



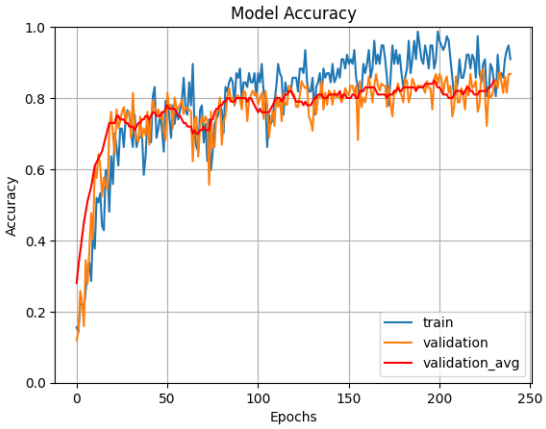
Obrázek B.2: Přehlednější rozlišení architektury ResNet34 [3]

Příloha C

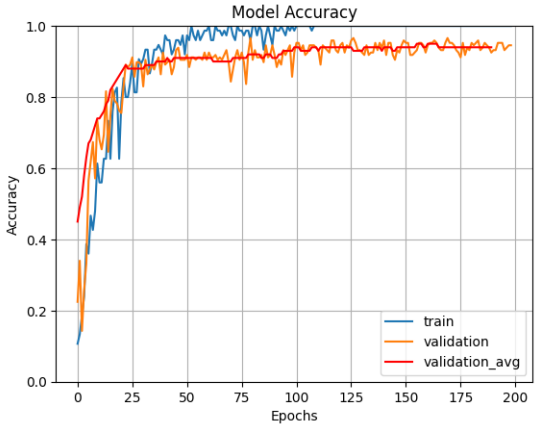
Ztrátové funkce, konfuzní matice, přesnosti



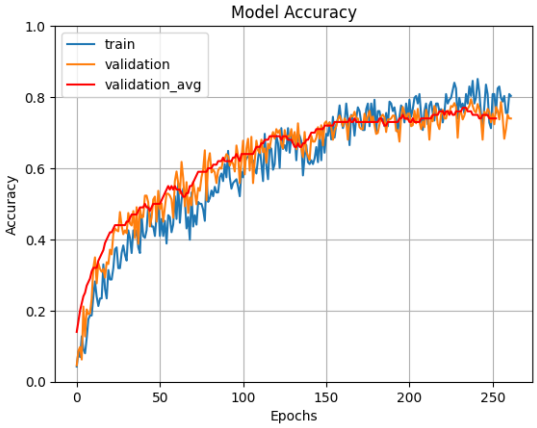
(a) Sada akcí 1 - Experiment A - Přesnost



(b) Sada akcí 2 - Experiment A - Přesnost

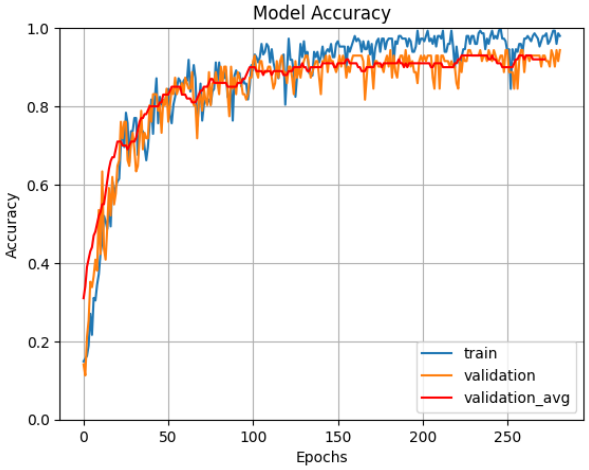


(c) Sada akcí 3 - Experiment A - Přesnost

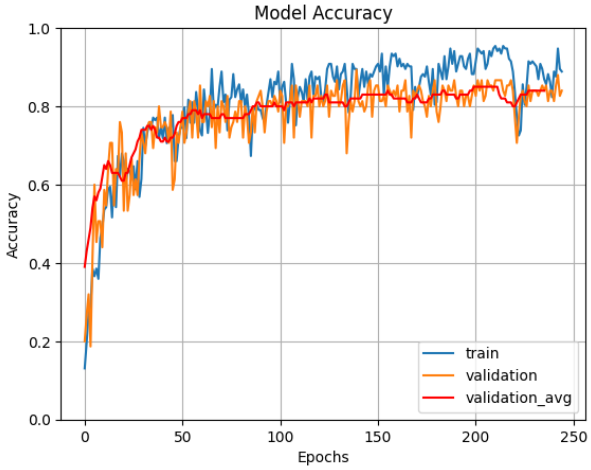


(d) Sada akcí 4 - Experiment A - Přesnost

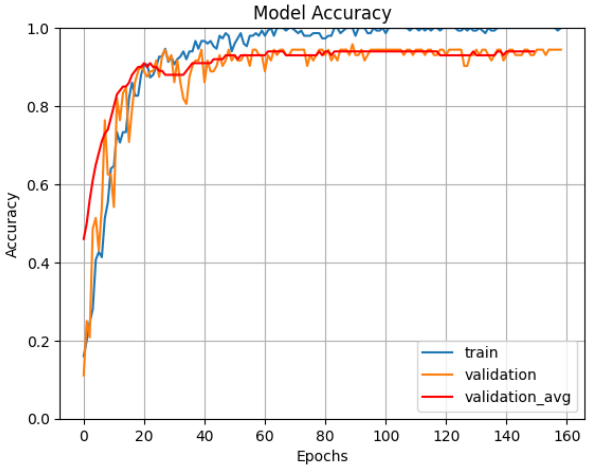
Obrázek C.1: Grafy přesností všech akčních sad Experimentu A datasetu **MSR**



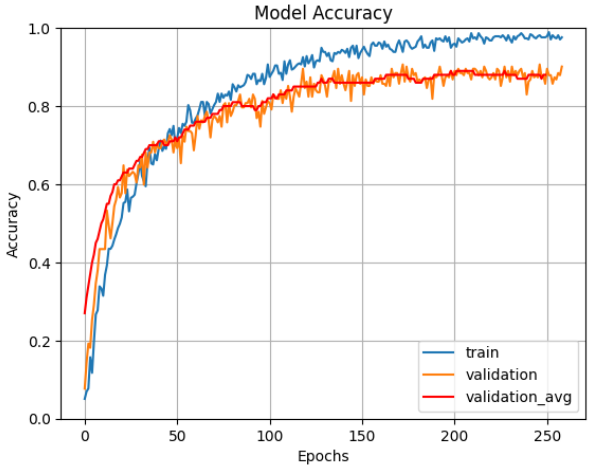
(a) Sada akcí 1 - Experiment B - Přesnost



(b) Sada akcí 2 - Experiment B - Přesnost

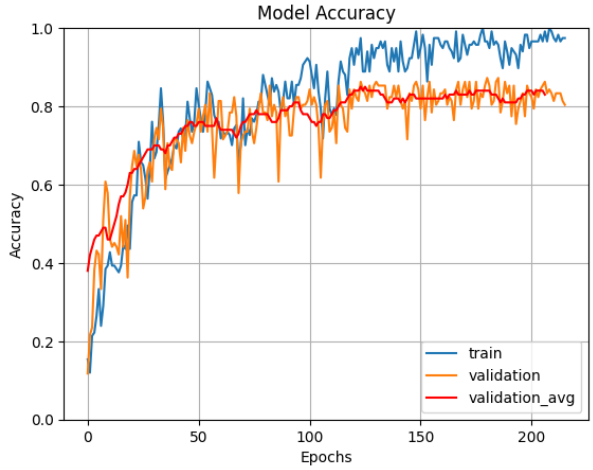


(c) Sada akcí 3 - Experiment B - Přesnost

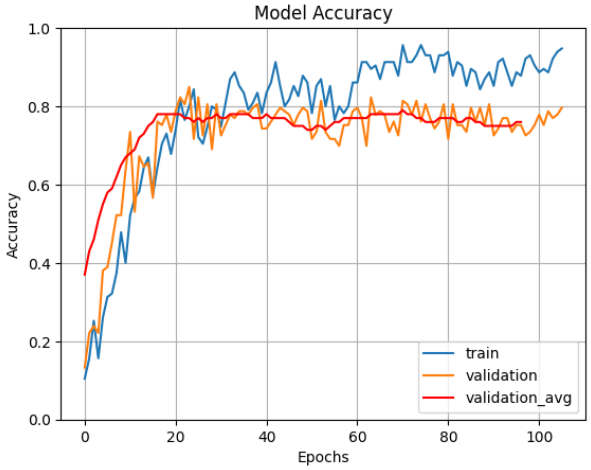


(d) Sada akcí 4 - Experiment B - Přesnost

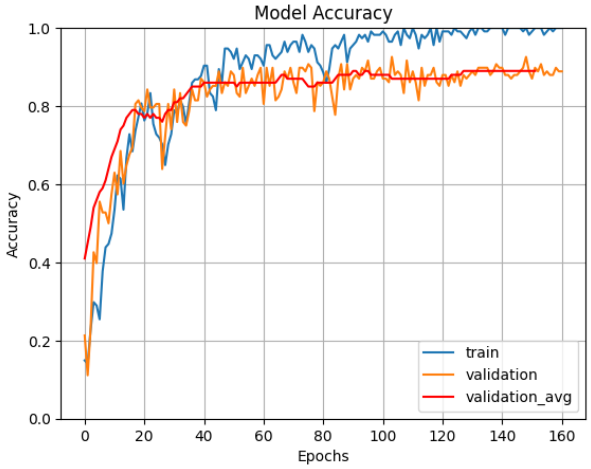
Obrázek C.2: Grafy přesností všech akčních sad Experimentu B datasetu **MSR**



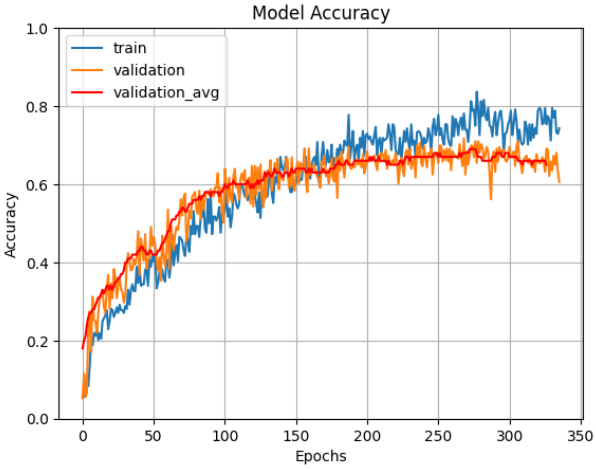
(a) Sada akcí 1 - Experiment C - Přesnost



(b) Sada akcí 2 - Experiment C - Přesnost

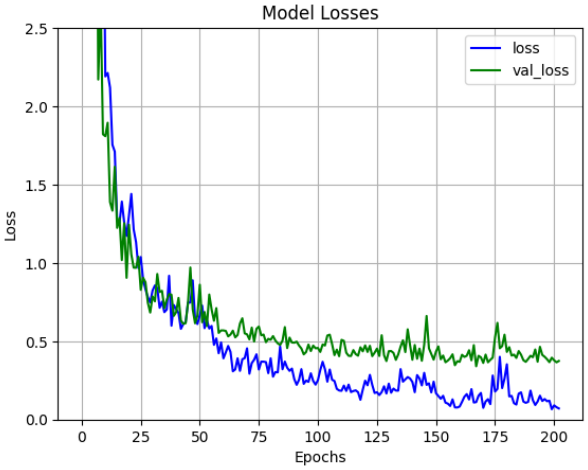


(c) Sada akcí 3 - Experiment C - Přesnost

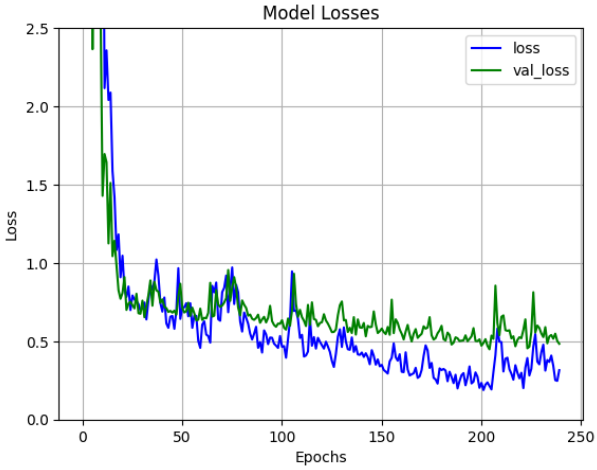


(d) Sada akcí 4 - Experiment C - Přesnost

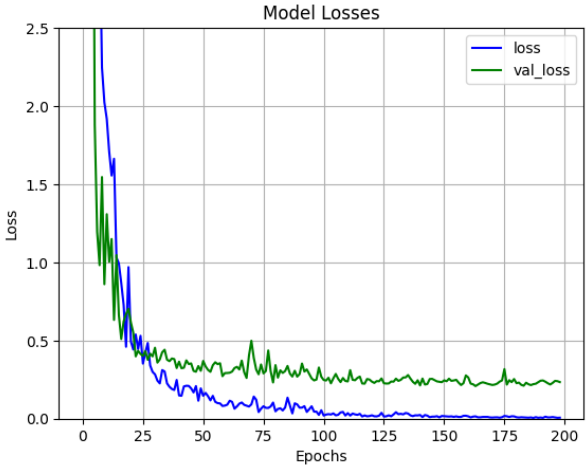
Obrázek C.3: Grafy přesností všech akčních sad Experimentu C datasetu **MSR**



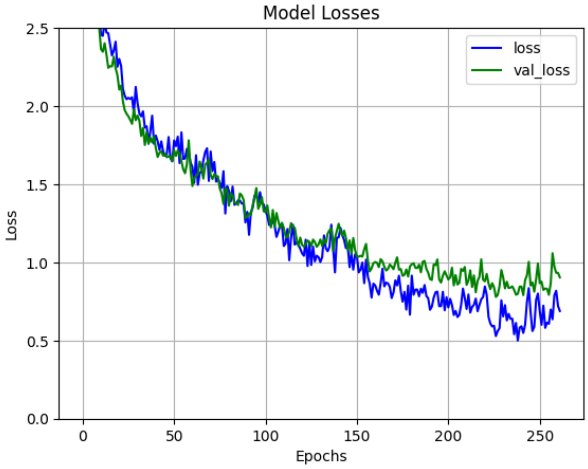
(a) Sada akcí 1 - Experiment A - Ztrátová funkce



(b) Sada akcí 2 - Experiment A - Ztrátová funkce

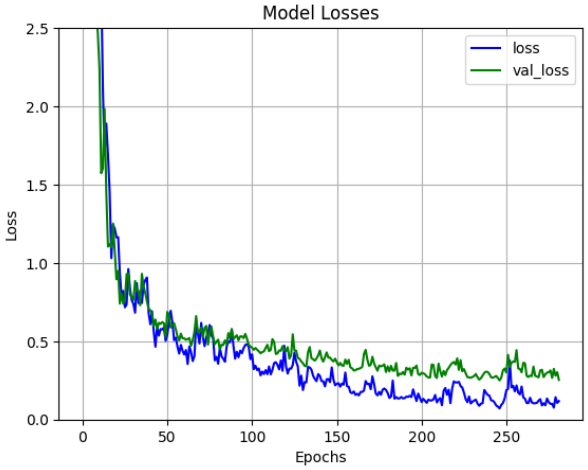


(c) Sada akcí 3 - Experiment A - Ztrátová funkce

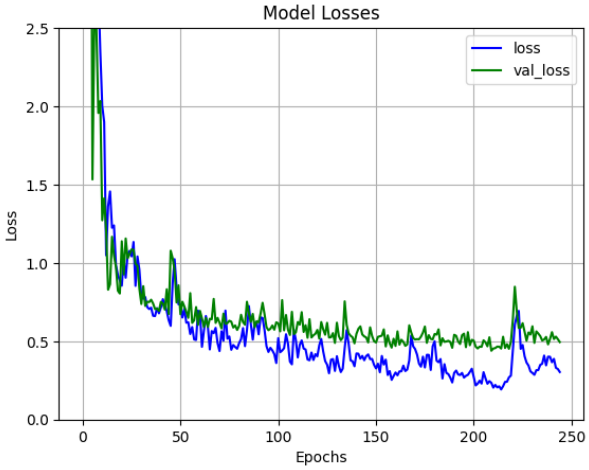


(d) Sada akcí 4 - Experiment A - Ztrátová funkce

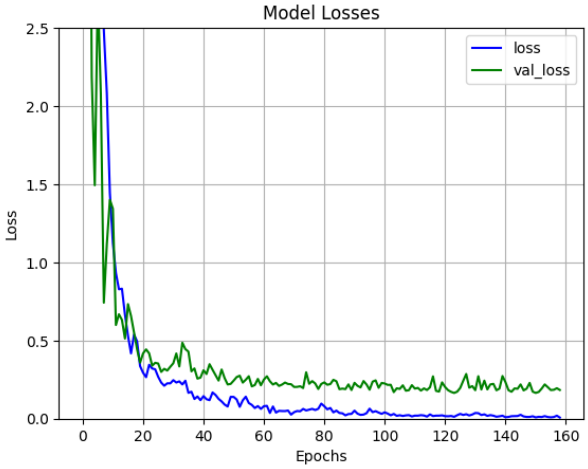
Obrázek C.4: Grafy ztrátových funkcí všech akčních sad Experimentu A datasetu **MSR**



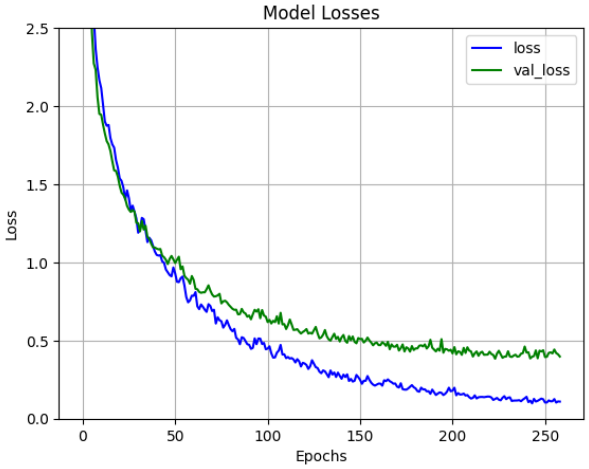
(a) Sada akcí 1 - Experiment B - Ztrátová funkce



(b) Sada akcí 2 - Experiment B - Ztrátová funkce

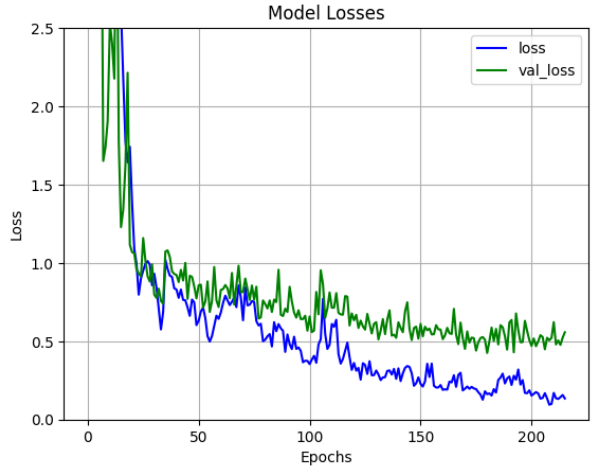


(c) Sada akcí 3 - Experiment B - Ztrátová funkce

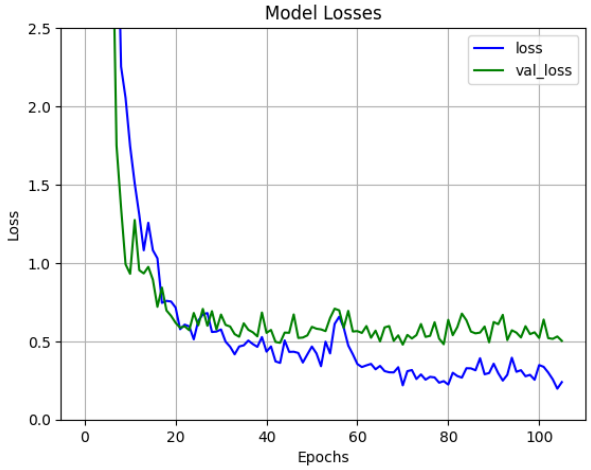


(d) Sada akcí 4 - Experiment B - Ztrátová funkce

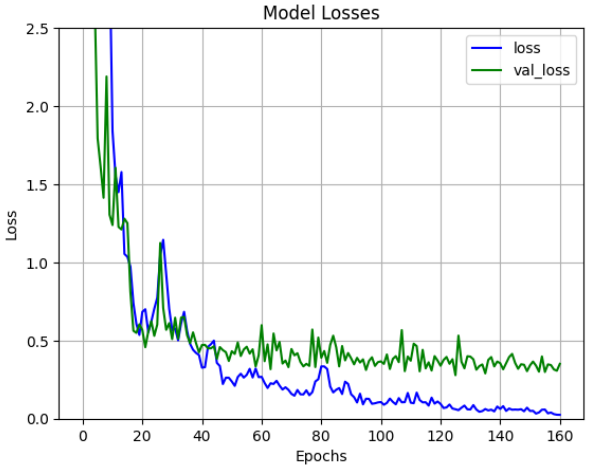
Obrázek C.5: Grafy ztrátových funkcí všech akčních sad Experimentu B datasetu **MSR**



(a) Sada akcí 1 - Experiment C - Ztrátová funkce



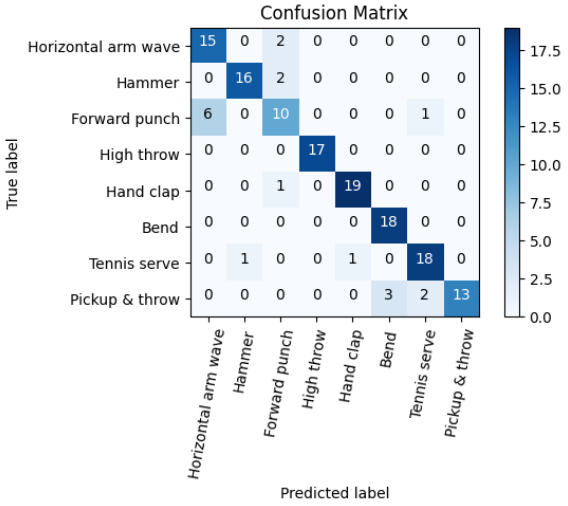
(b) Sada akcí 2 - Experiment C - Ztrátová funkce



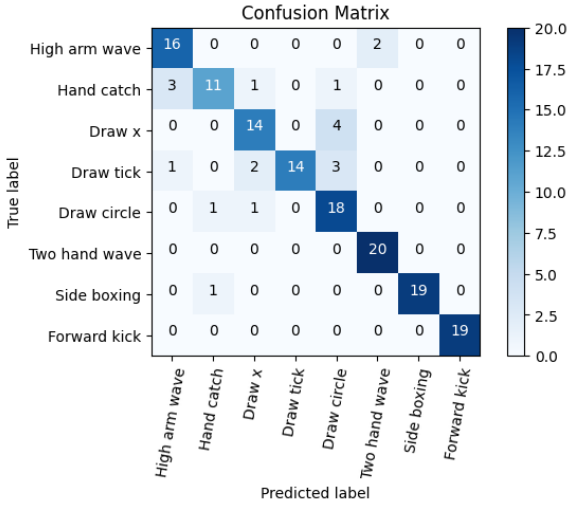
(c) Sada akcí 3 - Experiment C - Ztrátová funkce

(d) Sada akcí 4 - Experiment C - Ztrátová funkce

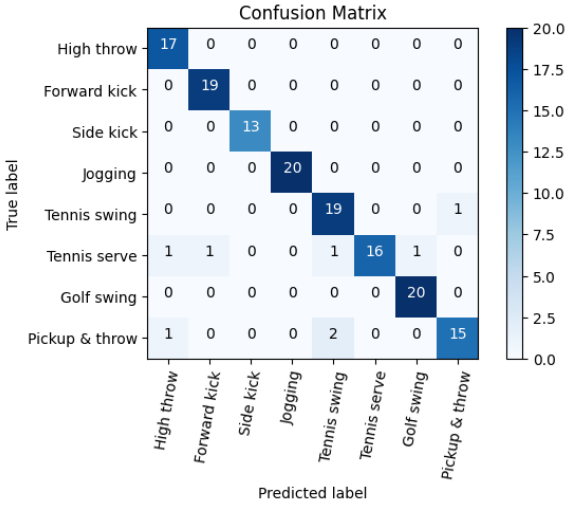
Obrázek C.6: Grafy ztrátových funkcí všech akčních sad Experimentu C datasetu **MSR**



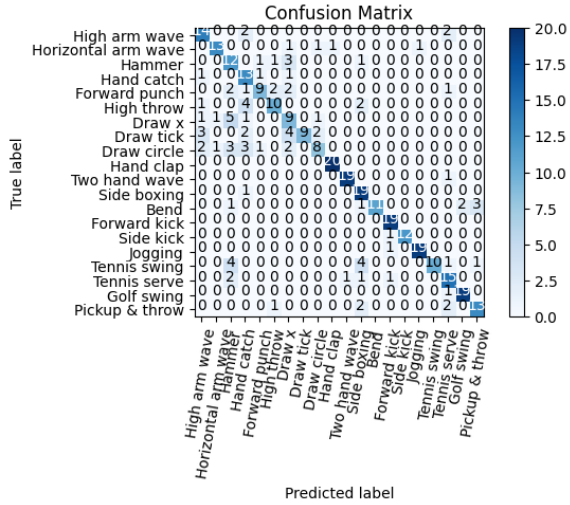
(a) Sada akcí 1 - Experiment A - Konfuzní mapa



(b) Sada akcí 2 - Experiment A - Konfuzní mapa

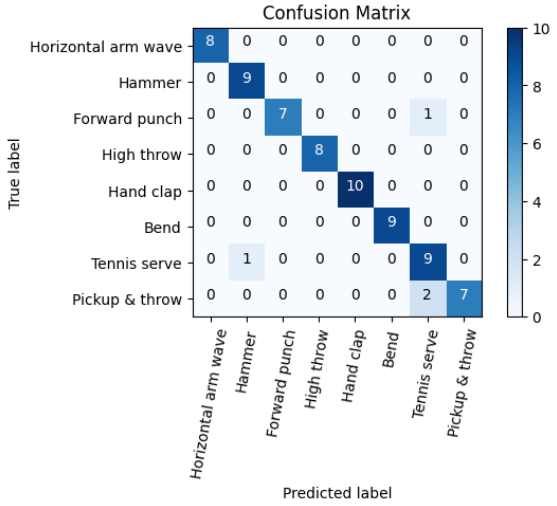


(c) Sada akcí 3 - Experiment A - Konfuzní mapa

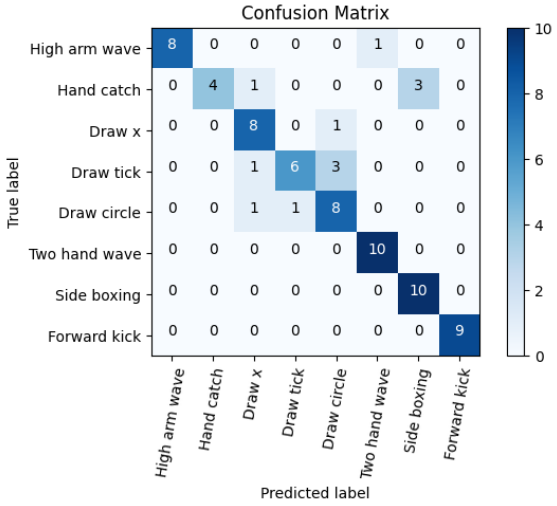


(d) Sada akcí 4 - Experiment A - Konfuzní mapa

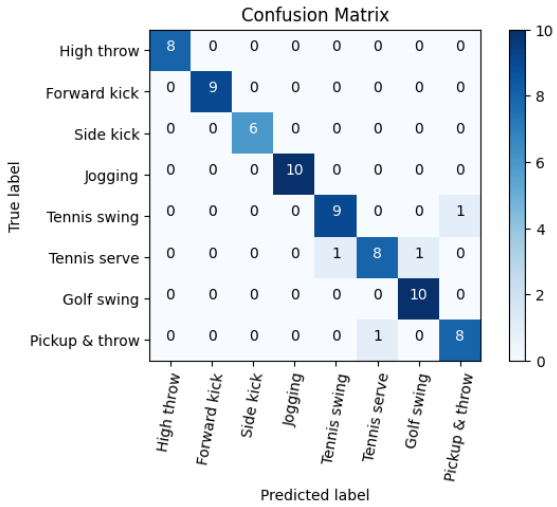
Obrázek C.7: Konfuzní mapy všech akčních sad Experimentu A datasetu **MSR**



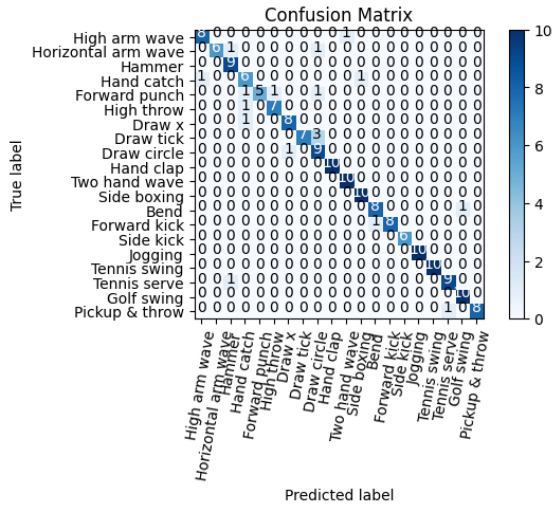
(a) Sada akcí 1 - Experiment B - Konfuzní mapa



(b) Sada akcí 2 - Experiment B - Konfuzní mapa

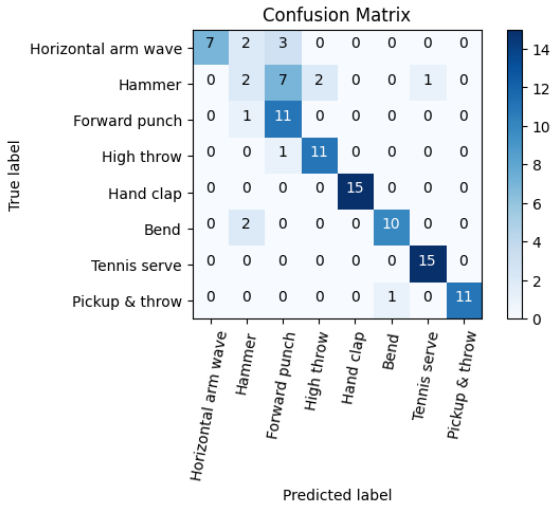


(c) Sada akcí 3 - Experiment B - Konfuzní mapa

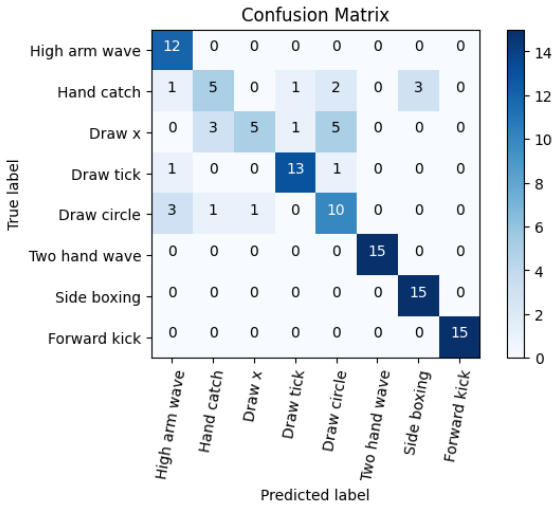


(d) Sada akcí 4 - Experiment B - Konfuzní mapa

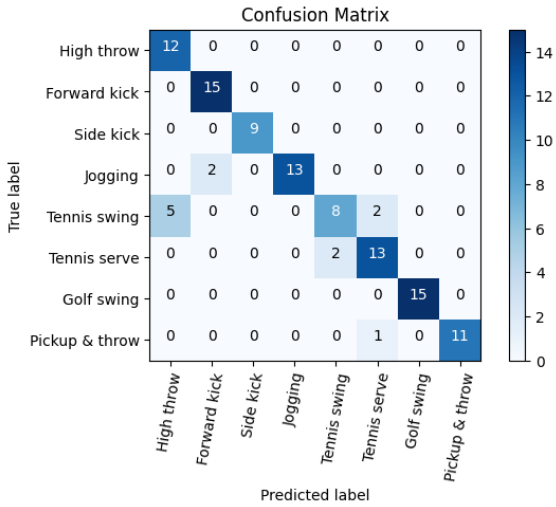
Obrázek C.8: Konfuzní mapy všech akčních sad Experimentu B datasetu **MSR**



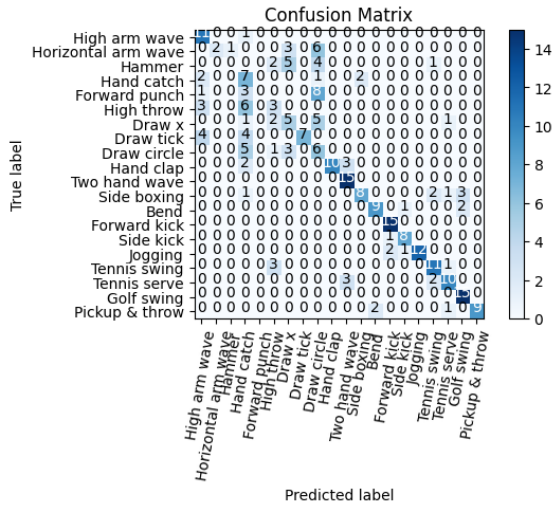
(a) Sada akcí 1 - Experiment C - Konfuzní mapa



(b) Sada akcí 2 - Experiment C - Konfuzní mapa

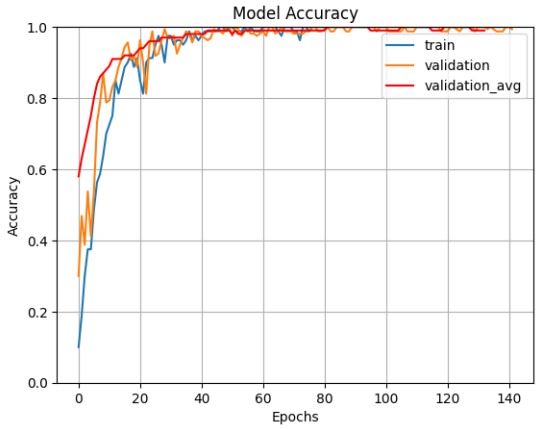


(c) Sada akcí 3 - Experiment C - Konfuzní mapa

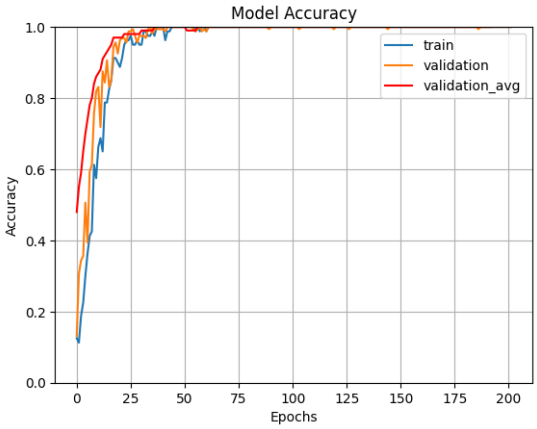


(d) Sada akcí 4 - Experiment C - Konfuzní mapa

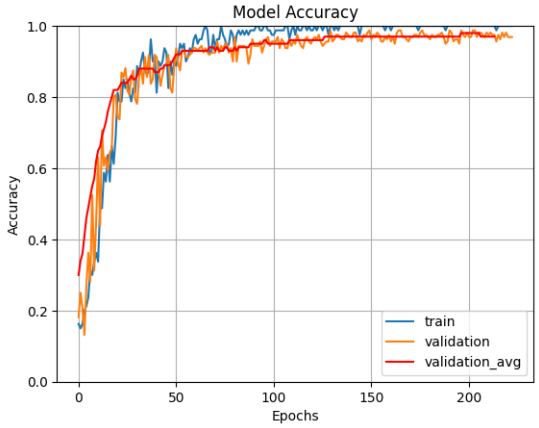
Obrázek C.9: Konfuzní mapy všech akčních sad Experimentu C datasetu **MSR**



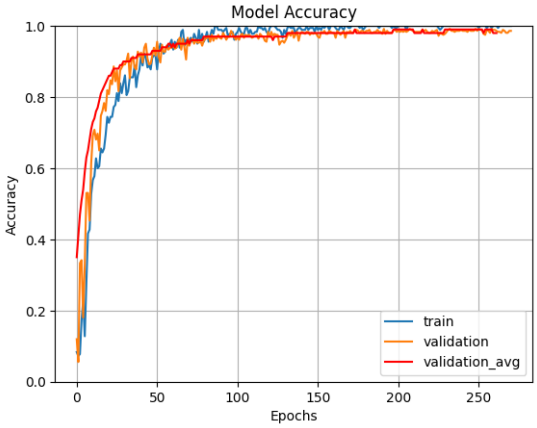
(a) Sada akcí 1 - Experiment A - Přesnost



(b) Sada akcí 2 - Experiment A - Přesnost

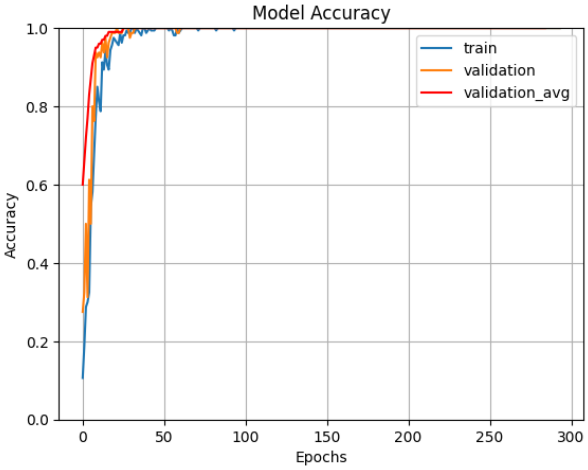


(c) Sada akcí 3 - Experiment A - Přesnost

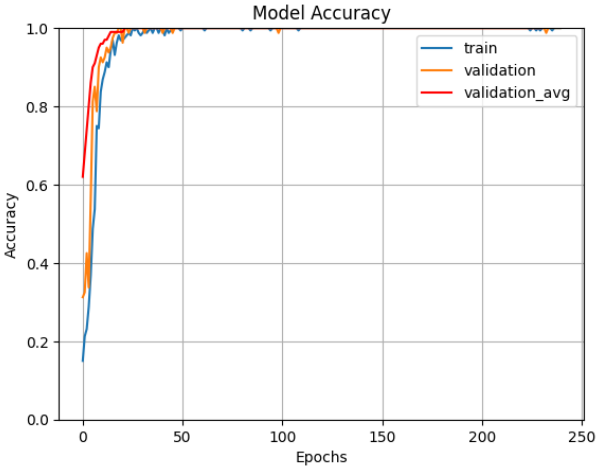


(d) Sada akcí 4 - Experiment A - Přesnost

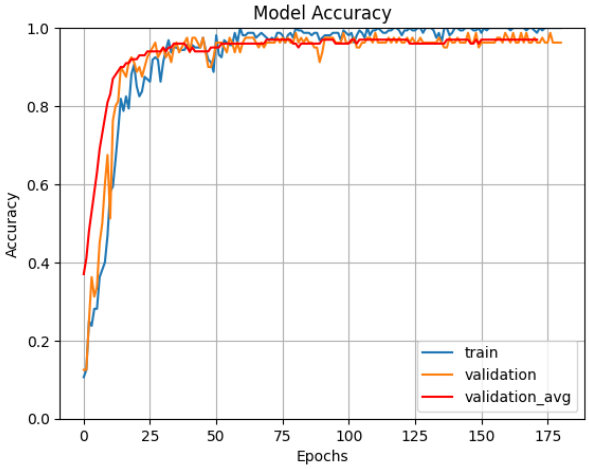
Obrázek C.10: Grafy přesností všech akčních sad Experimentu A datasetu **KARD**



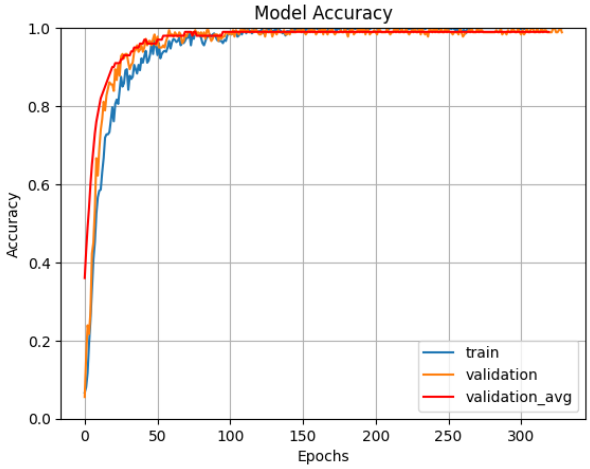
(a) Sada akcí 1 - Experiment B - Přesnost



(b) Sada akcí 2 - Experiment B - Přesnost

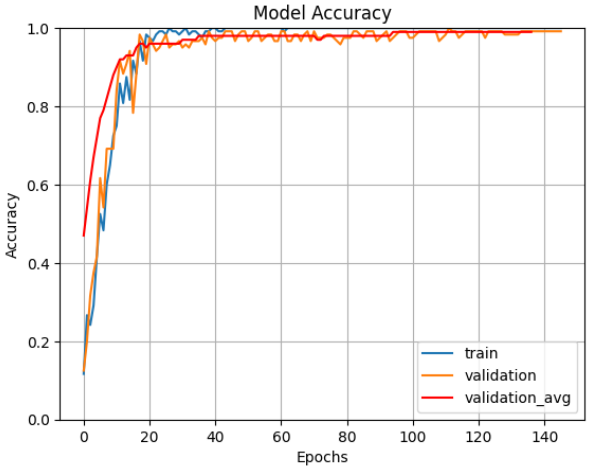


(c) Sada akcí 3 - Experiment B - Přesnost

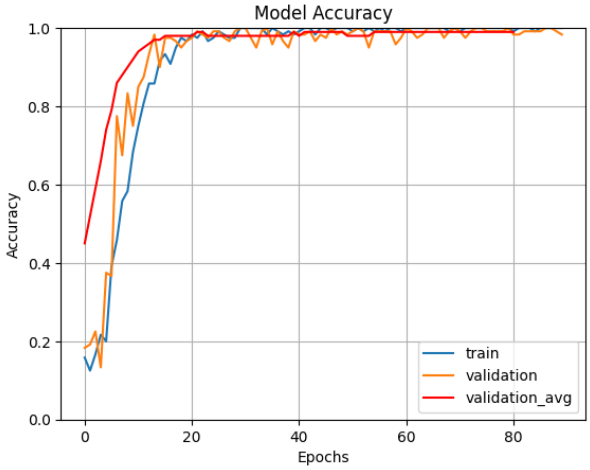


(d) Sada akcí 4 - Experiment B - Přesnost

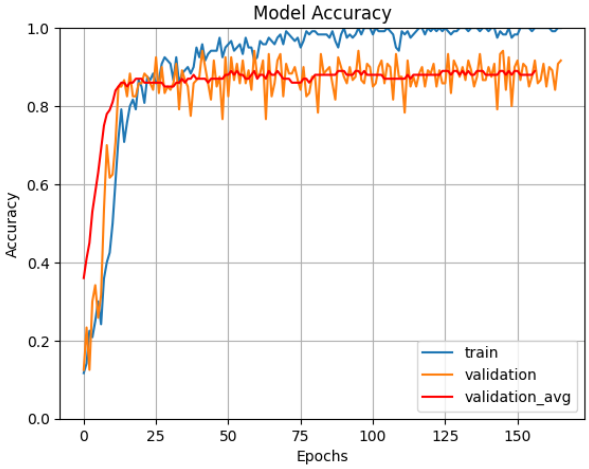
Obrázek C.11: Grafy přesností všech akčních sad Experimentu B datasetu **KARD**



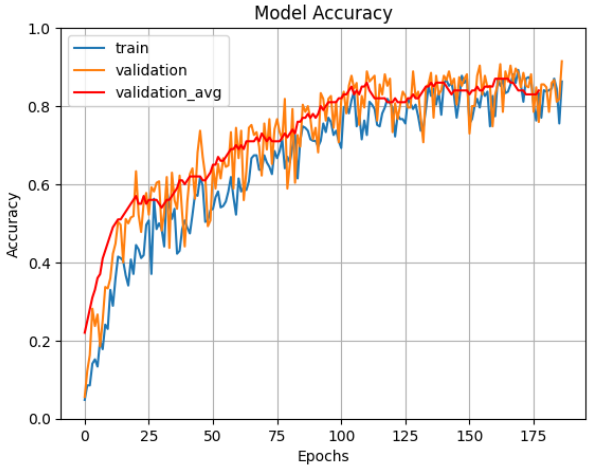
(a) Sada akcí 1 - Experiment C - Přesnost



(b) Sada akcí 2 - Experiment C - Přesnost

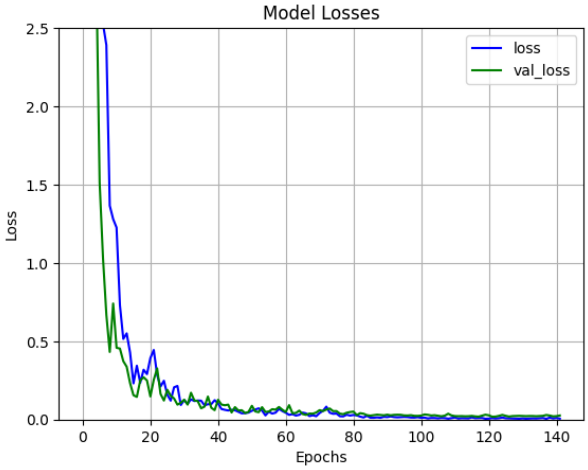


(c) Sada akcí 3 - Experiment C - Přesnost

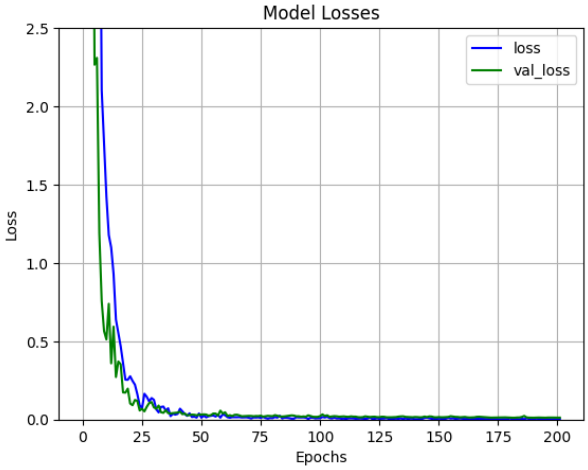


(d) Sada akcí 4 - Experiment C - Přesnost

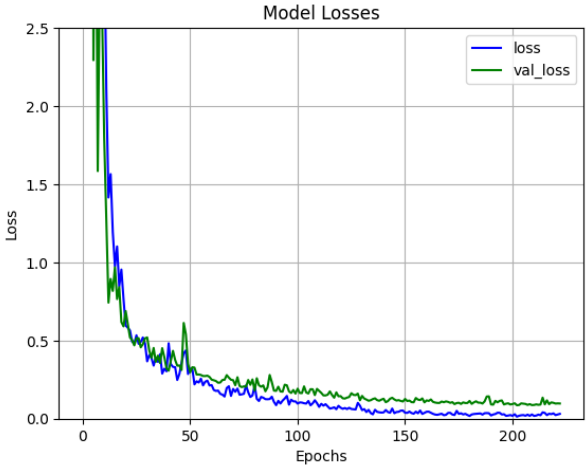
Obrázek C.12: Grafy přesností všech akčních sad Experimentu C datasetu **KARD**



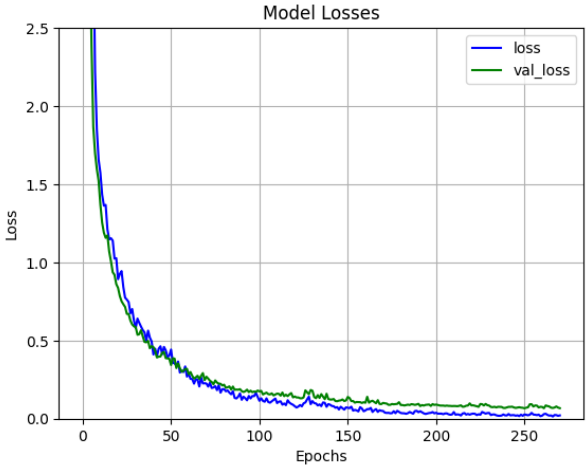
(a) Sada akcí 1 - Experiment A - Ztrátová funkce



(b) Sada akcí 2 - Experiment A - Ztrátová funkce

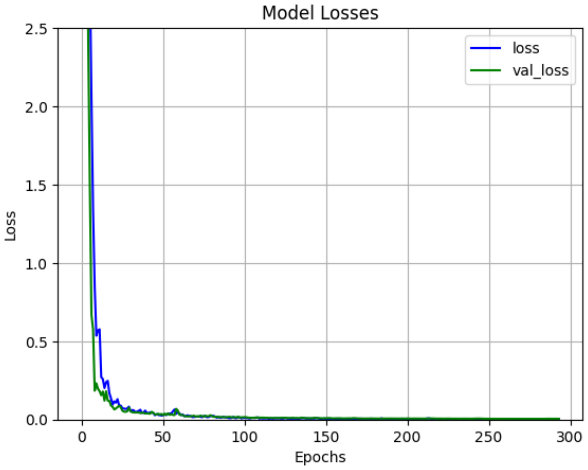


(c) Sada akcí 3 - Experiment A - Ztrátová funkce

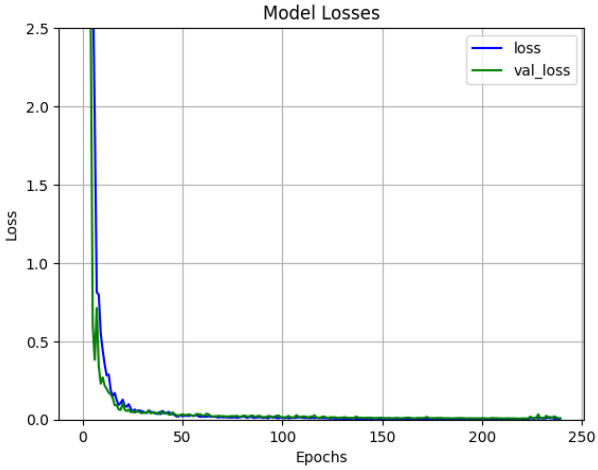


(d) Sada akcí 4 - Experiment A - Ztrátová funkce

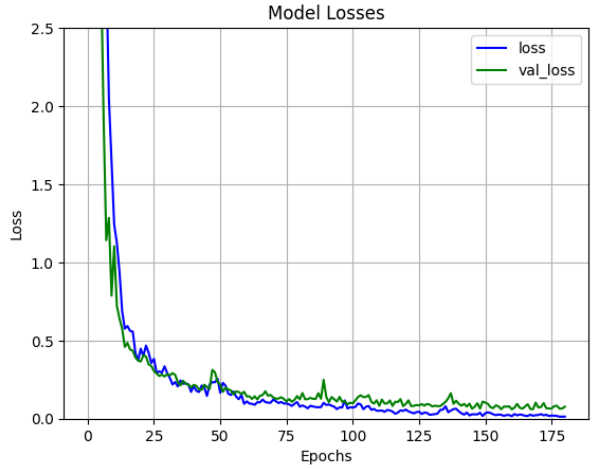
Obrázek C.13: Grafy ztrátových funkcí všech akčních sad Experimentu A datasetu **KARD**



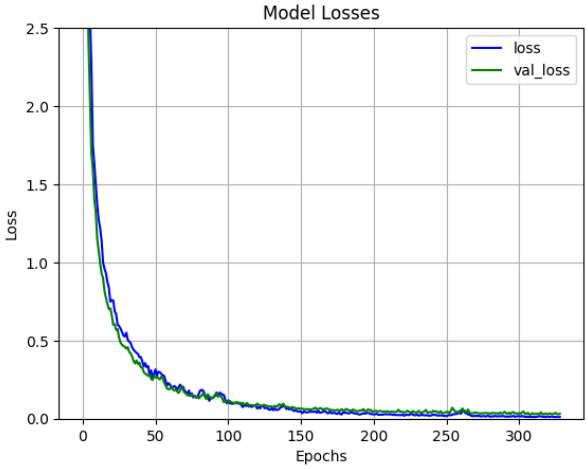
(a) Sada akcí 1 - Experiment B - Ztrátová funkce



(b) Sada akcí 2 - Experiment B - Ztrátová funkce

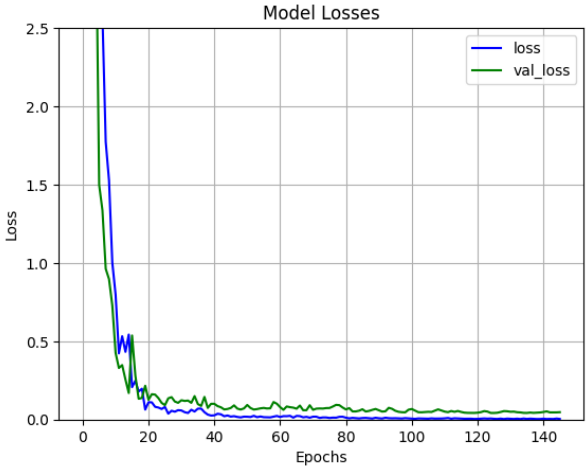


(c) Sada akcí 3 - Experiment B - Ztrátová funkce

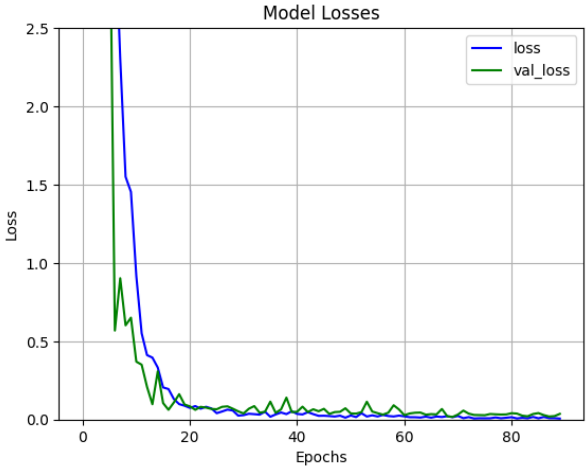


(d) Sada akcí 4 - Experiment B - Ztrátová funkce

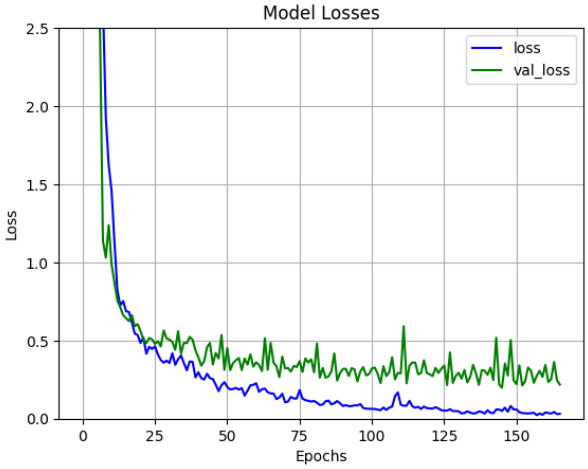
Obrázek C.14: Grafy ztrátových funkcí všech akčních sad Experimentu B datasetu **KARD**



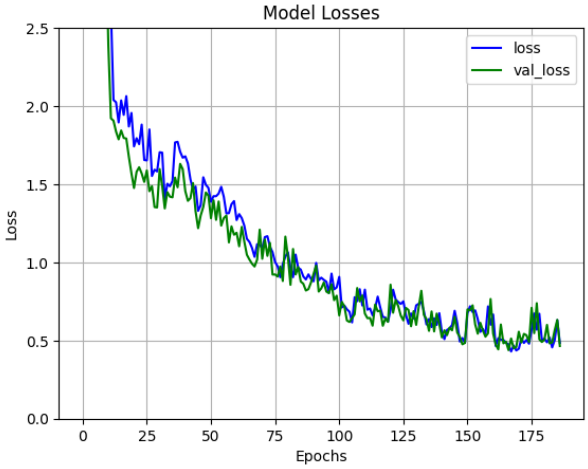
(a) Sada akcí 1 - Experiment C - Ztrátová funkce



(b) Sada akcí 2 - Experiment C - Ztrátová funkce

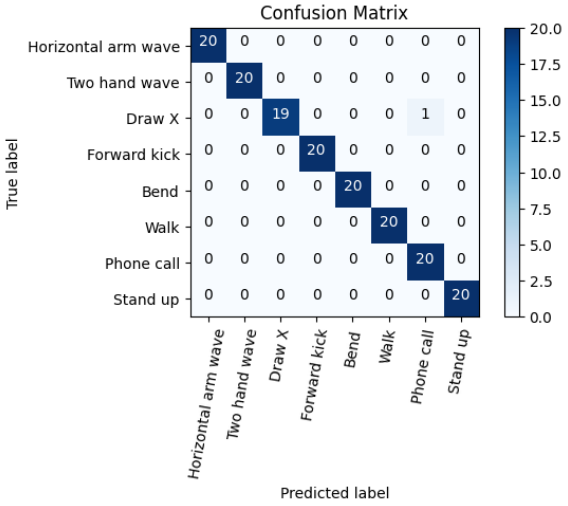


(c) Sada akcí 3 - Experiment C - Ztrátová funkce

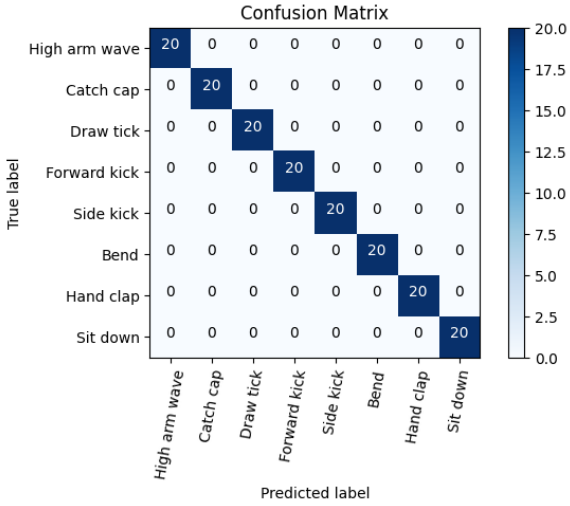


(d) Sada akcí 4 - Experiment C - Ztrátová funkce

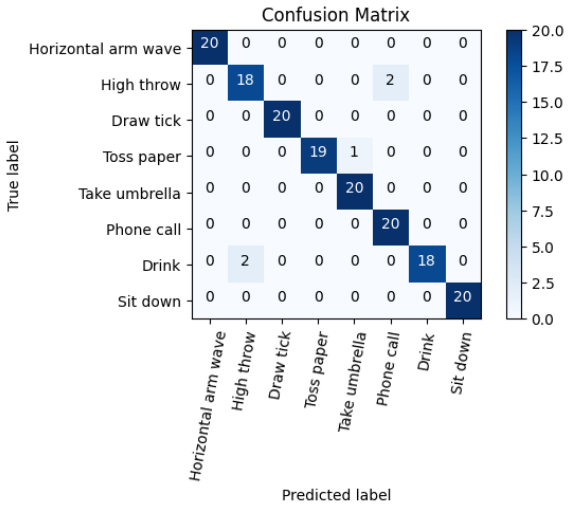
Obrázek C.15: Grafy ztrátových funkcí všech akčních sad Experimentu C datasetu **KARD**



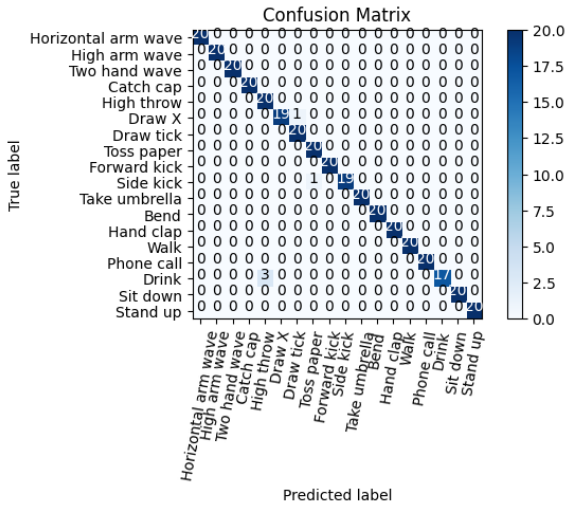
(a) Sada akcí 1 - Experiment A - Konfuzní mapa



(b) Sada akcí 2 - Experiment A - Konfuzní mapa

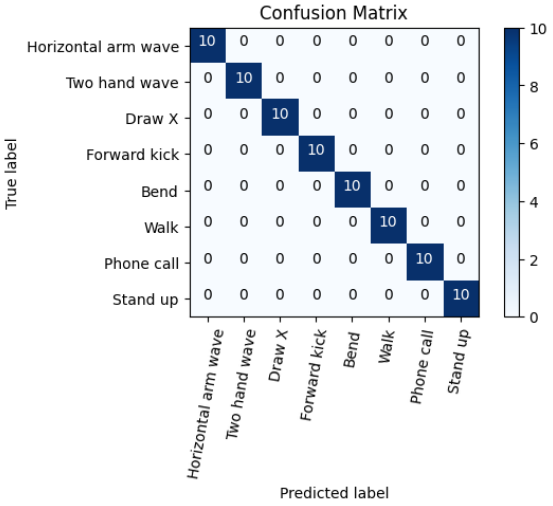


(c) Sada akcí 3 - Experiment A - Konfuzní mapa

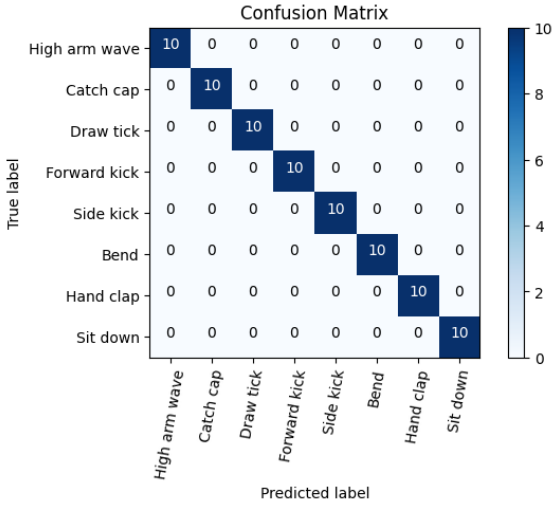


(d) Sada akcí 4 - Experiment A - Konfuzní mapa

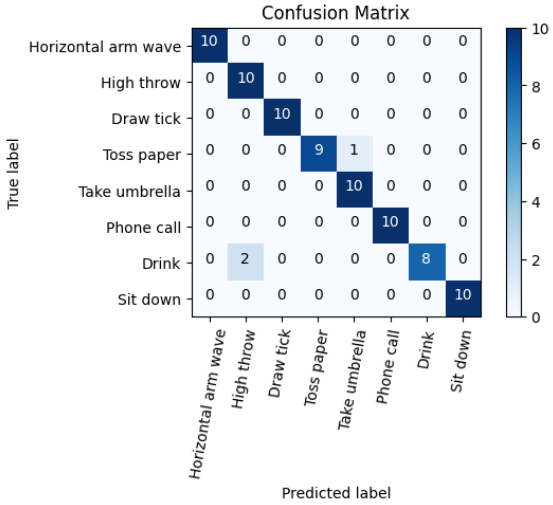
Obrázek C.16: Konfuzní mapy všech akčních sad Experimentu A datasetu **KARD**



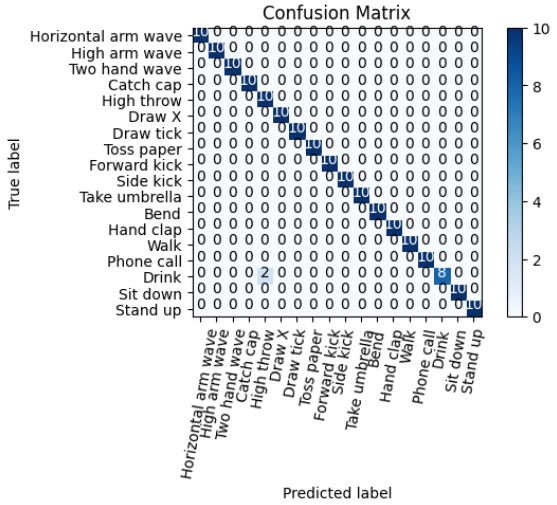
(a) Sada akcí 1 - Experiment B - Konfuzní mapa



(b) Sada akcí 2 - Experiment B - Konfuzní mapa

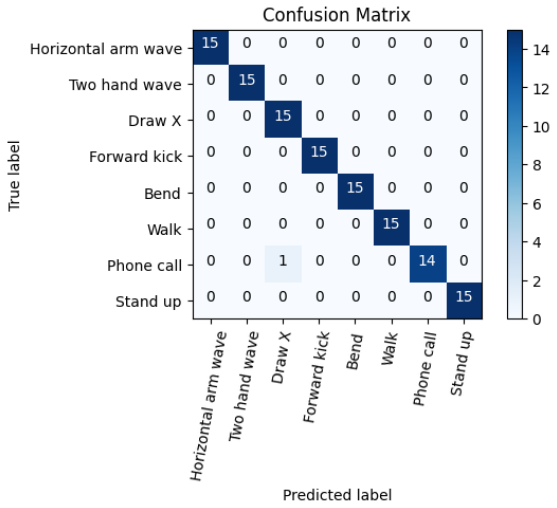


(c) Sada akcí 3 - Experiment B - Konfuzní mapa

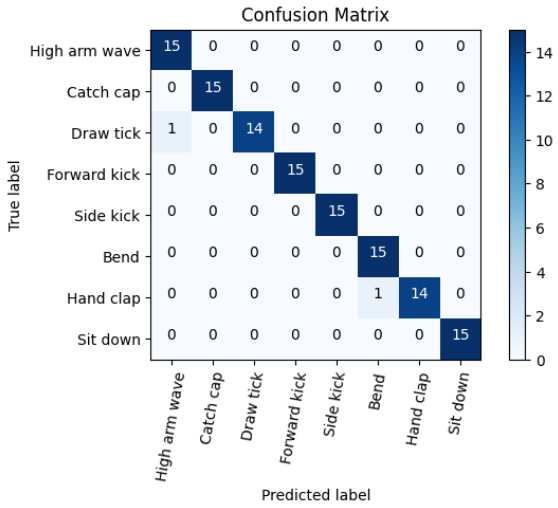


(d) Sada akcí 4 - Experiment B - Konfuzní mapa

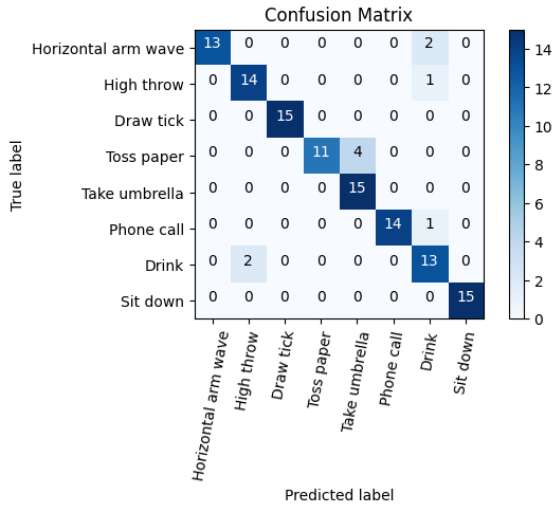
Obrázek C.17: Konfuzní mapy všech akčních sad Experimentu B datasetu **KARD**



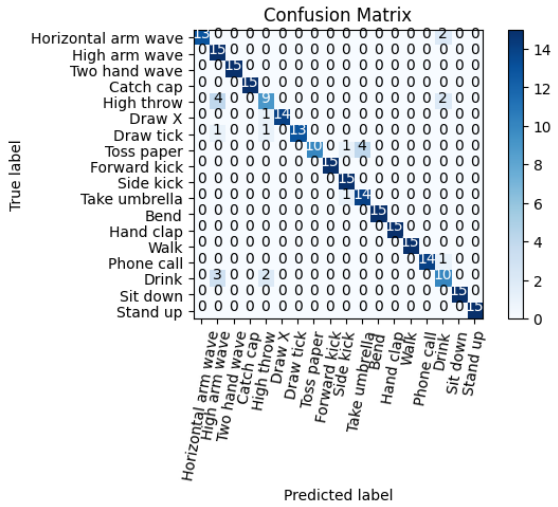
(a) Sada akcí 1 - Experiment C - Konfuzní mapa



(b) Sada akcí 2 - Experiment C - Konfuzní mapa



(c) Sada akcí 3 - Experiment C - Konfuzní mapa



(d) Sada akcí 4 - Experiment C - Konfuzní mapa

Obrázek C.18: Konfuzní mapy všech akčních sad Experimentu C datasetu **KARD**