VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

EXPERIMENTY S URČENÍM LIDSKÉ PÓZY V OBRAZE A VIDEU

EXPERIMENTS WITH ESTIMATION OF HUMAN POSE IN IMAGE AND VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAL HOREJŠ**
AUTHOR

VEDOUCÍ PRÁCE — **prof. Ing. ADAM HEROUT, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

143238

Ústav: Ústav počítačové grafiky a multimédií (UPGM)
Student: **Horejš Michal**
Program: Informační technologie
Specializace: Informační technologie

Název: **Experimenty s určením lidské pózy v obraze a videu**
Kategorie: Počítačové vidění
Akademický rok: 2022/23

Zadání:

1. Seznamte se s problematikou rozpoznání lidské pózy v obraze a videu.
2. Seznamte se s podobami konkrétních pozic a pohybů ve vybraných sportech.
3. Experimentujte s nástroji pro rozpoznání lidské pózy a obecně pro rozpoznávání jevů v obraze s cílem rozpoznávat konkrétní sportovní pozice a pohyby.
4. Shromážděte a získejte vhodné datové sady.
5. Experimentujte s dílčími prvky řešení nad shromážděnými daty a iterativně je vylepšujte.
6. Demonstrujte možnosti vyvíjených metod na datech.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Vincent Dumoulin, Francesco Visin: A guide to convolution arithmetic for deep learning, https://arxiv.org/abs/1603.07285
- Goodfellow et al.: Deep Learning, MIT Press, 2016
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Jan Erik Solem: Programming Computer Vision with Python, O'Reilly Media, 2012

Při obhajobě semestrální části projektu je požadováno:
body 1. a 2., značné rozpracování bodů 3. a 4.

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

Vedoucí práce: **Herout Adam, prof. Ing., Ph.D.**
Vedoucí ústavu: Černocký Jan, prof. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 31.10.2022

Abstrakt

Detekce jevů v obraze má široké uplatnění v mnoha oborech a je proto důležité detekci neustále vyvíjet a vylepšovat. Tato práce se konkrétně zabývá problémem detekce sportovních pozic v obraze a videu. Cílem bylo experimentovat s nástroji pro rozpoznání lidské pózy a obecně rozpoznání jevů. Během experimentů došlo k vytvoření tří nových datových sad, ve kterých jsou využity klíčové body lidského těla. Datasety poté sloužily k trénování několika namodelovaných architektur konvolučních neuronových sítí. Výsledky experimentů ukazují, že vhodné využití klíčových bodů může pomoci s detekcí sportovních pozic.

Abstract

The detection of phenomena in the image has a wide application in many fields and it is therefore important to constantly develop and improve the detection. This work specifically deals with the problem of detecting sports positions in images and videos. The goal was to experiment with tools for human pose recognition and general phenomenon recognition. During the experiments, three new data sets were created, in which key points of the human body are used. The datasets were then trained on several modeled architectures of convolutional neural networks. The results of the experiments show that the appropriate use of key points can help with the detection of sports positions.

Klíčová slova

konvoluční neuronové sítě, rozpoznání jógových pozic, optický tok, extrakce klíčových bodů, sítě s více vstupy

Keywords

convolutional neural networks, yoga pose detection, optical flow, key points extraction, multi-input networks

Citace

HOREJŠ, Michal. *Experimenty s určením lidské pózy v obraze a videu.* Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

Experimenty s určením lidské pózy v obraze a videu

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Michal Horejš
8. května 2023

Poděkování

Chtěl bych poděkovat prof. Ing. Adamovi Heroutovi, Ph.D. za odbornou pomoc a cenné rady při tvoření bakalářské práce a poskytnutí přístupu k výpočetnímu serveru sophie.fit.vutbr.cz.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Počítačové vidění se stává stále důležitějším oborem výzkumu, protože umožňuje automatizovat mnoho úkolů, které by jinak vyžadovaly manuální práci. Detekce lidské pózy je jedním z takových úkolů, které se dají automatizovat, což má potenciál výrazně usnadnit a zrychlit mnoho procesů. Jedním z těchto procesů je rozpoznávání sportovních póz.

Cílem této práce je hledání metod, které by mohly vylepšit rozpoznání sportovních pozic v obraze a videu. Pro tento účel byl použit dataset, který obsahuje obrázky 82 různých jógových pozic.

Kapitola 2 se zabývá konvolučními neuronovými sítěmi, konkrétně jejich architekturou a trénováním. Dále kapitola vysvětluje pojmy, které jsou vázány s konvolučními neuronovými sítěmi a jejich trénováním. Jsou zde uvedeny i významné architektury těchto sítí.

Kapitola 3 představí použitou datovou sadu, provedené modifikace datové sady a popisuje rozmanitost jednotlivých obrázků.

Poté jsou v kapitole 4 popsány použité nástroje k modelování sítí, nastavení jejich konfigurace a způsoby trénování sítí. Kapitola se zabývá také používáním a zpracováním datasetu.

V kapitole 5 jsou uvedeny návrhy experimentů provedených v rámci této práce. Experimenty jsou stavěny na extrakci klíčových bodů lidského těla a používaní různých architektur sítí. Je zde vysvětlen i způsob získání těchto klíčových bodů. Na tuto kapitolu navazují kapitoly 6 a 7.3, ve kterých jsou detailně vysvětleny přípravy a průběhy jednotlivých experimentů. Výsledky experimentů jsou pak vzájemně porovnány.

Kapitola 2

Detekce a klasifikace lidské pózy

Tato kapitola se zaměřuje na problematiku detekce a klasifikace lidské pózy v obraze. Tento problém je v posledních letech velmi aktuální, neboť rozpoznání lidské pózy má mnoho praktických využití, jako například v oblasti sportu, medicíny, bezpečnosti, průmyslu a mnoha dalších. Existuje mnoho přístupů a metod pro detekci a klasifikaci lidské pózy, které se vyznačují různými výhodami a nevýhodami. Nejběžnějším přístupem k detekci a klasifikaci lidské pózy, a obecně jevů v obraze, je použití konvolučních neuronových sítí.

2.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě (dále CNN nebo modely) patří mezi mnoho druhů umělých neuronových sítí. Jsou speciálně navrženy pro zpracování obrazových dat. Obvykle se skládají z konvolučních vrstev, přes které projde vstupní obrázek a vytváří z něho mapy příznaků (*feature maps*). Dále často obsahují takzvané pooling vrstvy a na konci plně propojené vrstvy (*fully connected*). V CNN jsou také používány v některých vrstvách aktivační funkce a je potřeba tyto sítě také trénovat vhodným způsobem. V následujících sekcích jsou použité termíny vysvětleny a jsou zde představeny i některé významné architektury CNN.

2.1.1 Konvoluční vrstvy

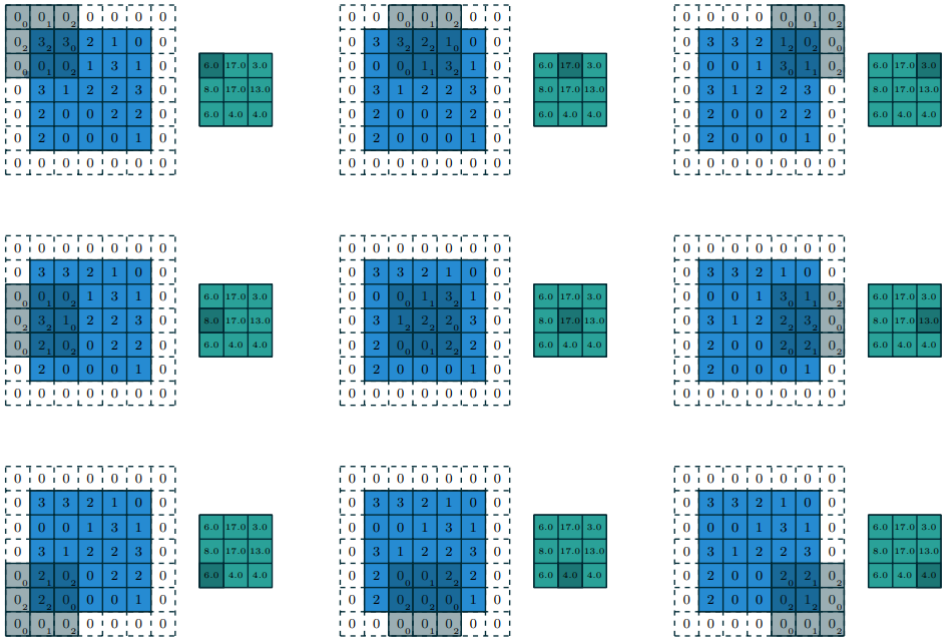
Hlavním pilířem konvolučních neuronových sítí jsou konvoluční vrstvy, které mají schopnost rozpoznávat různé vzorce a vlastnosti v obraze, jako jsou hrany, tvary a textury. Pro zpracování obrazu se používají 2D konvoluce a jejich vstupem jsou například černobílé obrázky, reprezentované maticí. Jeli vstupem barevný obrázek o třech kanálech, který je reprezentován jako tři matice poskládané za sebou (**tensor** o hloubce 3), konvoluční vrstva zpracovává každou matici samostatně.

Konvoluční vrstvu charakterizují čtyři základní hodnoty: rozměr vstupní matice (obrázku), velikost konvolučního jádra (*kernel size*), velikost kroku (*stride*) a nulová výplň (*zero padding*) [5]. Jako nejběžnější velikosti konvolučního jádra se používají velikosti 3×3 nebo 5×5. To se poté posouvá nad vstupní matici o zadanou velikost kroku. Velikost kroku do šířky a do výšky se může lišit. Vstupní obrázek může být také rozšířen pomocí nulové výplně. Ta ,dle zadané hodnoty, rozšíří matici rovnoměrně do všech stran a do nově přidaných pixelů dosadí nulovou hodnotu. Používá se pro možnost aplikovat konvoluci na okrajové pixely a tím zachovat informace o krajních pixelech obrázku. Výpočet výstupní hodnoty pixelu je určen součinem jednotlivých prvků konvolučního jádra s korespondujícími prvky matice a tyto součiny jsou následně sečteny (obrázek 2.1).

Výsledný rozměr výstupní matice, po aplikování konvoluce, lze vypočítat následovně:

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1 \tag{2.1}$$

V rovnici 2.1 udává i šířku a výšku vstupní matice, k výšku a šířku konvolučního jádra, s velikost horizontálního a vertikálního kroku a p velikost rozšíření matice do všech stran a vyplnění těchto okrajů hodnotou 0.



Obrázek 2.1: 2D konvoluce pro velikost vstupní matice $i = 5$, velikost konvolučního jádra $k = 3$, velikost kroku $s = 2$ a nulové výplně $p = 1$. Převzato z [5].

2.1.2 Aktivační funkce

Aktivační funkce slouží k zavedení nelinearity do CNN, jelikož lineární operace nejsou dostatečně silné k rozpoznání komplikovaných vzorů z poskytnutých dat. Nelinearita umožňuje síti se tyto vzory naučit a rozpoznat například hrany objektů nebo anomálie v obraze. Aktivační funkce se aplikuje na výstupy různých vrstev v CNN a upravují jednotlivé prvky matice.

Sigmoid

Aktivační funkce Sigmoid se používá pro mapování vstupních dat na hodnoty mezi 0 a 1. Vysoké hodnoty matice jsou konvertovány na hodnoty blížící se 1 a malé hodnoty naopak na hodnoty blížící se nule. Využívá se často pro sítě predikující pravděpodobnost nebo pro klasifikaci do dvou tříd.

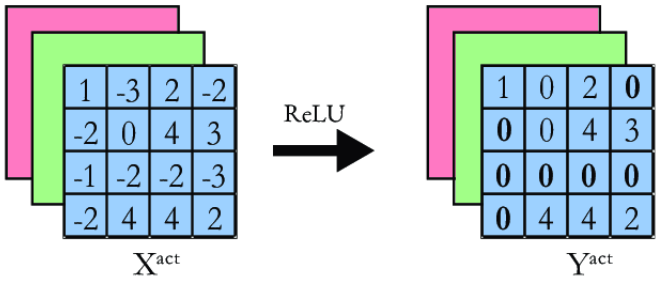
Softmax

Softmax aktivační funkce je často používána ve finální vrstvě CNN pro klasifikaci do dvou a více tříd. Tato funkce transformuje výstup poslední vrstvy sítě na pravděpodobnostní

rozdělení, které určuje, s jakou pravděpodobností patří vstupní data do daných tříd. Softmax zaručuje, že součet pravděpodobností všech tříd je roven 1. Dále je funkce diferencovatelná, což umožňuje aplikovat algoritmy učení zpětnou propagací chyby při trénování sítě.

Rectified Linear Unit (ReLU)

Jedná se o nejběžněji používanou aktivační funkci při extrakci mapy příznaků. Funkce je definována jako $f(x) = max(0, x)$. Pokud je vstup kladný, výstup je roven vstupu. Pokud je vstup záporný, výstup je roven nule. ReLU je rychlá a jednoduchá funkce na výpočet a dosahuje dobrých výsledků v praxi. Typicky se používá ve skrytých vrstvách sítě. Vylepšením ReLU je aktivační funkce Leaky ReLU. Je definovaná jako $f(x) = max(ax, x)$, kde a je malé kladné číslo. Pokud je vstup záporný, výstup je roven $a \cdot x$ a pokud kladný, je výstup roven vstupu. Leaky ReLU řeší problém mrtvých neuronů[1], který může nastat u ReLU pro záporné hodnoty vstupu.



Obrázek 2.2: Ukázka aktivační funkce ReLU na obrázku se třemi barevnými kanály. Převzato z [3].

2.1.3 Pooling vrstvy

Pooling vrstvy jsou další součástí konvolučních neuronových sítí a bývají umístěny mezi konvolučními vrstvami sítě. Tyto vrstvy přijímají data ve formě tenzorů o různých rozměrech z předchozí konvoluční vrstvy. Na ně aplikují transformaci, která redukuje rozměry těchto tenzorů (výšku a šířku), čímž snižují počet parametrů sítě a tím i její výpočetní výkon potřebný při trénování. Při této redukci se snaží zachovat nejdůležitější informace. Zmenšení prostorového rozlišení dat může zmenšit problém přetrénování (*overfitting*) a zlepšit generalizaci[2] sítě na nová data.

Prvním způsobem jak provést pooling je **average pooling**. Ten v vezme každý blok vstupu a vypočítá průměr všech hodnot ve vybraném bloku. Tato hodnota je použita jako výstup. Average pooling je používán v okamžiku, kdy je důležitý celkový vzhled obrazu. Pro nalezení významných prvků v obraze, jako jsou hrany nebo rohy, se používá **max pooling**. Ten v každém bloku vstupního obrazu vybere maximální hodnotu pixelu a tuto hodnotu použije na výstup. Velikost bloku se obvykle volí takovým způsobem, aby se po aplikaci pooling vrstvy zmenšila výška a šířka vstupního obrazu na polovinu.

[1] Mrtvý neuron je neuron, který má výstup o hodnotě 0 bez ohledu na jeho vstupní hodnotu. Výskyt mnoha takových neuronů může mít negativní vliv na úspěšnost predikce sítě.

[2] Generalizace značí schopnost modelu správně predikovat třídy pro neznámá data.

2.1.4 Plně propojené vrstvy

Plně propojené vrstvy (*dense*) jsou umístěny na konci sítě po konvolučních a pooling vrstvách. Slouží pro klasifikaci mapy příznaků do tříd. Tyto vrstvy obsahují určité množství neuronů a vyznačují se spojením každého neuronu v předchozí vrstvě s každým neuronem v následující vrstvě. Neurony jsou propojeny pomocí vah a biasů. Váha slouží k vynásobení vstupní hodnoty, ke které je poté přičtena hodnota biasu. Trénování plně propojených vrstev probíhá pomocí algoritmů učení zpětnou propagací chyby (sekce 2.3.3), stejně jako konvoluční vrstvy. Jelikož plně propojené vrstvy mohou být náchylné k přetrénování, je adekvátní používat techniky, pomocí kterých by se riziko přetrénování minimalizovalo (sekce 2.3.4).

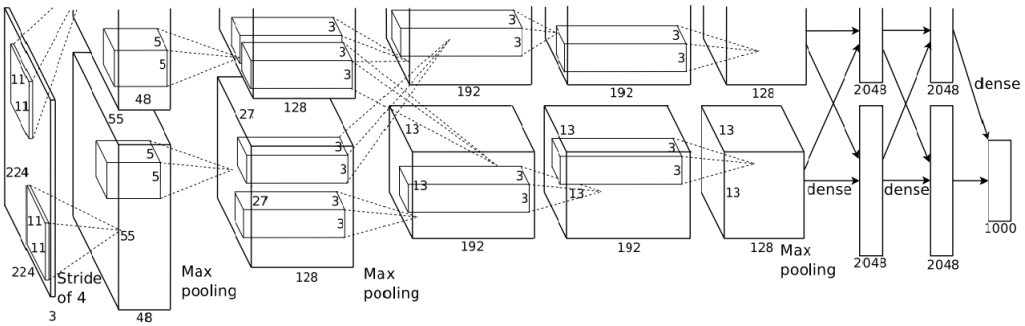
2.2 Významné modely CNN

Konvoluční neuronové sítě a jejich architektury jsou a byly vyvíjeny k řešení úkolů v oblasti počítačového vidění. V průběhu let byly zdokonalovány a vyvíjeny nové architektury a jejich implementace. Sítě se také s postupem času začaly modelovat mnohem rozmanitější a komplexnější. V této sekci jsou ukázány jedny z nejvýznamnějších sítí, které dosahovaly kvalitních výsledků v rámci různých soutěží.

2.2.1 AlexNet

AlexNet [12] je jedna z prvních nejvýznamnějších konvolučních neuronových sítí a přinesla revoluci v oblasti rozpoznávání obrazu. Jedná se o CNN s pěti konvolučními vrstvami a třemi max pool vrstvami. Na konci sítě jsou tři plně propojené vrstvy. Jako aktivační funkce jednotlivých vrstev zde byla použita ReLU a pro poslední vrstvu Softmax, která zajišťuje klasifikaci do jednotlivých tříd. Jelikož AlexNet obsahuje přes 60 miliónů parametrů, byly zde použity techniky pro prevenci přetrénování. Jedná se o dropout v plně propojených vrstvách a augmentaci vstupních dat.

V roce 2012 byla architektura AlexNetu použita v soutěži ImageNet Large Scale Visual Recognition Challenge (ILSVRC) a dosáhla o 10,8 % nižší chyby oproti druhému nejlepšímu řešení[14].



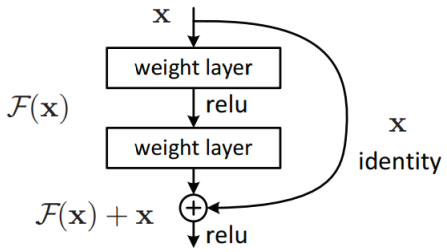
Obrázek 2.3: Ilustrace architektury AlexNetu. Převzato z [12].

2.2.2 ResNet

Residual Network [8], zkráceně ResNet, byl poprvé představen týmem výzkumníků z Microsoft Research Asia. Architektura byla navržena s cílem řešit problém s gradienty, které se

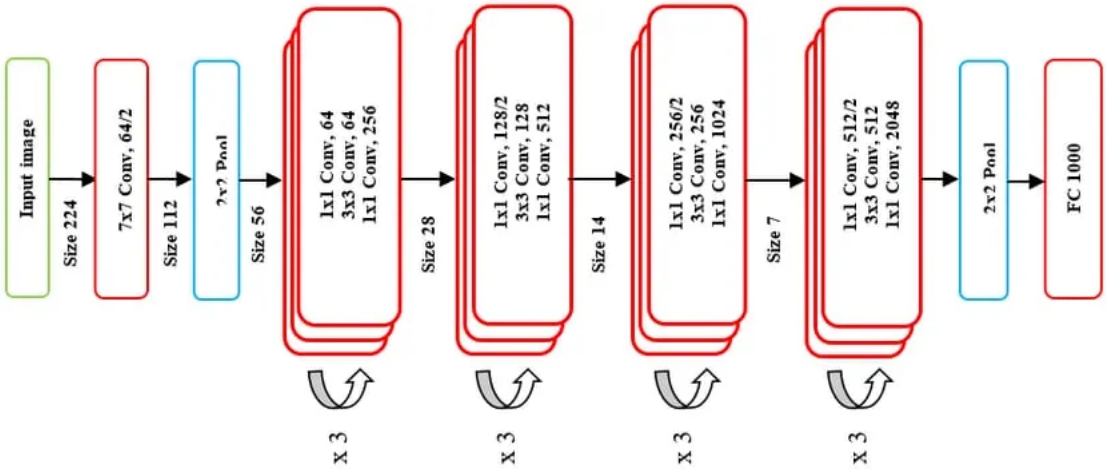
mohou ztratit *(gradient vanishing)* či zeslabit při trénování. Využívá residuálních bloků, které přidávají mezi vstupem a výstupem bloku propojovací kanál *(skip connection nebo identity mappings)*. Ten zajišťuje, že výstup bloku se může naučit pouze změny oproti vstupu, což usnadňuje učení.

Existuje několik druhů ResNet architektur, které se liší hloubkou sítě a počtem parametrů. Pro trénování na menších datasetech se používají ResNet-18 a ResNet-34, které obsahují 18 a 34 vrstev s váhami *(weight layer)*. Mezi další architektury ResNetu se řadí ResNet-50, ResNet-101 a ResNet-152, které obsahují 50, 101 a 152 vrstev s váhami. Ty poskytují větší kapacitu pro učení a vyžadují vetší množství dat pro trénování.



Obrázek 2.4: Residuální blok použitý v architekturách ResNet-18 a ResNet-34. Převzato z [8].

V roce 2015 byla architektura ResNetu použita v soutěži ILSVRC. V kategorii klasifikace obrázků se ResNet umístil na 1. místě se ztrátou chyby 3,57 % [8] pro top-5 predikci.[3]. V porovnání s ILSVRC v roce 2012, kde byla chyba přes 15 % na výherní architektuře AlexNet[12], se jedná o opravdu obrovské zlepšení.



Obrázek 2.5: Architektura ResNet-50. Převzato z [13].

Vylepšením ResNetu je ResNetv2. Hlavním cílem ResNetu-v2 je vylepšit původní architekturu ResNetu. Výsledky experimentů této studie [9] ukazují, že ResNetv2 dosahuje vyšší

[3]Top-5 predikce značí, že z výstupu modelu bylo vybráno 5 predikovaných tříd s nejvyšší pravděpodobností pro daný vstup a mezi nimiž se nachází predikce správné třídy.

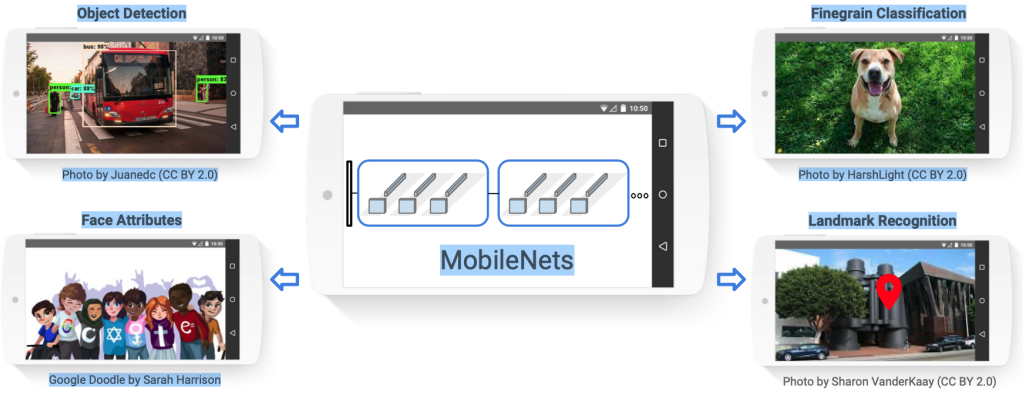
přesnosti na několika datových sadách, včetně ImageNetu a CIFAR-10 [17]. První změnou je úprava identity mappings, kde je přidána Batch Normalization vrstva a aktivační funkce ReLU před vynásobením vstupních dat váhou.

Další přidanou funkcí je **bottleneck residual block**, který nahrazuje residuální blok ResNetu. Bottleneck residual block je složen ze tří vrstev a obsahuje tak o jednu vrstvu víc, než klasický residual block. První vrstvou je konvoluční vrstva s jádrem 1×1, na kterou navazuje konvoluční vrstva s jádrem 3×3. Poslední vrstva je opět konvoluční vrstva s jádrem 1×1. První a poslední vrstva mají za úkol redukovat dimenzi dat. Druhá vrstva má na starosti extrakci příznaků. Bottleneck residual block přinesl významné zlepšení výkonu a efektivity sítě.

2.2.3 MobileNet

Zmíněné CNN architektury jsou při jejich trénování velmi výpočetně náročné a uložení jejich vah a architektur zabere mnoho paměti. Proto nejsou architektury vhodné pro používání na zařízeních s menší výpočetní kapacitou a pamětí. MobileNet [10] je architekturou CNN navrženou speciálně pro mobilní zařízení, tablety aj. Model byl představen týmem výzkumníků z Googlu v roce 2017.

MobileNet je založen na hloubkově separabilní konvoluci *(Depthwise Separable Convolution)*, která rozděluje konvoluční vrstvu na dvě části: hloubkovou konvoluci *(depthwise convolution)* a bodovou konvoluci *(pointwise convolution)*. Hloubková konvoluce aplikuje jeden filtr na každý vstupní kanál. Bodová konvoluce aplikuje 1×1 konvoluci pro spojení výstupů hloubkové konvoluce. Mezi další klíčovou vlastnost patří použití globálního průměru v koncové vrstvě. Kombinace těchto vlastností vyústila ke drastickému zmenšení počtu parametrů sítě a nároky na výpočetní výkon za cenu zmenšení přesnosti predikce.



Obrázek 2.6: Možnosti využití MobileNetu. Převzato z [10].

2.3 Trénování CNN

Cílem trénování CNN je přizpůsobování vah a biasů tak, aby se minimalizovala chybnost predikce na vstupních trénovacích datech a tato data co nejlépe generalizovala. Proces trénování je založen na algoritmu učení, který se snaží váhy pro vstupní data optimalizovat a zároveň minimalizovat ztrátovou funkci. Při trénování je důležité určit optimální velikost

učení *(learning rate)*, počet vzorků zpracovaných před aktualizací vah a biasů *(batch size)*, počet průchodů celým trénovacím datasetem *(epochs)* aj.

2.3.1 Ztrátová funkce

Ztrátová funkce je matematická funkce, která měří, jak dobře neuronová síť předpovídá výstupy pro vstupy. Cílem při trénování modelu je minimalizovat hodnotu této funkce. Hodnota této funkce se nazývá chyba (*error* nebo *loss*) pro trénovací sadu dat a validační chyba *(validation loss)* pro validační sadu dat. Chyba je poté používána pro penalizaci modelu, při špatné predikci tříd vstupních dat, pomocí zpětné propagace.

Categorical cross-entropy

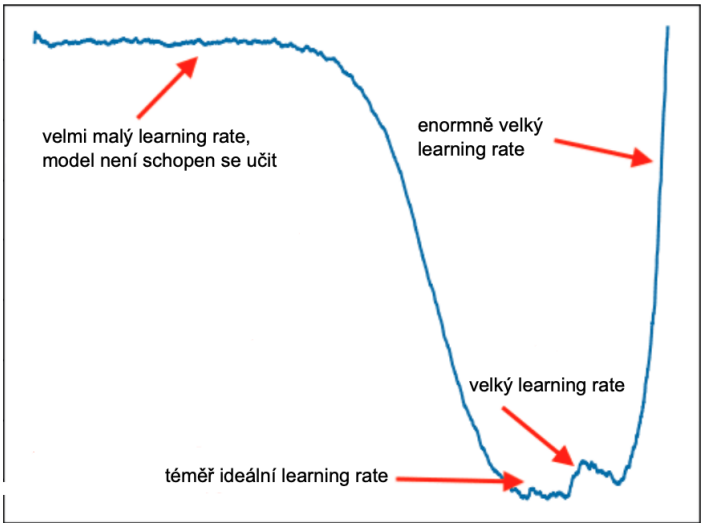
Categorical cross-entropy se používá při klasifikaci vstupů do více než dvou tříd. Pokud je potřeba klasifikovat N tříd, tak výstupní vrstva modelu má N neuronů, z nichž každý odpovídá jedné třídě. Categorical cross-entropy porovnává výstup sítě s očekávanou hodnotou pro každou třídu a následně vypočítá ztrátu na základě rozdílu mezi nimi. Zpravidla se používá s aktivační funkcí softmax. Matematicky je categorical cross-entropy formulována v následující rovnici (2.2), kde y_i je očekávaná hodnota pro i-tou třídu (1 nebo 0), $\hat{y}_i$ je hodnota predikovaná sítí pro i-tou třídu pomocí aktivační funkce softmax a M značí počet tříd[6].

$$loss = \sum_{i=1}^{M} y_i \cdot \log \hat{y}_i \tag{2.2}$$

Při trénování CNN je tato ztráta vypočtena pro každý obrázek a tyto ztráty jsou poté zprůměrovány. Pomocí tohoto výpočtu lze sledovat generalizaci sítě na datech.

2.3.2 Optimalizátor

Optimalizace je důležitou součástí při trénování CNN. Využívá se pro nalezení optimálních vah a biasů sítě a to nalezením minima ztrátové funkce. U optimalizátoru se nastavuje parametr learning rate *(rychlost učení)*, který určuje velikost kroku při každé iteraci. Pro nalezení požadovaného minima se běžně využívá algoritmus **gradient descent**. Ten využívá výpočtu gradientu funkce v každém bodě kroku a dle hodnoty learning rate postupuje s určitou velikostí proti směru gradientu, což umožňuje se přibližovat k minimu funkce. Pokud je learning rate moc vysoký, může optimalizátor minimum přeskočit nebo kolem minima oscilovat. Tím pádem nedojde k nalezení nejoptimálnějších parametrů sítě. V opačném případě, tedy když je learning rate nízký, může dojít k nalezení lokálního minima a nedojde k nalezení minima globálního. Znázorněno na obrázku (2.7).



Obrázek 2.7: Ukázka různých velikostí učení.

Adaptive Moment Estimation

Adaptive Moment Estimation, zkráceně Adam [11], je jedním z nejpopulárnějších optimalizačních algoritmů. Adam využívá technik adaptivního krokování a momentového odhadu. Prvním momentem je průměr a druhým momentem je variace gradientů. Ty jsou aktualizovány při každém kroku trénování. Tyto odhady jsou poté využity pro upravování kroku učení podle četnosti výskytu daných gradientů.

Adam prokázal vysokou efektivitu v řešení problému mizejících gradientů. Další výhodou je rychlost tréninku sítě a schopnost se přizpůsobit různě velkým krokům při trénování.

Stochastic Gradient Descent

Stochastic Gradient Descent, zkráceně SGD, je optimalizační algoritmus, který počítá gradienty pouze na určitém počtu vzorků namísto celého datasetu. Díky tomuto přístupu je SGD výpočetně velmi efektivní a používá se pro trénování sítí na velkých datasetech. Dále SGD aktualizuje parametry modelu při trénování po každém, předem definovaném, počtu vzorků a tím se může rychleji přiblížit ke globálnímu minimu.

Jelikož je gradient vypočten pouze na omezeném počtu vzorků, nemusí SGD gradienty funkce vypočítat správně. To může vést k pomalejšímu tréninku a nemusí nalézt nejoptimálnější parametry sítě. Důležitost zvolení správných parametrů popisuje tato studie [16].

2.3.3 Zpětná propagace

Zpětná propagace *(back propagation)* je algoritmus pro trénování neuronových sítí, který umožňuje síti se učit ze špatně predikovaných vstupních dat, upravováním vah jednotlivých neuronů a tím minimalizovat hodnotu ztrátové funkce. Hlavní myšlenkou zpětné propagace je propagovat chybu z výstupní vrstvy až ke vstupní vrstvě a aktualizovat váhy každé vrstvy.

Překážkou zpětné propagace jsou problémy mizejících nebo explodujících gradientů. V neuronových sítích může být vypočítaný gradient velmi malý nebo naopak obrovský, což může ztížit či znemožnit aktualizování vah efektivně. Pro vyřešení problémů mizejících nebo

explodujících gradientů jsou používány techniky jako inicializace vah, změna měřítka výstupu vrstev nebo použití korektních optimalizátorů s vhodnou inicializací jejich parametrů.

2.3.4 Přetrénování sítě

Přetrénování sítě značí situaci, kdy model dosahuje dobrých výsledků predikce na trénovací sadě, ale díky špatné generalizaci dat dosahuje špatných predikcí na neznámých datech. To značí, že se model naučil špatně rozpoznávat jednotlivé vzory, což je pro síť kritická vlastnost. Metody používané jako prevence pro přetrénování sítě se nazývají regularizační techniky.

Vyřazení náhodných neuronů vrstvy

Vyřazení náhodných neuronů vrstvy (dále *dropout*) je technika regularizace, která se používá k zabránění přetrénování CNN a je převážně používána v kombinaci s plně propojenými vrstvami.

Principem dropoutu je náhodné vyřazení některých neuronů během trénování sítě. V každé iteraci tréninku je náhodně vybráno určité procento neuronů v dané vrstvě, které budou ignorovány. Konkrétně budou dočasně odpojeny od všech ostatních neuronů jak na vstupu, tak na výstupu neuronu. Touto technikou se síť stává robustnější vůči drobným změnám ve vstupních datech a nezaměřuje se na konkretní vzory, které by mohly způsobit přetrénování.

Předčasné zastavení

Předčasné zastavení *(early stopping)* je další technikou která může pomoci s přetrénováním. Myšlenkou této techniky je zastavit trénink v nejideálnější části, kdy model nejlépe generalizuje na validačních datech. Zde ale vzniká otázka: „Kdy při tréninku model nejlépe generalizuje a měli bychom zastavit trénink ?“. Využívá se zde nejčastěji monitorování validační ztráty. Validační ztráta by se při tréninku měla zmenšovat a v momentě, kdy se začne zvyšovat, by se mělo zavolat předčasné ukončení tréninku.

Předčasné zastavení může ale vést k suboptimálnímu natrénování sítě v momentě, kdy nedojde k úplné konvergenci. Proto se uvádí tolerance a k přerušení nedojde, dokud nedojde k jejímu porušení. Tolerance musí být pečlivě vybrána v závislosti na chování sítě při tréninku. Často používanou tolerancí je například počet epoch po kterých se má trénování přerušit, pokud nedojde ke zmenšení validační ztráty.

Augmentace dat

Augmentace dat se používá pro zvětšení rozmanitosti a počtu trénovacích dat a tím pomoci modelu lépe generalizovat při tréninku. Zvětšení počtu trénovacích dat je dosáhnuto generováním (kopírováním) trénovacích dat. Samotné generování by nemělo na trénink ale žádný vliv, jelikož by se model učil z opakujících se dat. Z toho důvodu jsou na originální a nově vygenerovaná data použity různé transformace. V případě trénovacích dat ve formě obrázků jsou použity náhodné rotace, převrácení, přiblížení, ořezání nebo modifikace barevných složek obrázků.

Využití augmentace dat může být nápomocné, jsou-li data v určitých ohledech limitována (například když je CNN pro zvolený dataset moc velká). Augmentace musí být zvoleny odpovídajícím způsobem dle chování sítě při tréninku. Špatně zvolené nebo příliš silné

augmentace mohou naopak trénování sítě uškodit. Při náhodném ořezání nebo přiblížení může například dojít ke smazání klíčové informace z obrázku a tím znemožnit správnou predikci.

Obrázek 2.8: Ukázka augmentací na obrázku převzatého z datasetu Yoga-82 [18]. Mezi použité transformace patří rotace, přiblížení, převrácení, posun a změna jasu.

Kapitola 3

Datová sada

Datová sada je soubor dat, který se používá pro trénování, validaci a testování architektur CNN. Po konzultaci s vedoucím práce byla vybrána datová sada Yoga-82 (dále dataset) [18]. Dataset obsahuje 82 různých tříd, kde každá třída reprezentuje jednotlivou pozici při cvičení jógy. Článek [19] uvádí, že dataset celkem obsahuje 28478 obrázků.

Dataset je distribuován ve formě několika textových souborů. Každý soubor obsahuje URL odkazy na jednotlivé obrázky a jména těchto souborů udávají, jaká jógová pozice je na obrázcích předváděna. Pro stažení jednotlivých obrázků a jejich následné rozdělení do patřičných tříd jsem vytvořil skript. Jednotlivé třídy jsou po spuštění skriptu reprezentovány složkami, kde každá složka obsahuje konkrétní obrázky jógové pozice. Název složky je poté použit jako jméno třídy.

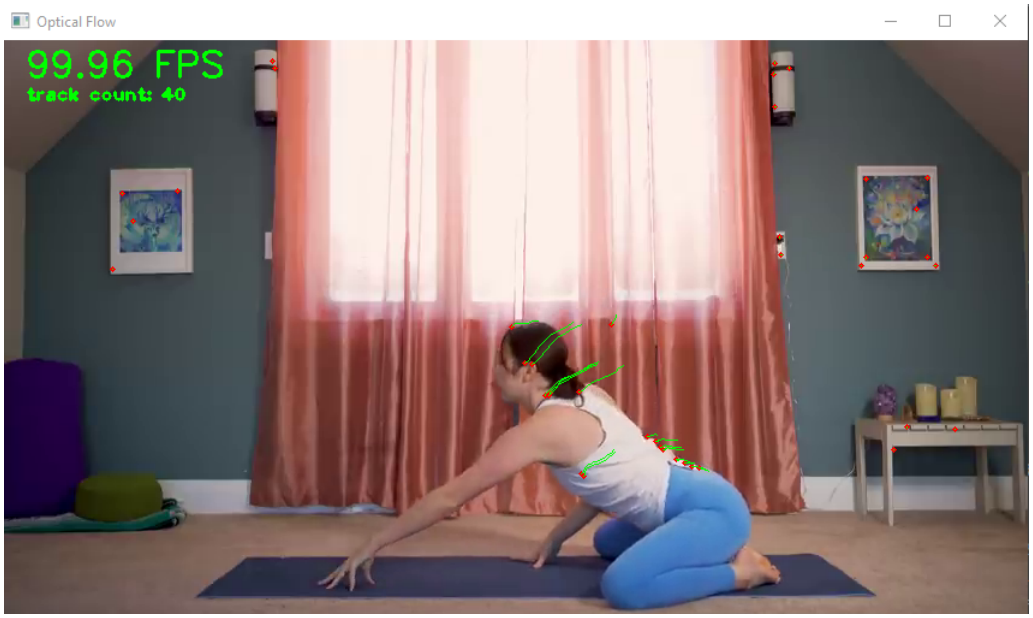
Jelikož byly obrázky poskytnuty pouze URL odkazem, mnoho obrázků již nebylo dostupných ke stažení. Staženo bylo ~12000 obrázků. Dostupnost obrázků pro každou třídu byla velmi různorodá a dataset se tak stal nevyváženým[1]. Dále některé webové stránky nedisponovaly potřebným obrázkem a místo něj byl stažen například chybový či černý obrázek. Dataset jsem ručně přetřídil a takové obrázky byly smazány. Dataset byl následně částečně vyvážen.

3.1 Vyvážení datasetu

Při trénování CNN na nevyváženém datasetu může dojít ke snížení přesnosti predikce a model se bude lépe učit na třídách, které jsou více zastoupeny. Pro minimalizaci tohoto problému se datasety vyvažují. V této práci vznikl pro vyvážení datasetu skript za použití knihovny OpenCV [15, Kapitola 10]. Ten načte video, které obsahuje jedince cvičícího jógu, uložené na disku. Následně pomocí metody **Lucas–Kanade** odhaduje **optický tok** *(optical flow)* objektů ve videu.

Optický tok popisuje pohyb objektů v obraze nebo videu mezi jednotlivými snímky. Toho je poté ve skriptu využito pro určení pohybu. Pokud není v úseku několika snímků odhadnut téměř žádný pohyb, je snímek uložen. Následně bylo potřeba manuálně tyto uložené snímky roztřídit do patřičných složek. Obrázky byly vkládány do složek s nejmenším počtem vzorků.

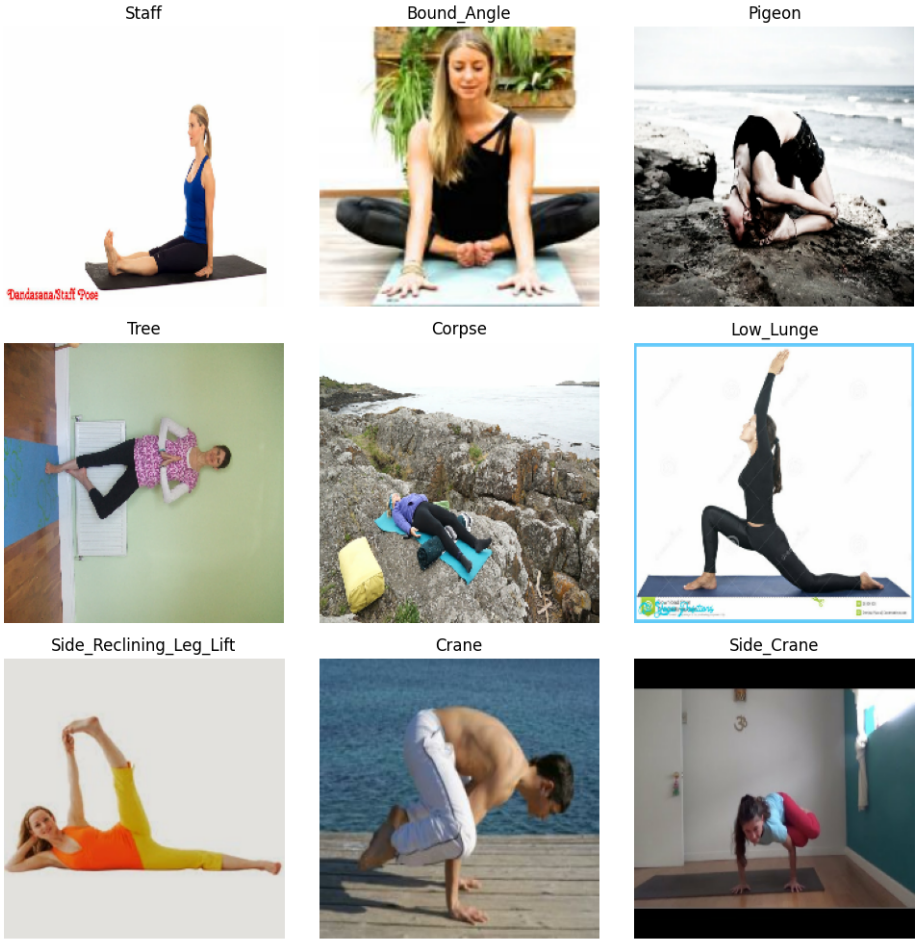
[1] Některé třídy mají výrazně méně vzorků než ostatní třídy.



Obrázek 3.1: Ukázka optického toku detekujícího pohyb ve videu. Červené tečky znázorňují aktuální pozici detekovaného pixelu metodou Lucas–Kanade. Zelené čáry reprezentují pozici detekovaného pixelu v posledních 75 snímcích.

Některé jógové pozice obsažené v datasetu nejsou velmi známé a bylo velmi obtížné získat alespoň nějaké obrázky pro vyvážení datasetu. Proto došlo pouze k částečnému vyvážení datasetu. Každá třída nyní obsahuje ~100 - 300 obrázků konkrétní jógové pozice. Při vyvažování byl dataset rozšířen o ~4500 nových obrázků a celkem obsahuje 16712 obrázků. Obrázky byly náhodně rozděleny v poměru 3 : 1 na trénovací sadu (obsahující 12535 obrázků) a validační sadu (obsahující 4177 obrázků).

Dále je nutné podotknout, že obrázky datasetu jsou velmi rozmanité. Rozmanitost obrázků je určena zobrazením jógových pozic z různých úhlů, za odlišných světelných podmínek nebo v rozdílných prostředích (např. uvnitř nebo venku). Dataset obsahuje i kreslené obrázky.



Obrázek 3.2: Ukázka náhodně vybraných obrázků z rozšířeného datasetu. Obrázky jsou znázorněny v rozlišení 224×224 pixelů. Nad každým obrázkem je uveden korespondující název třídy (jógové pozice).

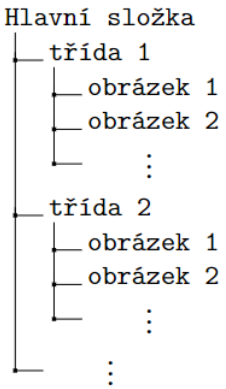
Kapitola 4

Knihovna Tensorflow

Tensorflow [2] je knihovna vyvinutá společností Google a její zdrojové kódy jsou volně k dispozici *(open-source)*. Knihovna poskytuje možnosti pro práci s tenzory, proměnnými nebo gradienty. V roce 2017 byla do knihovny Tensorflow začleněna i knihovna Keras [4], která poskytuje vysokoúrovňovou API pro tvorbu různých architektur CNN, tzv. modelů. Keras také umožňuje nastavení konfigurace jednotlivých vrstev modelů, optimalizátorů, metrik, ztrátové funkce atd.

4.1 Zpracování dat

Před trénováním modelu je nutné korektně připravit data. Dále je nutné určit parametry, kterými jsou například rozlišení obrázků nebo velikost dávky *(batch size)*. Pro načtení datasetu z disku a pro určení parametrů je používána metoda `image_dataset_from_directory()`, která je obsažena v modulu `tf.keras.utils`. Voláním této funkce vznikne objekt `tf.data.Dataset`, který daný dataset reprezentuje. Pro zrychlení tréninku je možné použít různé konfigurace, jako jsou `Dataset.prefetch()` nebo `Dataset.cache()`. Aby bylo možné monitorovat validační ztrátu, je nutné obdobným způsobem načíst i validační dataset. Formát datasetu na disku by měl být následující:



Obrázek 4.1: Příklad formátu uloženého datasetu.

4.2 Vytvoření modelu

K definování architektury modelu se používá tzv. **Functional API** nebo **Sequential API**. Použitím Sequential API vznikne sekvenční model, který musí mít lineární topologii a právě jeden vstup a výstup. Použitím Functional API vznikne funkční model, který umožňuje vytvářet modely s libovolným počtem vstupů a výstupů. Model je charakterizován počtem a typem vrstev, ze kterých se skládá. Mezi nejběžnější patří:

- vstupní vrstva **(Input)** - definuje očekávané vlastnosti vstupních tenzorů,
- konvoluční vrstva **(Conv2D)** - aplikuje konvoluční operace,
- pooling vrstvy **(AveragePooling, MaxPooling)** - zmenšují prostorové rozlišení dat a snaží se zachovat důležité informace,
- dropout vrstva **(Dropout)** - náhodně vyřadí část neuronů plně propojené vrstvy při trénování,
- plně propojené vrstva **(Dense)** - klasifikuje data do tříd,
- **batch normalization** vrstva - normalizuje data před vstupem do jiných vrstev,
- a **flatten** vrstva - převádí tenzor z původního formátu na 1D pole.

Po definování jednotlivých vrstev je funkční model vytvořen instanciací `Model` objektu, který je obsažen v modulu `tf.keras.models`.

```
def get_cnn_model():
    inputs = tf.keras.layers.Input(shape=(28, 28, 3)

    # Convolutional layers
    x = tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu')(inputs)
    x = tf.keras.layers.MaxPooling2D()(x)
    x = tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu')(x)
    x = tf.keras.layers.MaxPooling2D()(x)

    # Flatten and dense layers
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(units=128, activation='relu')(x)
    outputs = tf.keras.layers.Dense(units=82, activation='softmax')(x)

    # Define functional model
    model = tf.keras.models.Model(inputs=inputs, outputs=outputs)

    return model
```

Výpis 4.1: Příklad definice jednoduché konvoluční neuronové sítě. CNN obsahuje vstupní vrstvu, dvě konvoluční vrstvy, dvě pooling vrstvy, flatten vrstvu a dvě plně propojené vrstvy. Jako aktivační funkce je zde použita ReLU a pro klasifikaci softmax.

Na vytvořeném modelu lze zavolat funkce `summary()`. Ta zobrazí dodatečné informace, mezi které patří například: počet trénovatelných a netrénovatelných parametrů jednotlivých vrstev a celého modelu nebo formát výstupního tenzoru z vrstev.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param
=================================================================
 input_1 (InputLayer)         [(None, 28, 28, 3)]       0
 conv2d (Conv2D)              (None, 26, 26, 32)        896
 max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
 conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
 max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 64)        0
 flatten (Flatten)            (None, 1600)              0
 dense (Dense)                (None, 128)               204928
 dense_1 (Dense)              (None, 82)                10578

=================================================================
Total params: 234,898
Trainable params: 234,898
Non-trainable params: 0
_________________________________________________________________
```

Výpis 4.2: Výstup volání metody summary() na modelu definovaném v 4.1.

4.3 Trénování modelu

Aby byl model správně nakonfigurovaný k tréninku, je potřeba na modelu zavolat funkci `compile()`. Argumenty této funkce definují jaký optimalizátor, ztrátová funkce a metriky budou použity při tréninku.

Trénování modelu se spustí funkcí `fit()` zavolané na definovaném modelu. V podobě argumentů jsou této funkci předány trénovací data, počet epoch, velikost dávky, validační data a další. Funkce, po každé epoše, vypisuje na výstup zvolené metriky, zadané při konfiguraci modelu. Metriky slouží ke sledování tréninku a na základě jejich hodnot lze získat aktuální informace o stavu sítě (např. validační chybu). Z těchto poznatků lze zjistit, jestli se síť trénuje adekvátně nebo jestli je potřeba proces trénování upravit. Tensorflow umožňuje trénovací cyklus do jisté míry upravit pomocí parametru `callbacks`. Tento parametr obsahuje seznam funkcí, které se mají po dokončení epochy vykonat. Tímto způsobem lze například měnit hodnotu learning rate zvoleného optimalizátoru či předčasně zastavit trénování modelu.

Každou epochou jsou modelu poskytnuta všechna data z datasetu. Jednotlivá data vstupují postupně do sítě a jakmile počet vstoupených dat dosáhne určené velikosti dávky, dojde k vykonání algoritmu zpětné propagace (sekce 2.3.3). Tento cyklus se opakuje a jakmile všechna data projdou sítí, ukončí se epocha. Na konci epochy jsou provedeny predikce na validačním modelu a je vypočtena validační ztráta.

Natrénovaný model se následně může zhodnotit na testovací sadě[1] pomocí metody `evaluate()`, která predikuje třídy a vypočítá metriky pro testovací sadu. Pokud by byl model nasazen v praxi, například v rámci aplikace, je vypočtení metrik zbytečné a proto se pro predikci jednoho či více vzorků se využívá funkce `predict()`.

[1] Datová sada, která nebyla použita při tréninku.

Kapitola 5

Návrh experimentů

Při návrhu modelu by bylo vhodné použít již existující architekturu CNN, jelikož sítě v dnešní době dosahují již kvalitních výsledků v oblasti detekce jevů v obraze. Vytvoření vlastní architektury by zabralo mnoho času a dost pravděpodobně by nepřekonala výkonnost dnešních sítí. Po konzultaci s vedoucím práce jsem vybral architekturu ResNet. Konkrétně ResNet50v2, jelikož dosahuje lepších výsledků oproti originální architektuře.

Síť ResNet50v2 byla trénována pro klasifikaci do 1000 tříd a je nutné ji tedy upravit, pro potřeby datasetu, ke klasifikaci do 82 tříd. Dále tato architektura obsahuje 25,6 miliónů parametrů, z nichž je zhruba 23 miliónů trénovatelných. Jedná se tedy o relativně velkou síť a bylo by vhodné použít předem definované váhy, z důvodu úspory času a výpočetního výkonu při trénování. Váhy byly načteny ze soutěže ILSVRC, které se tato architektura účastnila, a které jsou již kvalitně přizpůsobeny k detekci jevů v obraze.

Při zkoumání výsledků experimentů je nutné tyto výsledky porovnávat a zkoumat. Konkrétně jestli došlo ke zlepšení přesnosti predikce a generalizace modelu, či nikoliv. Pro porovnání s experimenty by bylo vhodné natrénovat model (dále základní model), kde architektura sítě bude téměř nezměněna, stejně jako vstupní data a konfigurace sítě.

5.1 Doladění konfigurace sítě

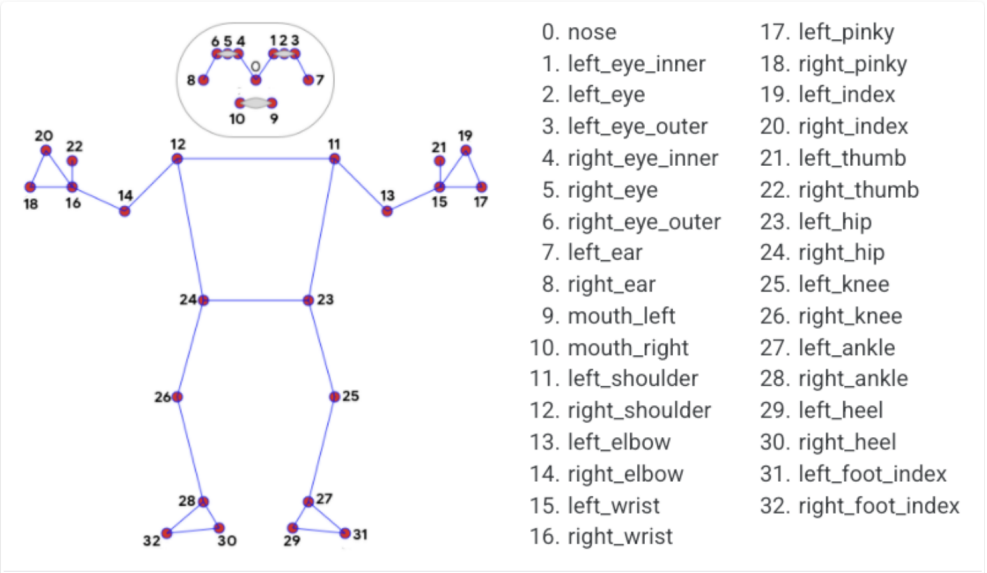
Při změně datasetu dojde i ke změně hodnot ztrátové funkce při trénování. Proto byly zkoušeny různé optimalizátory a learning rate hodnoty a byl zkoumán jejich vliv na generalizaci modelu. ResNet50v2 byl trénován na datasetu obsahujícím ~1,4 miliónů obrázků. Nejednalo se ale ve většině případů o detekci lidské postavy, proto byly vyzkoušeny různé augmentace dat datasetu Yoga-82 a jejich vliv na přesnost predikce. Každý obrázek byl pětkrát zkopírován. Výsledky experimentů jsou poté porovnávány s takto natrénovaným modelem.

5.2 Detekce klíčových bodů

Jako hlavní návrh pro experimenty mě napadlo využít extrakce klíčových bodů z obrazu a využít těchto informací k tréninku a možnému vylepšení detekce jógových pozic. Konkrétně se jedná o nalezení částí lidského těla v obraze. Hledat a anotovat přes 16 tisíc obrázků ručně je zbytečně zdlouhavé a náročné. Existuje ale několik prostředků pro získání klíčových bodů. Jedním z nich je framework MediaPipe.

MediaPipe

MediaPipe [1] je multiplatformní a open-source framework vyvíjený společností Google. Framework obsahuje předem definované modely pro rozpoznání objektů v obraze. Modely umožňují rozpoznávání ručních gest, detekci klíčových bodů obličeje, detekci objektů a v neposlední řadě i detekci lidské pózy. MediaPipe dokáže detekovat až 33 bodů lidského těla. I když je detekce prováděna na obrázcích, dokáže MediaPipe určit souřadnice (x, y, z) jednotlivých bodů ve 3D prostoru. Souřadnici hloubky odhaduje z naučeného modelu. Lze tak i kostru člověka vykreslit v prostoru 3D. Dále MediaPipe uvádí u bodů i jejich viditelnost na obrázku.



Obrázek 5.1: Ukázka 33. klíčových bodů, které dokáže MediaPipe detekovat. Vpravo je název klíčového bodu korespondujícího s číslem na kostře. Převzato z [1].

5.2.1 Využití klíčových bodů

MediaPipe ukládá všechny nalezené body a jejich souřadnice. Souřadnic lze využít pro nakreslení klíčových bodů do obrázku. Body pak lze propojit a vytvořit tak kostru člověka. Kostra člověka je v experimentech použita více způsoby:

1. Kostra je vložena přímo do obrázku.
2. Kostra je uložena jako separátní obrázek, který má černé pozadí.
3. Kostra je uložena jako separátní obrázek, který má černé pozadí a jednotlivé části lidského těla byly obarveny rozdílnými barvami. Přidání této barevné informace pro CNN by mohlo přinést zlepšení generalizace.

Nově vytvořené datasety z obrázků byly poté použity na vstupy upravených modelů.

5.3 Úpravy základního modelu

V závislosti na sekci 5.2.1 byl model pro jednotlivé experimenty modifikován.

Nejdříve byl model upraven tak, aby byl schopný přijímat dva obrázky při tréninku. Model se tedy rozdělil na pravou a levou větev. Pro levou větev byla použita architektura ResNet50v2. Pro pravou větev byla použita modifikovaná architektura sítě AlexNet. Výstupy levé a pravé větve byly následně spojeny a napojeny na plně propojené vrstvy. Poté byly zkoumány vlivy těchto úprav modelu na generalizaci modelu a porovnány se základním modelem.

Další experiment se také věnoval úpravě modelu, který by byl schopen zpracovávat dva obrázky paralelně. Nyní ale bylo využito různých částí architektury ResNet. Konkrétně jsem rozdělil ResNet architekturu na dva úseky v různých hloubkách sítě mezi bottleneck residuálními bloky. První úsek sítě byl poté zkopírována a použit ke vstupu pro druhý obrázek. Výstupy obou částí byly následně propojeny a napojeny na zbytek architektury.

Detailnější popis a konkrétní architektury modelů jsou zobrazeny v sekcích 7.3 a 7.2.

Kapitola 6

Trénink základního modelu

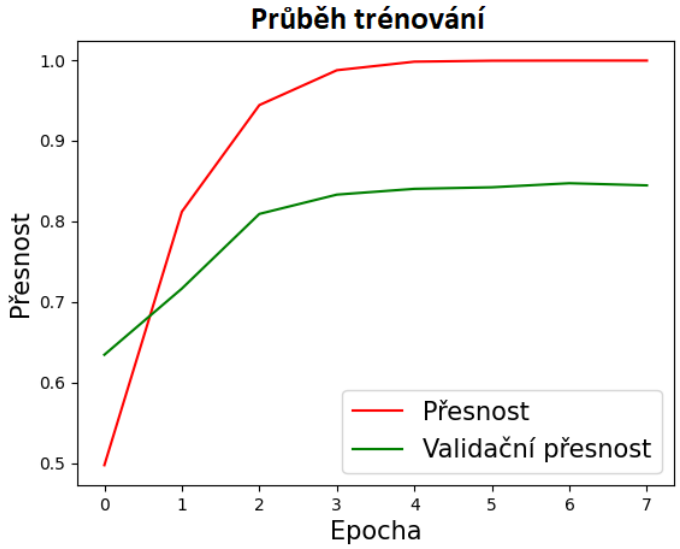
Tato kapitola se zabývá úpravou architektury ResNet50v2 pro detekci 82 tříd. Vstupní data nejsou nijak modifikována. Jako optimalizátor je použit Adam s learning rate hodnotou 0,001, která je výchozí hodnotou nastavenou knihovnou TensorFlow. Váhy byly přednastaveny hodnotami ze soutěže ILSVRC.

Dataset byl načten metodou `image_dataset_from_directory()`, která je popsána v sekci 4.1. Data byla načtena v rozlišení 224×224 a velikostí dávky 32. Stejným způsobem byl načten i validační dataset. Dále musel být model upraven pro klasifikaci do 82 tříd, kvůli použití datasetu Yoga-82. Toho bylo dosaženo pomocí Functional API knihovny Tensorflow.

```
resnet = tf.keras.applications.ResNet50V2(
    include_top=False,
    weights='imagenet',
    input_shape=(224, 224, 3),
    pooling='avg',
)

x = tf.keras.layers.Dense(units=82, activation='softmax', name='predictions')(resnet.output
    )

model = tf.keras.models.Model(inputs=resnet.input, outputs=x)

model.compile(optimizer='Adam', loss="categorical_crossentropy", metrics=['accuracy'])
```

Výpis 6.1: Úkázka vytvoření základního modelu, pomocí knihovny TensorFlow, pro klasifikaci 82 tříd datasetu Yoga-82.

Jako ztrátová funkce byla použita categorical cross-entropy. Předčasné zastavení bylo nastaveno s trpělivostí 3 a monitorováním validační ztráty. Trénink byl spuštěn na 50 epoch. Byl ale předčasně zastaven po dokončení sedmé epochy. Přesnost predikce na trénovaném datasetu dosáhla 100 %. Validační přesnost dosáhla 84,462 %. Došlo tedy k přetrénování modelu.



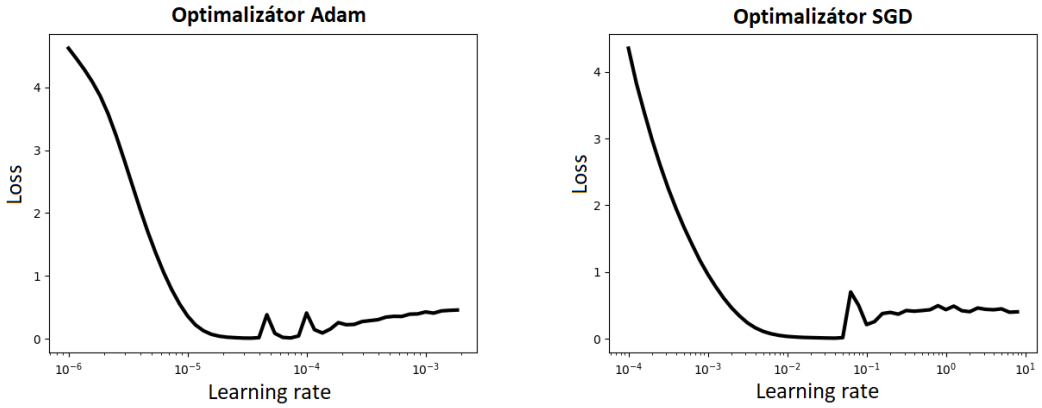
Obrázek 6.1: Průběh tréninku základního modelu. Červená křivka značí přesnost predikce na trénovacím datasetu a zelená na validačním datasetu.

6.1 Hledání ideálního optimalizátoru a hodnoty learning rate

Jedním ze způsobů nalezení ideální hodnoty learning rate je náhodné manuální zkoušení různých hodnot. Nakonec se vybere ta hodnota, po které síť dosahuje nejlepších výsledků. To je ale velmi časově a výpočetně náročné.

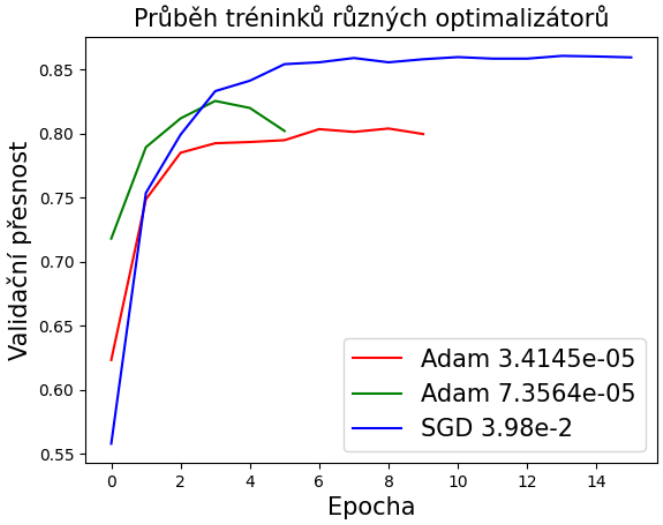
Další metodou je exponencionálně zvyšovat learning rate na konci každé epochy a sledovat hodnotu ztrátové funkce. Tímto způsobem lze zjistit chování sítě při tréninku pro různé learning rate hodnoty. Learning rate je poté vybrán v moment, kdy je hodnota ztrátové funkce nejnižší. Jako optimalizátory jsem vybral jedny z nejpoužívanějších optimalizátorů a to optimalizátory Adam a SGD.

Pro zmíněné optimalizátory umožňuje Tensorflow hledání hodnoty learning rate, a to pomocí parametru `callbacks` funkce `fit()`. Pro exponenciální zvyšování hodnoty learning rate je využit objekt `LearningRateScheduler`, který je obsaženu v modulu `tf.keras.callbacks`. Pro optimalizátor Adam byly vyzkoušeny hodnoty v rozmezí $10^{-6} - 2,1 * 10^{-3}$ a pro SGD hodnoty v rozmezí $10^{-4} - 10$. Rozmezí bylo vyzkoušeno více, ale tyto dosáhly nejnižší hodnoty ztrátové funkce.



Obrázek 6.2: Hledání ideální hodnoty learning rate. Vpravo pro optimalizátor Adam. Vlevo pro SGD optimalizátor.

Pro optimalizátor Adam byly nalezeny hodnoty $3,414 * 10^{-5}$ a $7,356 * 10^{-5}$. Pro SGD hodnota $3,98 * 10^{-2}$. Hodnoty byly následně vyzkoušeny při tréninku modelu. Při tréninku dosáhl nejlepší generalizace optimalizátor SGD. Všechny tréninky byly předčasně zastaveny. Nejvyšší generalizace dosáhl SGD s validační přesností predikce 86,01 %.



Obrázek 6.3: Porovnání tréninků optimalizátorů s nalezenými hodnotami velikosti učení.

6.2 Augmentace dat

Následující sekce se bude zabývat technikami augmentace vstupních dat. Veškeré augmentace jsou prováděny pomocí modulů `tf.image` a `tfa.image`[1]. Augmentační techniky byly prováděny nastavením argumentu **seed** jednotlivých transformačních funkcí. Tento argument zajistí, že při každém spuštění jsou provedeny stejné pseudo-náhodné augmentace. Důvodem byla reprodukovatelnost těchto transformací pro různé síly augmentací, se kterými jsem

[1]Jedná se o repositář s dalšími funkcemi pro augmentaci obrázků.

experimentoval. Pokud byla například rotace pro jeden trénink nastavena na 5 % a u druhého tréninku 15 %, obrázek se otočil stejným směrem, akorát o jiný úhel. Tímto jsem se snažil zajistit, aby mohly být výsledky tréninků co nejlépe porovnatelné.

Experimentoval jsem s následnými technikami transformace: pseudo-náhodné horizontální a vertikální převrácení, rotace a se změnou barev obrázku (kontrast, saturace a odstín). Záměrně jsem nepoužil techniky jako posun obrázku nebo oříznutí. Při takových augmentacích by mohlo dojít ke smazání klíčových informací obrázku, které jsou pro detekci jógových kritické.

Rotace

Experimenty s rotací probíhaly v rozmezí $[5°, 45°]$ a otočení bylo provedeno pseudo-náhodně v obou směrech. Při testování jsem zjistil, že nejlepší hodnota pro otočení je v rozmezí $[-15°, 15°]$.

Horizontální a vertikální převrácení

Horizontální převrácení probíhalo s pravděpodobnostmi v rozmezí 10 % až 80 %. Stejné pravděpodobnosti byly nastaveny i pro vertikální převrácení obrázku. Experimenty s hodnotami přineslo vždy téměř identické výsledky přesnosti predikce modelu a pravděpodobnost aplikování těchto dvou transformací jsem ponechal na 50 %.

Kontrast

Funkce `random_contrast()` obsahuje dva argumenty, horní a dolní mez kontrastního faktoru. Pro každý obrázek je poté vybrán náhodný faktor kontrastu ze zadané meze a výpočet nové hodnoty pixelu je proveden následovně:

$$new\_pixel\_value = (pixel\_value - mean) * contrast\_factor + mean \tag{6.1}$$

V rovnici 6.1 značí *pixel_value* hodnotu vstupního pixelu v rozmezí $[0, 255]$, *contrast_factor* pseudo-náhodně vygenerovanou hodnotu z dané meze, *mean* průměrnou hodnotu všech pixelů v obrázku dané barvy a *new_pixel_value* nově vypočtenou hodnotu pixelu.

Při testování jsem vyzkoušel hodnoty dolní hranice $0,3 - 0,9$ a horní hranice $1,1 - 2$. Při vyšších hodnotách horní hranice byl obrázek velmi světlý a nečitelný, při nižších hodnotách dolní hranice zase moc tmavý a proto nebyly testovány. Nejlepšího výsledku dosáhlo nastavení kontrastního faktoru 0,7 pro dolní hranici a 1,5 pro horní hranici.

Saturace

Ke změně saturace je nejdříve funkcí `random_saturation()` obrázek převeden do HSV (Hue, Saturation, Value) barevného modelu. Následně je vybrána hodnota saturačního faktoru ze zadaného rozmezí (obdobně jako u funkce *contrast_factor()*) a dojde k vynásobení pixelů saturačního kanálu touto hodnotou. Nakonec je obrázek převeden zpět do RGB barevného modelu. Nejideálnější hodnota pro dolní mez byla 0,7 a pro horní mez 1,2.

Odstín

Změna odstínu (*Hue*) je provedena funkcí `random_hue()`, která převede obrázek do HSV barevného modelu obdobně jako funkce `random_saturation()`. Jako argument se udává

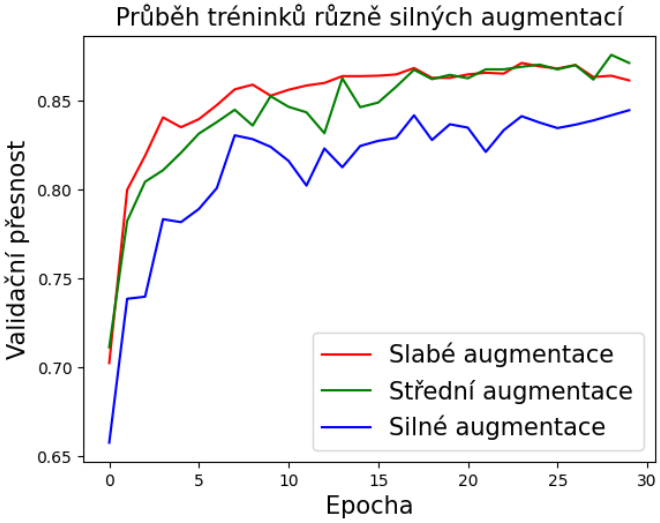
delta, která se poté generuje náhodnou hodnotu v rozmezí $[-delta, delta]$. Vygenerovanou hodnotou je poté Hue kanál obrázku upraven a následně je obrázek převeden zpět to RGB barevného modelu.

Při testování této augmentace jsem pracoval s hodnotami parametru delta v rozmezí $0,1 - 0,5$, kde nejlepšího výsledku bylo dosaženo pro hodnotu 0,3.

6.2.1 Různé síly augmentací a jejich ovlivnění generalizace modelu

Experimentování s augmentačními technikami proběhlo s různě silnými augmentacemi. Nejdříve jsem experimentoval s velmi slabými augmentacemi a postupně tyto transformace zesiloval.

Model byl následně trénován na různých silách augmentace s optimalizátorem SGD a learning rate hodnotou 0,0398. Na obrázku 6.4 je ukázáno porovnání některých, mnou vyzkoušených, sil augmentací. Model dosáhl při tréninku v nejlepším případě přesnosti validační predikce 87,12 %.



Obrázek 6.4: Trénování modelu s různou silou augmentací vstupních dat. Červená křivka značí slabší augmentace, zelené středně silné augmentace a modrá silné augmentace.

Kombinací vhodně použitého optimalizátoru a použití adekvátních augmentací byla přesnost predikce vylepšena o ~2,65 %, v porovnání se základním modelem. Došlo i ke zmírnění přetrénování sítě.

Kapitola 7

Experimenty s využitím klíčových bodů a modifikací modelu

Tato kapitola se zaměřuje na využití klíčových bodů v obraze použitím frameworku MediaPipe. První sekce se zabývá úpravou datasetu. Pro úpravu datasetu jsem vytvořil skript, který vykreslí kostru člověka do původních obrázků datasetu Yoga-82. Obrázky jsou poté využity k trénování základního modelu a je porovnán s předchozími výsledky.

Další sekce ze zabývá vytvořením dvou nových datasetů. První dataset obsahuje kostru vykreslenou na černém pozadí. V druhém datasetu byla kostra, nalezena frameworkem MediaPipe, obarvena různými barvami a bylo jí dáno černé pozadí (obdobně jako u prvního datasetu). Dále jsem upravil model AlexNet pro klasifikaci těchto dvou datasetů. Tento model byl poté připojen k výstupu architektury ResNet50v2. Následně byl zkoumán vliv obarvené a neobarvené kostry na validační přesnost predikce modelu. Ten byl porovnán se základním modelem.

V poslední sekci experimentuji s kopírováním různých částí architektury ResNet50v2, které jsou poté využity pro vstup druhého obrázku z datasetu, kde jsou části kostry obarveny rozdílnými barvami.

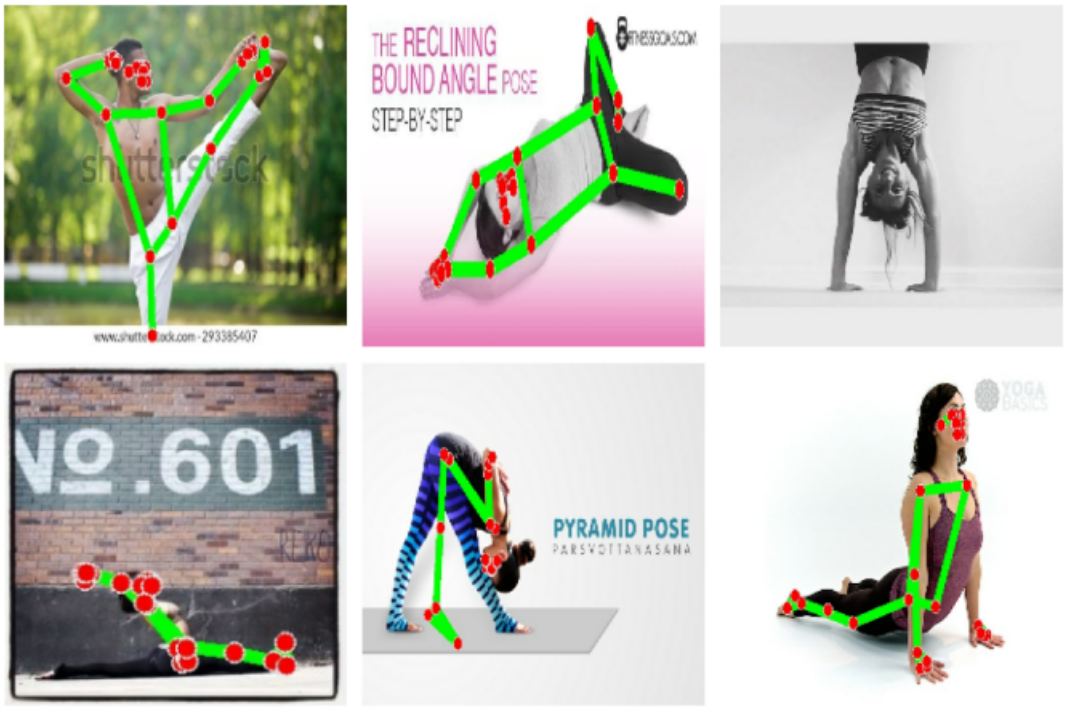
7.1 Trénování základního modelu na datech s vyobrazenou kostrou

Experiment se zabývá vytvořením nového datasetu z původního datasetu Yoga-82, kde dojde ke přidání kostry do obrázků. Následně je nalezena optimální hodnota learning rate a model je poté na nově vzniklé datové sadě trénován.

7.1.1 Vytvoření nového datasetu

Pro nalezení kostry člověka jsem použil framework MediaPipe. Ten disponuje předtrénovaným modelem, který detekuje klíčové body lidského těla. Model je obsažen v modulu `mediapipe.solutions.pose`. Obrázky byly načítány postupně ze složek datasetu funkcí `imread()`, kterou disponuje knihovna OpenCV. Jelikož tato funkce načítá obrázky v barevném modelu BGR, byly obrázky převedeny do modelu RGB. Na každém obrázku byly následně určeny klíčové body pomocí předtrénovaného modelu MediaPipe a jejich hodnoty uloženy. Vykreslení kostry proběhlo použitím modulu `mediapipe.solutions.drawing_utils`. Klíčové body jsou vykresleny červenou barvou. Propojení klíčových bodů zelenou barvou.

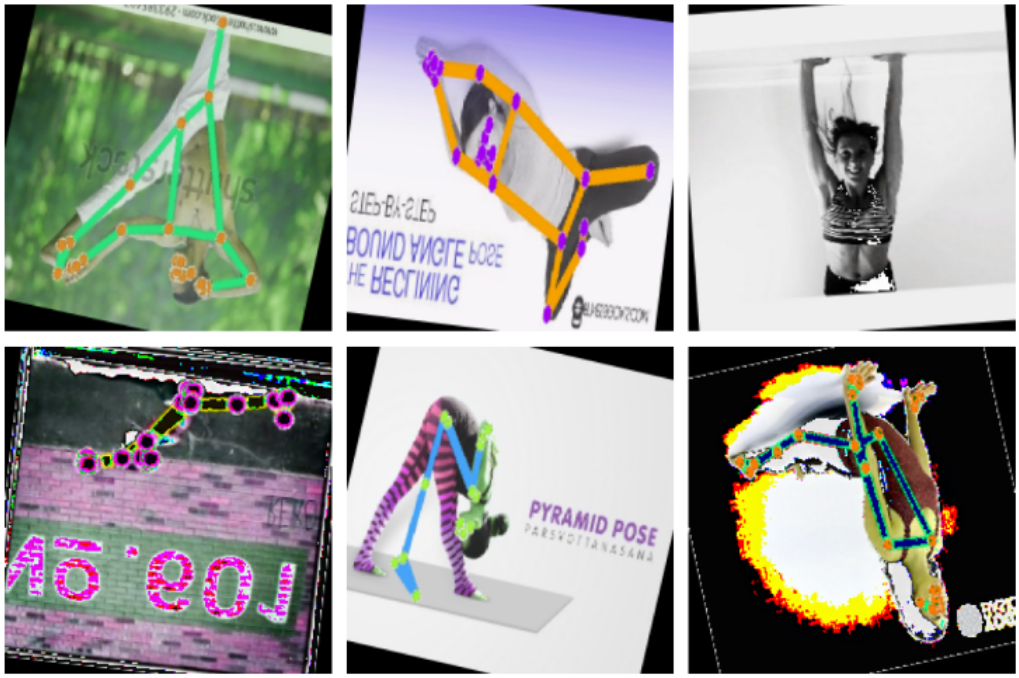
Pokud MediaPipe klíčové body nenalezl, obrázek byl uložen do nového datasetu v originální podobě.



Obrázek 7.1: Ukázka některých obrázků datasetu. Obrázek vpravo nahoře neobsahuje kostru, jelikož klíčové body nebyly nalezeny. Z obrázků lze i vidět, že MediaPipe nedokáže určit klíčové body se 100% přesností.

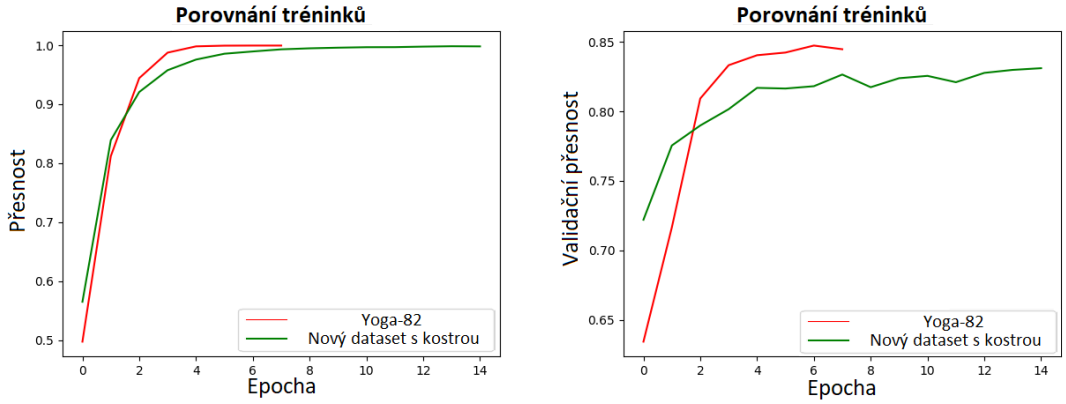
7.1.2 Trénování základního modelu na nově vzniklém datasetu

Trénování probíhalo velmi podobným způsobem jako v kapitole 6. Nejideálnější nalezená learning rate hodnota byla $5,4 * 10^{-3}$. Ostatní parametry pro trénink byly nastaveny identicky.



Obrázek 7.2: Ukázka některých augmentovaných obrázků nově vytvořeného datasetu, které byly použity při experimentu.

Trénink byl předčasně zastaven po dokončení 16. epochy. Model dosáhl přesnosti 83,1 % validační predikce. Jedná se tedy o ~4% zhoršení v porovnání se základním modelem. Domnívám se, že ke zhoršení došlo kvůli zakrytí velké části lidské pózy, proto jsem dále experimentoval i s tloušťkami vykreslených čar. Zde model vykazoval velmi podobné přesnosti predikce jako základní model. Příliš tenké čáry byly pravděpodobně modelem ignorovány a proto nejsou tyto experimenty uvedeny v této práci.



Obrázek 7.3: Porovnání tréninku základního modelu na originálním (červená křivka) a nově vytvořeném datasetu (zelená křivka). Vlevo jsou zobrazeny trénovací přesnosti predikce. Vpravo validační přesnosti predikce.

7.2 Využití dvou obrázků k trénování modelu

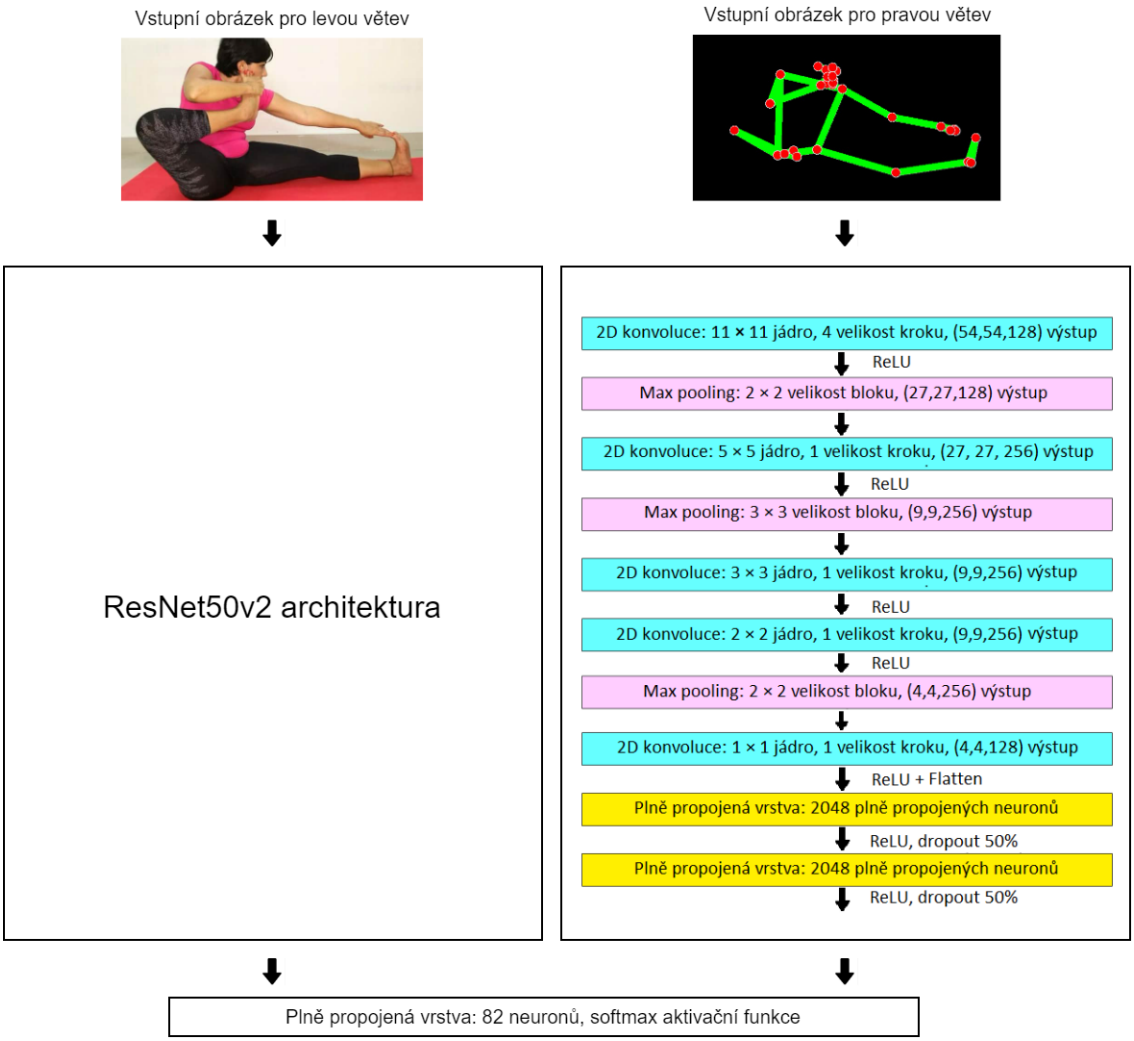
V předešlém experimentu (7.1) docházelo k zakrytí lidské pózy nalezenými klíčovými body. Bylo by tedy vhodné využít klíčové body k tréninku sítě jiným způsobem. Jako vhodné řešení tohoto problému mě napadlo vytvořit další dataset a upravit model tak, aby dokázal zpracovávat dva obrázky současně.

Tvorba dalšího datasetu proběhla velmi podobným způsobem, jako v minulém experimentu. A to využitím frameworku MediaPipe. Kostra ale nebyla nakreslena do původního obrázku. Pomocí knihovny NumPy [7], konkrétně použitím funkce `numpy.zeros()`, bylo vytvořeno vícerozměrné pole o rozměru $224 \times 224 \times 3$. Všechny hodnoty pole jsou funkcí nastaveny na hodnotu 0. Následně byla nalezená kostra vložena do pole, které bylo poté uloženo jako obrázek. Pokud kostra nebyla nalezena, pole se uložilo jako černý obrázek, jelikož všechny hodnoty byly nastaveny na hodnotu 0.

7.2.1 Definování modelu se dvěma vstupy

Při definování modelu byl model rozdělen na dvě paralelní větve: levou větev a pravou větev. Levá větev slouží ke zpracování obrázků datasetu Yoga-82. Pro levou větev byla využita ResNet50v2 architektura. Pravá větev slouží ke zpracování obrázků s vykreslenou kostrou na černém pozadí. Pro pravou větev byla využita upravená architektura AlexNet. Úpravy spočívaly ve změnách konkrétních parametrů jednotlivých vrstev sítě a přidáním či odebíráním vrstev. Hledání ideálního nastavení vrstev probíhalo trénováním této větve na obrázcích.

Výstupy posledních vrstev větví byly následně spojeny. Toho bylo dosaženo pomocí vrstvy `Concatenate`, která je obsažena v modulu `tf.keras.layers`. Tato vrstva byla poté napojena na finální vrstvu pro klasifikaci do 82 tříd.



Obrázek 7.4: Ukázka modelu a jeho větví. Levá větev slouží pro zpracování obrázků z datasetu Yoga-82. Pravá větev slouží ke zpracování obrázků nově vytvořeného datasetu a jsou zde vyznačeny jednotlivé vrstvy.

Trénování modelu se dvěma vstupy

Nejdříve byla nalezena learning rate hodnota $1,36 * 10^{-2}$. Jako optimalizátor byl použit SGD optimalizátor. Datasety byly načteny funkcí `image_dataset_from_directory()` a následně propojeny funkcí `tf.dataset.Dataset.zip()` do formátu vyžadovaného modelem. Augmentace obrázků proběhla pouze u datasetu určeného pro levou větev.

Trénování sítě nepřineslo žádné závratné výsledky. Validační přesnost predikce modelu byla ~60 %. Jedná se tedy o výrazné zhoršení oproti základnímu modelu. Důvodem zhoršení je malé využití informací klíčových bodů. Tím je myšleno, že jednotlivé části kostry mají stejnou barvu, stejně jako klíčové body.

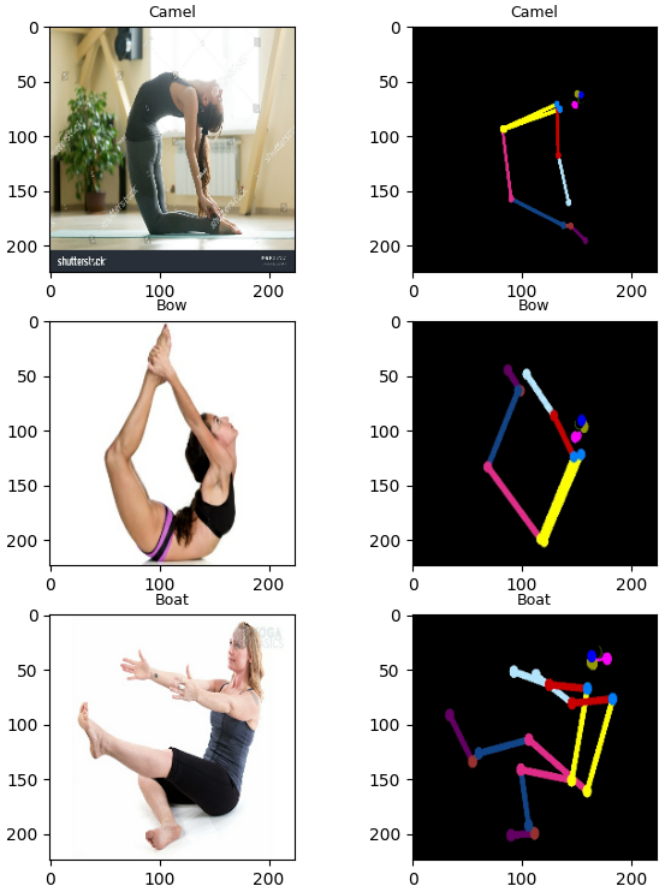
7.2.2 Obarvení kostry

Při obarvení částí kostry a klíčových bodů, které reprezentují jednotlivé části lidského těla, dojde k přidání dalších informací. Každá barva v obrázku je totiž reprezentována jinými hodnotami pixelů. Model má tudíž k dispozici rozsáhlejší množství dat, ze kterých může při jeho trénování čerpat.

MediaPipe neposkytuje funkce k obarvení jednotlivých částí různými barvami. Proto bylo znovu využito knihovny OpenCV. Pro každý klíčový bod jsem vybral barvu. Barvy pro levé a pravé části těla jsou shodné (např. levý a pravý loket bude znázorněn stejnou barvou). Důvodem je, že obrázky jsou foceny z různých úhlů a většina jógových póz vypadá z obou stran stejně.

Pro vykreslení klíčových bodů byla použita funkce `cv2.circle()`, která vkreslí do obrázku kruh. Pozice kruhu je určena souřadnicemi nalezeného klíčového bodu. MediaPipe udává i parametr viditelnosti každého klíčového bodu. Pokud je viditelnost klíčového bodu malá, bod není vykreslen.

K vykreslení kostry byla použita funkce `cv2.line()`. Funkce bere jako parametry počáteční a cílový bod, mezi kterými vykreslí obarvenou čáru. Pro vykreslení čáry, která reprezentuje danou končetinu, jsou vždy vybrány dva korespondující klíčové body (např. levý kotník a koleno).

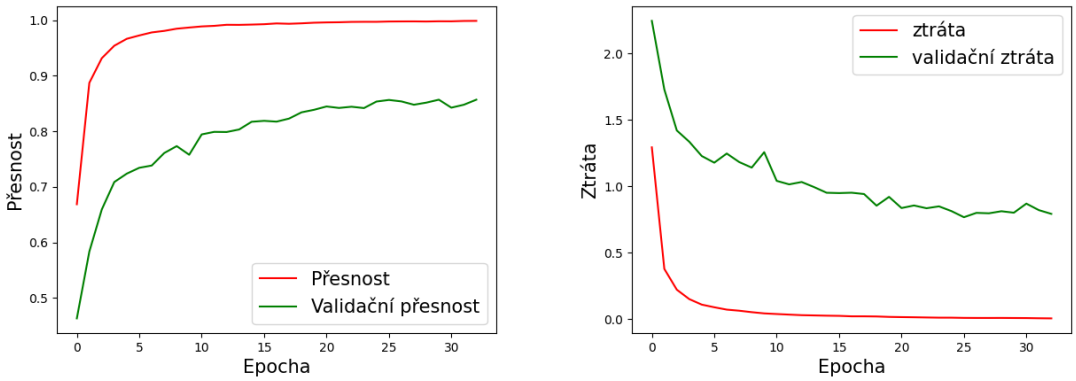


Obrázek 7.5: Ukázka korespondujících obrázků obou datasetů. Text nad obrázkem značí název třídy. Číselné hodnoty značí rozlišení obrázku v pixelech. Vlevo: obrázky z datasetu Yoga-82. Vpravo: obrázky z nově vytvořeného datasetu s obarvenou kostrou.

7.2.3 Trénování modelu na datasetu s obarvenou kostrou

Jako model byl použit model definovaný v sekci 7.2.1. Levá větev modelu zpracovává augmentované obrázky datasetu Yoga-82. Pravá větev zpracovává obrázky s obarvenou kostrou. Původní plán bylo využít identické augmentace pro korespondující obrázky obou datasetů. Obrázky určené pro pravou větev ale nakonec augmentovány nebyly. Nepovedlo se mi zajistit použití identických augmentací pro oba datasety (např. otočit oba obrázky stejným směrem), přestože byl u jednotlivých augmentačních technik nastaven parametr `seed`.

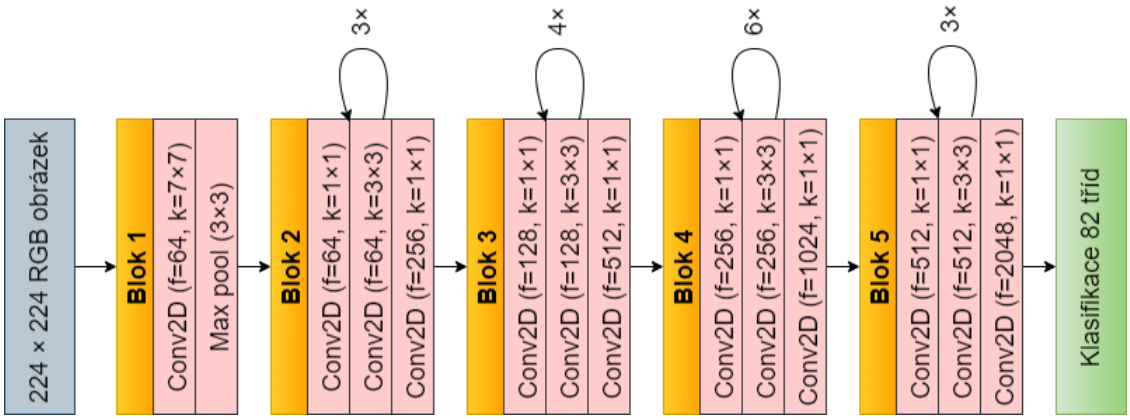
Trénink byl po 33. epoše předčasně zastaven z důvodu zvyšování validační ztráty. Validační přesnost predikce modelu byla 85,68 %. Oproti tréninku na datasetu s jednobarevnou kostrou došlo tedy k výraznému zlepšení. Jiný způsob využití klíčové informace tedy pomohl modelu lépe generalizovat a dosáhnout vyšší přesnosti predikce. Validační přesnost predikce je ale stále o ~2 % nižší v porovnání se základním modelem. Testováním jsem zjistil, že predikce dvojice obrázků, kde na druhém obrázku nebyla nalezena kostra, byla více chybná, než u dvojice s nalezenou kostrou. Domnívám se, že kdyby MediaPipe nalezl klíčové body na všech obrázcích, byla by validační přesnost predikce základního modelu překonána.



Obrázek 7.6: Ukázka průběhu trénování modelu na datasetu s obarvenou kostrou. Vlevo: zobrazení přesností predikce na trénovacím a validačním datasetu. Vpravo: zobrazení ztráty a validační ztráty na trénovacím a validačním datasetu.

7.3 Modifikace architektury ResNet50v2

V sekci 2.2.2 je zmíněno, že architektura ResNet50v2 je založena na bottleneck residuálních blocích. Tyto bloky obsahují vždy tři konvoluční vrstvy. V celé sítí jsou bottleneck residuální bloky, které mají konvoluční vrstvy identické (opakují se). Lze díky tomu rozdělit architekturu do sedmi částí.



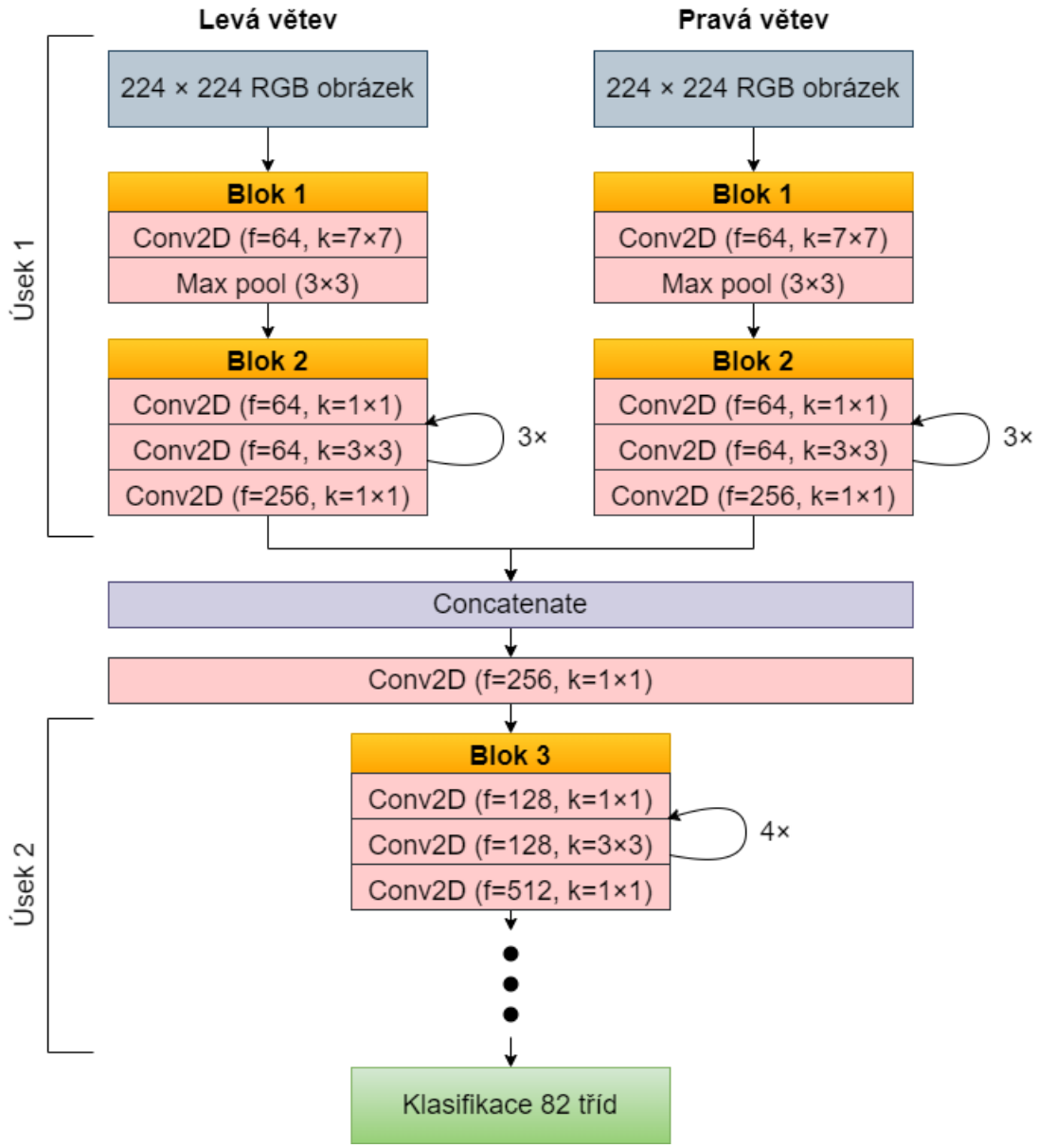
Obrázek 7.7: Zjednodušená architektura ResNet50v2, která je rozdělená do sedmi částí. f značí počet filtrů a tím i hloubku výstupního tenzoru. k značí velikost konvolučního jádra.

7.3.1 Úprava architektury ResNet50v2 pro zpracování dvou vstupů

V obrázku 7.7 je architektura rozdělena na 7 částí. Při tomto experimentu vzniklo pět modelů. Každý z modelů umožňuje vstup dvou obrázků a rozděluje se na dvě větve.

Každý model se liší svojí architekturou. Každá větev obsahuje určitý počet částí, které vznikly rozdělením architektury ResNet50v2. Například u prvního modelu obsahuje levá a pravá větev první dvě části (vstupní vrstvu a blok 1), u druhého modelu obsahují větvě 3 části (vstupní vrstvu, blok 1 a blok 2) atd.

Vytváření druhého modelu proběhlo pomocí Functional API. Nejdříve byla načtena celá architektura ResNet50v2, kde každá vrstva měla předem definované váhy. Následně byla architektura rozdělena na dva úseky: vstup - blok 2, blok 3 - finální vrstva. První úsek byl použit pro levou větev. Levá větev byla poté zkopírována a tím vznikla pravá větev. Výstupy obou větví byly spojeny vrstvou `Concatenate`. Výstupní tenzor z této vrstvy má rozměry $56 \times 56 \times 512$. Vstupní vrstva druhého úseku ale vyžaduje rozměry $56 \times 56 \times 256$. Proto byla použita vrstva `Conv2D`, které byl nastaven poloviční počet filtrů a velikost jádra byla určena 1×1. Přidáním této vrstvy došlo ke zmenšení hloubky tensoru na požadovaný rozměr. Poté byl na tuto konvoluční vrstvu připojen druhý úsek. Nakonec byla připojena plně propojená vrstva s aktivační funkcí softmax pro klasifikaci do 82 tříd.



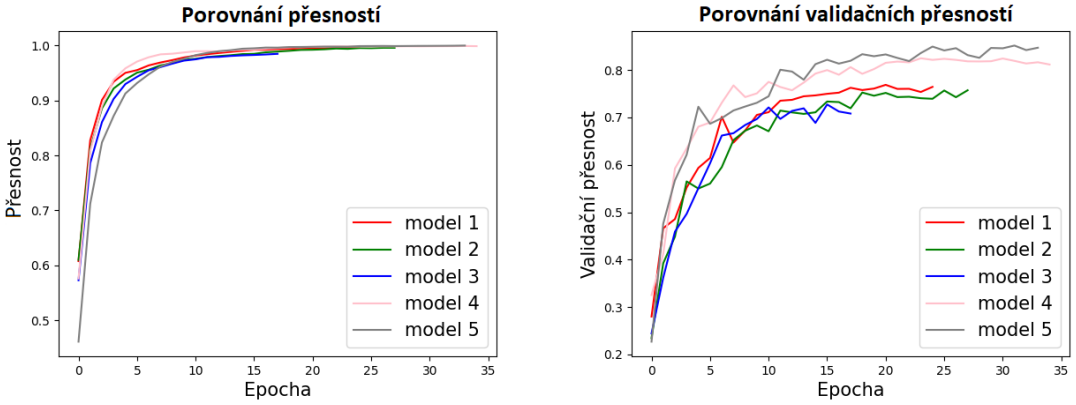
Obrázek 7.8: Architektura druhého modelu. f značí počet filtrů a tím i hloubku výstupního tenzoru. k značí velikost konvolučního jádra.

Obdobným způsobem byly vytvořeny i zbylé čtyři modely, kde architektura byla rozdělena mezi různými částmi. Modely vypadají podobně jako na obrázku 7.8, jsou jen spojeny mezi jinými částmi.

7.3.2 Trénování upravené architektury ResNet50v2

Levé větve modelů byly trénovány na datasetu Yoga-82 a data byla augmentována. Pravé větve na datasetu s obarvenou kostrou, jelikož se tento dataset projevil jako vhodnější volba. Tato data nebyla augmentována ze stejného důvodu jako v sekci 7.2.3. Jako optimalizátor byl zvolen SGD. Pro každý model byla nalezena learning rate hodnota metodou exponenciálního zvyšování.

Trénování všech modelů bylo předčasně ukončeno. Nejlepšího výsledku dosáhl pátý model s validační přesností predikce 84,702 %. Ostatní modely dosáhly nižších validačních přesností predikce (viz. 7.9). Oproti základnímu modelu došlo ke zhoršení validační predikce o ~2,4 %.



Obrázek 7.9: Porovnání tréninků všech modelů vzniklých úpravou architektury ResNet50v2.

Kapitola 8

Závěr

Tato práce se zabývala experimenty s využitím klíčových bodů lidského těla a způsoby, jakými lze těchto informací využít pro klasifikaci jógových pozic.

Trénovací a testovací obrázky byly vytvořeny rozdělením datasetu Yoga-82, který byl manuálně přetříděn od nevalidních obrázků a následně rozšířen. Obrázky datasetu byly využity k vytvoření tří nových datasetů. První dataset obsahoval obrázky s vykreslenou kostrou člověka na původních obrázcích datasetu Yoga-82. Vykreslená kostra reprezentuje jednotlivé části lidského těla. Druhý dataset obsahoval obrázky s vykreslenou kostrou na obrázcích s černým pozadím. Obrázky posledního vzniklého datasetu obsahovaly obarvenou kostru vykreslenou na obrázcích s černým pozadím. Kostra byla vytvořena pomocí detekce klíčových bodů lidského těla frameworkem MediaPipe a jejich patřičným propojením.

Jako architektura konvoluční neuronové sítě byla vybrána architektura ResNet50v2. Architektura byla upravena pro klasifikaci 82 tříd a natrénována na datasetu. Dále byla síti nalezena optimální konfigurace a byla přetrénována na augmentovaných datech. Síť dosáhla přesnosti validační predikce 87,12 % a slouží pro porovnání výkonů sítí vzniklých v experimentech.

Experimenty se zabývaly způsoby využití klíčových bodů při trénování rozdílných architektur sítí a jejich dopadem na výsledné predikce. V prvním experimentu bylo využito prvního pomocného datasetu a byla na něm natrénována upravená ResNet50v2 architektura. Experiment dosáhl validační přesnosti 83,1 % a nedošlo k vylepšení stávajícího řešení.

V druhém experimentu byla namodelována síť rozdělena na dvě větve a tím umožňovala zpracovávat dva obrázky paralelně. Pro první větev byla využita architektura ResNet50v2 a zpracovávala obrázky datasetu Yoga-82. Pro druhou větev byla použita modifikovaná architektura AlexNet, která zpracovávala obrázky druhého a třetího pomocného datasetu. Experiment ukázal, že obarvení kostry výrazně pomohlo modelu lépe generalizovat data. Prokázal také, že správné využití informací klíčových bodů vede k lepším výsledkům tréninku. Síť dosáhla přesnosti validační predikce 85,68 %. Nedošlo ale k vylepšení stávajícího řešení.

Poslední experiment ze zabýval rozdělováním architektury ResNet50v2 na dvě části v různých hloubkách sítě. První část byla potom využita k vytvoření druhé větve, které byly následně spojeny a napojeny na druhou část. Tímto způsobem vzniklo pět nových modelů. Levé větve zpracovávaly dataset Yoga-82 a pravé větve zpracovávaly dataset s obarvenou kostrou. Nejlepší model dosáhl validační přesnosti predikce 84,702 %. V experimentu nedošlo ke zlepšení výsledků stávajícího řešení.

Pro budoucí pokračování této práce by bylo vhodné prozkoumat další možnosti využití klíčových bodů. Například z klíčových bodů vypočítat úhly ohybu jednotlivých končetin a dostat tuto informaci do sítě. Je možné také experimentovat s jinými extraktory klíčových

bodů. Další možností je otestovat experimenty této práce na jiných datasetech, které obsahují jiné sportovní pozice.

Tato práce mi přinesla mnoho nových znalostí z oblastí počítačového vidění a strojového učení. Naučil jsem se modelovat a trénovat vlastní i převzaté architektury konvolučních neuronových sítí. Také jsem se naučil, jak lépe pracovat s obrázky a jaké možné úpravy lze na obrázcích provádět.

Literatura

[1] *MediaPipe, google developers*. Google. Dostupné z: https://developers.google.com/mediapipe.

[2] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR*. 2016, abs/1603.04467. Dostupné z: http://arxiv.org/abs/1603.04467.

[3] Abdelouahab, K. *Reconfigurable hardware acceleration of CNNs on FPGA-based smart cameras*. Disertační práce.

[4] Chollet, F. et al. *Keras* [https://keras.io]. 2015.

[5] Dumoulin, V. a Visin, F. *A guide to convolution arithmetic for deep learning*. 2018.

[6] Gombru, R. *Understanding categorical cross-entropy loss, binary cross-entropy loss, Softmax loss, logistic loss, focal loss and all those confusing names*. Květen 2018. Dostupné z: https://gombru.github.io/2018/05/23/cross_entropy_loss/.

[7] Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P. et al. Array programming with NumPy. *Nature*. Springer Science and Business Media LLC. září 2020, sv. 585, č. 7825, s. 357–362. DOI: 10.1038/s41586-020-2649-2. Dostupné z: https://doi.org/10.1038/s41586-020-2649-2.

[8] He, K., Zhang, X., Ren, S. a Sun, J. *Deep Residual Learning for Image Recognition*. 2015.

[9] He, K., Zhang, X., Ren, S. a Sun, J. *Identity Mappings in Deep Residual Networks*. 2016.

[10] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W. et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017.

[11] Kingma, D. P. a Ba, J. *Adam: A Method for Stochastic Optimization*. 2017.

[12] Krizhevsky, A., Sutskever, I. a Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C., Bottou, L. a Weinberger, K., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, sv. 25. Dostupné z: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[13] Mukherjee, S. *The annotated resnet-50*. Towards Data Science, Aug 2022. Dostupné z: https://towardsdatascience.com/the-annotated-resnet-50-a6c536034758.

[14] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*. 2015, sv. 115, č. 3, s. 211–252. DOI: 10.1007/s11263-015-0816-y.

[15] Solem, J. E. *Programming Computer Vision with Phyton*. O'Reilly, 2012.

[16] Sutskever, I., Martens, J., Dahl, G. a Hinton, G. On the importance of initialization and momentum in deep learning. In: Dasgupta, S. a McAllester, D., ed. *Proceedings of the 30th International Conference on Machine Learning*. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, sv. 28, č. 3, s. 1139–1147. Proceedings of Machine Learning Research. Dostupné z: https://proceedings.mlr.press/v28/sutskever13.html.

[17] Torralba, A., Fergus, R. a Freeman, W. T. 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2008, sv. 30, č. 11, s. 1958–1970. DOI: 10.1109/TPAMI.2008.128.

[18] Verma, M., Kumawat, S., Nakashima, Y. a Raman, S. Yoga-82: A New Dataset for Fine-grained Classification of Human Poses. In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2020, s. 4472–4479.

[19] Verma, M., Kumawat, S., Nakashima, Y. a Raman, S. Yoga-82: A New Dataset for Fine-grained Classification of Human Poses. *CoRR*. 2020, abs/2004.10362. Dostupné z: https://arxiv.org/abs/2004.10362.

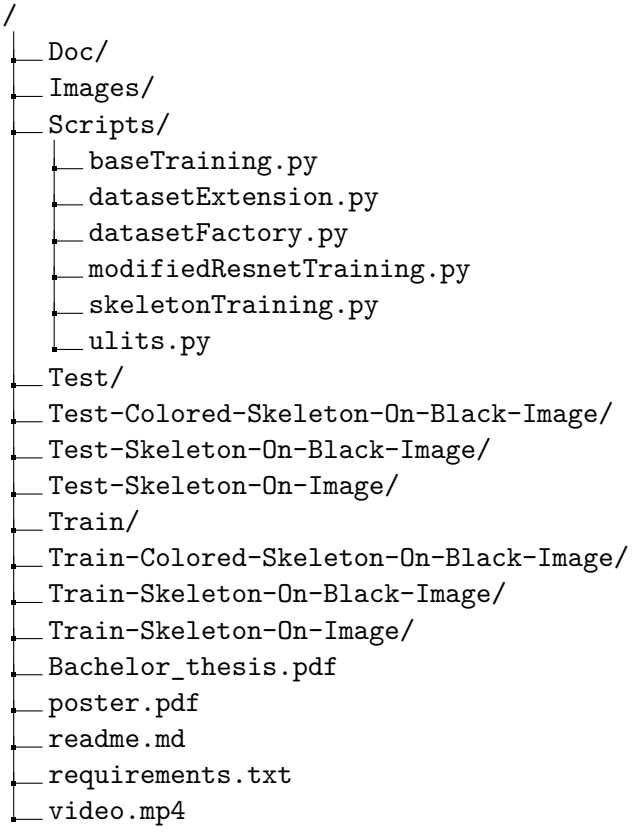
Příloha A

Obsah přiloženého paměťového média

V souboru `readme.md` se nachází podrobná dokumentace k celému obsahu paměťového média a skriptům. Složka `Doc` obsahuje zdrojové kódy LaTeXu. Složka `Images` obsahuje obrázky rozšířeného datasetu Yoga-82. Složka `Test` obsahuje obrázky datasetu Yoga-82 použité k validaci modelů. Složka `Train` obsahuje obrázky datasetu Yoga-82 použité k trénování modelů. Složky s názvy `Test-Colored-Skeleton-On-Black-Image`, `Test-Skeleton-On-Black-Image` a `Test-Skeleton-On-Image` obshajují obrázky tří nově vzniklých datasetů, které byly využity k validaci modelů. Složky s názvy `Train-Colored-Skeleton-On-Black-Image`, `Train-Skeleton-On-Black-Image` a `Train-Skeleton-On-Image` obshajují obrázky tří nově vzniklých datasetů, které byly využity k trénování modelů. Vzhled jednotlivých obrázků datasetů je popsán v sekci 5.2.1.

Složka `Scripts` obsahuje skripty jazyka Python. Jednotlivé skripty jsou:

- `baseTraining.py` - pro trénování a vytvoření základního modelu,
- `datasetExtension.py` - detekuje optický tok ve videu a ukládá obrázky, ve kterých nedochází k pohybu,
- `datasetFactory.py` - slouží k vytvoření prvního, druhého a třetího datasetu,
- `modifiedResnetTraining.py` - pro trénování a vytvoření modifikované architektury ResNet50v2, která umožňuje dva paralelní vstupy,
- `skeletonTraining.py` - pro trénování a vytvoření modelu, který umožňuje dva paralelní vstupy,
- `ulits.py` - třídění a konvertování formátů obrázků.



Obrázek A.1: Schéma obsahu paměťového média.