
**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**


# POUŽITÍ SELF-SUPERVISED LEARNING PRO ROZPOZNÁNÍ SPORTOVNÍCH POZIC V OBRAZE

**SELF-SUPERVISED LEARNING FOR RECOGNITION OF SPORTS POSES IN IMAGE**

**BAKALÁŘSKÁ PRÁCE**
**BACHELOR'S THESIS**


**AUTOR PRÁCE** **SAMUEL OLEKŠÁK**
**AUTHOR**

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



**BRNO 2022**

**Vysoké učení technické v Brně**
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2021/2022

# Zadání bakalářské práce


Student: **Olekšák Samuel**
Program: Informační technologie
Název: **Použití self-supervised learning pro rozpoznání sportovních pozic v obraze**
**Self-Supervised Learning for Recognition of Sports Poses in Image**
Kategorie: Zpracování obrazu

Zadání:

1. Seznamte se s problematikou strojového učení pro počítačové vidění a s problematikou rozpoznání sportovních pozic v obraze a videu.
2. Získejte a/nebo sestavte datovou sadu (sady) obrázků sportovních pozic.
3. Experimentujte s metodami self-supervised learning nad sestavenou datovou sadou (sadami).
4. Demonstrujte použitelnost vyvinutých technik pro rozpoznání sportovních pozic.
5. Iterativně vylepšujte vyvinuté techniky i datovou sadu směrem k maximální použitelnosti.
6. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek a krátké video pro prezentování výsledků projektu.

Literatura:


- Goodfellow, Bengio, Courville: Deep Learning, MIT Press, 2016
- Bharath Ramsundar, Reza Bosagh Zadeh: TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning, O'Reily Media, 2018
- Gary Bradski, Adrian Kaehler: Learning OpenCV; Computer Vision with the OpenCV Library, O'Reilly Media, 2008
- Richard Szeliski: Computer Vision: Algorithms and Applications, Springer, 2011
- Grill J-B et al.: Bootstrap your own latent: A new approach to self-supervised Learning, NeurIPS 2020, https://arxiv.org/abs/2006.07733
- Caron M et al.: Emerging Properties in Self-Supervised Vision Transformers, https://arxiv.org/abs/2104.14294
- Sermanet et al.: Time-Contrastive Networks: Self-Supervised Learning from Video, ICRA 2018, https://arxiv.org/abs/1704.06888
- Asano et al.: Self-labelling via simultaneous clustering and representation learning, ICLR 2020, https://arxiv.org/abs/1911.05371
- L. Jing, Y. Tian, Self-supervised visual feature learning with deep neural networks: A survey, IEEE PAMI, 2020


Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2, značné rozpracování bodů 3 až 5.

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, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 1. listopadu 2021

# Abstrakt

Táto práca demonštruje spôsob, ako minimalizovať množstvo potrebných označených trénovacích dát pri klasifikácií športových pozícií s použitím neurónovej siete trénovanej metódou contrastive self-supervised learning. Trénovanie prebieha v dvoch etapách. V prvej sa trénuje extraktor príznakov, ktorý využíva neoznačené trénovacie obrázky extrahované z nahrávok cvičení z viacerých uhlov. V druhej etape sa s využitím malého množstva označených dát trénuje jednoduchý klasifikátor napojený na extraktor príznakov. Práca pojednáva o klasifikácií v kontexte jogových póz, avšak výsledné riešenie sa dá jednoducho aplikovať aj na iné športy v prípade získania vhodnej dátovej sady. Pri návrhu riešenia je kladený dôraz na výkon výsledného modelu, aby mohol byť použiteľný v mobilných zariadeniach. Výsledný model na dátovej sade so štyrmi označenými obrázkami na každú jogovú pózu dosiahol s využitím augmentácií vstupných dát úspešnosť 76 %. Na väčšej dátovej sade s 800 označenými obrázkami na všetky pozície je úspešnosť 82 %.



# Abstract

This thesis demonstrates a solution for minimizing the amount of necessary labelled training data in the classification of sports poses using a neural network trained with contrastive self-supervised learning. Training consists of two stages. The first stage trains a feature extractor which uses unlabelled training images extracted from recordings of exercises from multiple viewpoints. In the second stage, using a small amount of labelled data, a simple classifier connected to the feature extractor is trained. The thesis discusses classification in the context of yoga poses, however, the final solution can be easily applied to any other sport in case of obtaining a suitable dataset. During the development of the solution, emphasis is placed on the performance of the resulting model so that it can be used on mobile devices. The resulting model reached an accuracy of 76 % using augmentations with a data set containing four labelled images per yoga pose. On a larger data set with 800 labelled images for all poses, an accuracy of 82 % is reached.



# Kľúčové slová

rozpoznávanie obrazu, odhad pózy, kontrastívne učenie, self-supervised learning

# Keywords

image recognition, pose estimation, contrastive learning, self-supervised learning


# Citácia

OLEKŠÁK, Samuel. *Použití self-supervised learning pro rozpoznání sportovních pozic v obraze*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce prof. Ing. Adam Herout, Ph.D.

# Použití self-supervised learning pro rozpoznání sportovních pozic v obraze

## Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením prof. Ing. Adama Herouta, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Samuel Olekšák
17. mája 2022


## Poďakovanie

Chcel by som sa poďakovať prof. Ing. Adamovi Heroutovi, Ph.D. za odbornú pomoc pri tvorení bakalárskej práce a tiež za poskytnutie dátovej sady a prístupu k výpočtovému serveru `sophie.fit.vutbr.cz`.

# Obsah

# Kapitola 1

# Úvod

Pandémia koronavírusu si vynútila obmedzenie medziľudského kontaktu, čo malo za následok dlhodobé uzavretie športovísk, skončenie skupinových cvičení a nemožnosť osobného styku s trénermi. Mnoho ľudí nebolo schopných sa podieľať na ich pravidelných kolektívnych alebo individuálnych športoch a fyzických aktivitách mimo domu. Čiastočnou náhradou sa stalo cvičenie z domu, čo spôsobilo zvýšený dopyt po aplikáciách, ktoré pri tom asistujú. Jednou z populárnych fyzických aktivít je cvičenie jogy. Cvičenie jogy bez možnosti pomoci inštruktora môže byť náročné, a aj preto je dopyt po mobilných aplikáciách na asistenciu pri cvičení jogy, ktoré by inštruktora dokázali aspoň sčasti nahradiť.

Cieľom tejto práce je vytvoriť model neurónovej siete, ktorý dokáže rozpoznávať jogové pozície bez toho, aby potreboval veľké množstvo anotovaných dát. Za týmto účelom sú použité techniky self-supervised learning (preložiteľné do slovenčiny ako *samomonitorované učenie*, slovenský výraz sa zatiaľ neujal), ktoré pri rozdelení trénovacieho procesu na viac etáp umožňujú v časti trénovacieho procesu používať iba neoznačené dáta.

Pri návrhu riešenia bol kladený dôraz na výkon výsledného modelu, keďže by bol veľmi dobre využiteľný v podobe aplikácie na mobilnom telefóne. Fragmentácia trénovacieho procesu má aj ďalšiu výhodu – umožňuje náročnejšiu časť trénovacieho procesu vykonať pred distribúciou cieľovej aplikácie, napríklad vytrénovať sieť na klasifikáciu jogových póz na výpočtovo výkonnom zariadení a zvyšnú časť trénovania, tzv. dotrénovanie (*fine-tuning*), dokončiť na cieľovom zariadení, napríklad spôsobom, že používateľ si definuje pózy, ktoré chce rozoznávať a aplikácii poskytne označené trénovacie dáta. Táto metóda sa nazýva trénovanie na cieľovom zariadení (*on-device training*) a podporuje ju aj knižnica na nasadzovanie modelov do mobilných zariadení *TensorFlow Lite*[1].

Práca využíva dátovú sadu pozostávajúcu z videozáznamov cvičení jogy zaznamenávaných z rôznych uhlov. Popísaný je spôsob extrakcie snímok z videí, aby mohli byť využité pri trénovaní neurónovej siete. Nasleduje popis architektúry siete schopnej klasifikovať obrázky spolu s popisom trénovacej slučky. Experimentálne je overená presnosť klasifikácie siete a vplyv rôznych náhodných augmentácií vstupných dát na ňu. Časť trénovacích a testovacích dát sa nachádza na priloženom pamäťovom médiu (príloha A) spolu so všetkými skriptami, ktoré spracúvajú dátovú sadu a vytvárajú výsledný natrénovaný model. Riešenie je implementované v jazyku Python s použitím knižníc *TensorFlow* a *OpenCV*.

Výsledné riešenie sa dá pri získaní vhodných trénovacích dát aplikovať na ľubovoľný šport alebo proces, ktorý vyžaduje klasifikáciu pozícií jednej osoby v obraze a má potenciálne aplikácie pri trénovaní, analýze pohybu pri športe, rehabilitácií ai.

[1] https://www.tensorflow.org/lite/examples/on_device_training/overview

# Kapitola 2

# Detekcia športových pozícií

S rastúcim výkonom mobilných zariadení rastie aj ich využiteľnosť v oblastiach klasifikácie obrazu v reálnom čase. Motiváciou k návrhu a implementácií modelu na báze self-supervised learning je vytvorenie mobilnej aplikácie, ktorá by bola schopná asistovať pri cvičení jogy tým, že bude cvičené jogové pozície zaznamenávať a klasifikovať. Keďže jogových pozícií je veľmi veľký počet a ich názvy nie sú univerzálne, bolo by vhodné, aby si používateľ dokázal sám v rámci aplikácie definovať pozíciu tým, že si ju sám nazve a nahrá niekoľko fotiek ako túto pozíciu cvičí.

V prípade použitia monolitickej architektúry modelu by sa celá sieť po definícií novej pózy musela natrénovať znovu, čo na cieľovom mobilnom zariadení nie je výpočtovo možné. Riešením je rozdelenie siete a trénovacieho procesu na dve časti.

Prvá časť siete bude zodpovedná za extrakciu príznakov z obrázka. Ideálnym stavom by bolo, aby príznaky boli závislé čisto od sémantickej informácie (napr. príznak udávajúci či je ľavé koleno ohnuté v pravom uhle alebo príznak reprezentujúci, či sú obe ruky predpažené) a zanedbávali informáciu nepodstatnú pre klasifikáciu (farbu oblečenia cvičiaceho a jeho telesné proporcie, predmety v pozadí, poloha kamery vzhľadom k cvičiacemu ai.).

Druhou časťou modelu by bol klasifikátor, ktorý by zobrazoval vektor príznakov na vektor pravdepodobností jednotlivých póz. Klasifikačná časť modelu by sa mohla dotrénovať na cieľovom zariadení nezávisle na extraktore príznakov, aby dokázala poznatky z používateľom definovaných pozícií zakomponovať do klasifikácie.

Pred trénovaním prvej časti slúžiacej na extrakciu príznakov by sme si mohli klásť otázku: „Ako určíme, ktoré príznaky zodpovedajú danému vstupnému štítku, aby sme vedeli určiť očakávaný výstup pri trénovaní?“ Cieľom extraktora príznakov nie je zobrazovať vstupy na konkrétne výstupy, ale snažiť sa, aby podobné pózy dostali podobné ohodnotenie. Preto je lepšia formulácia otázky: „Ako naučíme extraktor zobrazovať podobné pózy na podobné príznakové vektory?“ Za povšimnutie stojí to, že pri využití tohto riešenia nie je potrebné poznať štítok vstupného obrázka s jogovou pózou v prípade, že obrázky získavame z videa. Dve snímky z rôznych časov vo videu s vysokou pravdepodobnosťou obsahujú dve rôzne pozície, pričom znalosť konkrétneho názvu pozície nie je potrebná na to, aby sme sieti vedeli zadať úlohu, aby sa snažila reprezentácie týchto dvoch obrázkov od seba oddialiť. Priestor, kde sú príznakové vektory podobných vzoriek zobrazené blízko seba, sa nazýva latentný priestor (*latent space*). Tento diskriminatívny prístup sa nazýva kontrastívne učenie (*contrastive learning*), ktoré je podkategóriou self-supervised learning.

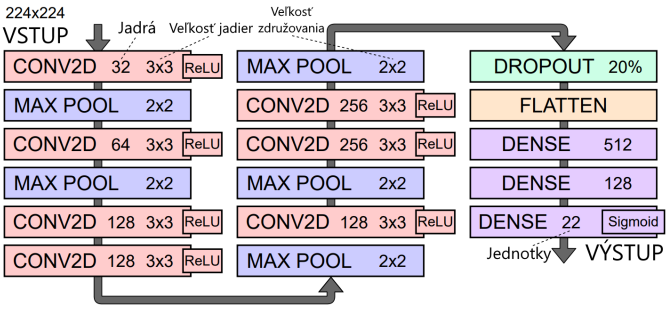
## 2.1 Existujúce riešenia

V tejto sekcii sú popísané dve existujúce riešenia na klasifikáciu jogových pozícií v obraze a dve existujúce riešenia používajúce diskriminatívny prístup k učeniu sa reprezentácií z obrázkov.

### Detection of a Yoga Poses in Image [18]

Práca navrhuje a implementuje konvolučnú neurónovú sieť na detekciu jogových pozícií (architektúra je zobrazená na obr. 2.1). Používa sa veľmi podobná dátová sada, avšak využíva sa iba učenie s učiteľom. Použitá dátová sada obsahuje až 22 000 označených trénovacích obrázkov pre 22 rôznych jogových pozícií.

Presnosť modelu v klasifikácií jogových pozícií dosahuje 91 %. Implementácia pracuje s knižnicou *TensorFlow* a najmä s jej modulom *Keras*. Práca experimentuje so vstupnými obrázkami s veľkosťou $224 \times 224$ a $96 \times 96$ pixelov. Súčasťou práce je aj anotačný nástroj napísaný v jazyku Python, ktorý bol inšpiráciou na vytvorenie webovej anotačnej aplikácie použitej v tomto projekte.




Obr. 2.1: Schéma architektúry jedného z najúspešnejších modelov práce *Detection of a Yoga Poses in Image*. Model obsahuje 7,7 milióna trénovateľných parametrov pre vstup $224 \times 224$ a 2,5 milióna pre vstup $96 \times 96$. Prevzaté z [18] a následne preložené.

### YogAI: Smart Personal Trainer [20]

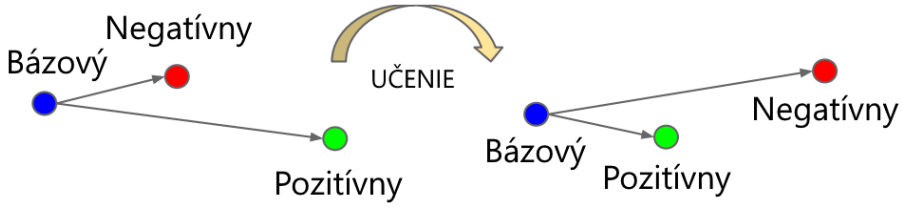
Projekt rieši odhad pozície pomocou zariadenia *Raspberry Pi*. Odhad slúži na asistenciu a opravu pozícií pri cvičení jogy. Riešenie používa knižnicu *OpenPose*, ktorá dokáže detegovať kľúčové body tela (až 135 kľúčových bodov) v obraze s viacerými osobami v reálnom čase. Získané kľúčové body vstupujú do klasifikátora založeného na konvolučnej neurónovej sieti.

Pri trénovaní siete sú využívané augmentácie kľúčových bodov, čo má za následok obohatenie trénovacej dátovej sady a prevenciu preučenia. Konkrétne využité náhodné augmentácie sú posunutie všetkých bodov po osi $x$ alebo $y$, horizontálne preklopenie a otočenie.

### FaceNet: A Unified Embedding for Face Recognition and Clustering [22]

Predstavená sieť *FaceNet* sa učí zobrazovať tváre do kompaktného euklidovského priestoru, kde vzdialenosť zodpovedá metrike podobností tvárí. Po tom, ako je tento priestor vytvorený, úlohy ako rozpoznávanie tváre, verifikácia a zhlukovanie sú jednoduché na implementáciu s použitím štandardných techník, pričom vstupom je vektor príznakov získaný zo siete *FaceNet*.

Metóda používa hlbokú konvolučnú neurónovú sieť, ktorá je trénovaná s využitím trojicovej stratovej funkcie (*triplet loss*, obr. 2.2) tak, aby jej výstupom bol 128-dimenzionálny príznakový vektor. Trojice pozostávajú z dvoch orezov jednej tváre a jedného orezu inej tváre.

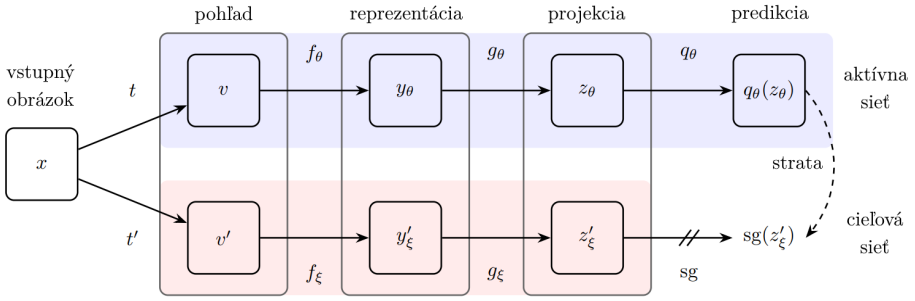



Obr. 2.2: Cieľom trojicovej stratovej funkcie je minimalizovať vzdialenosť medzi bázovou a pozitívnou vzorkou, pričom obe majú rovnakú identitu, a zároveň maximalizovať vzdialenosť medzi bázovou a negatívnou vzorkou, ktoré majú rôzne identity. Prevzaté z [22] a následne preložené.

### Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning [14]

*Bootstrap Your Own Latent* (BYOL) predstavuje prístup k učeniu reprezentácií, ktorý sa spolieha na dve neurónové siete, nazývané aktívna (*online*) a cieľová (*target*), ktoré sa navzájom učia (obr. 2.3). Aktívna sieť je trénovaná z augmentovanej verzie obrázka a jej úlohou je predpovedať, akú bude mať cieľová sieť reprezentáciu pre rovnaký obrázok s inou augmentáciou. Zároveň sa cieľová sieť priebežne aktualizuje kĺzavým priemerom aktívnej siete.

Zatiaľ čo sa väčšina moderných self-supervised metód spolieha na negatívne páry, BYOL dosahuje vysokú úspešnosť aj bez nich – 74,3 % top-1 presnosť klasifikácie na dátovej sade *ImageNet* s použitím lineárnej evaluácie s *ResNet-50* architektúrou a 79,6 % s väčšou *ResNet* sieťou.

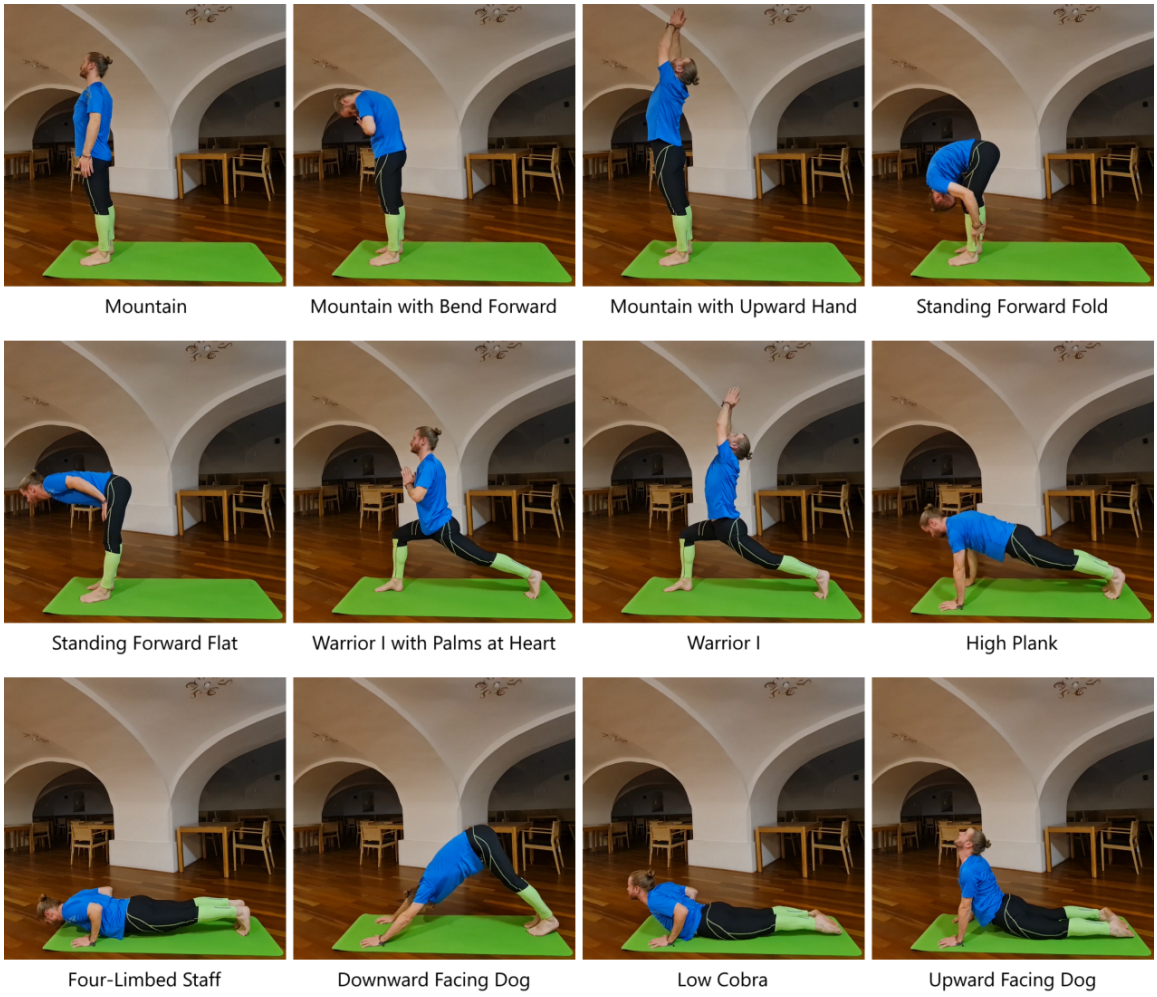



Obr. 2.3: Architektúra BYOL, ktorá sa snaží minimalizovať hodnotu podobnostnej stratovej funkcie medzi $q_\theta(z_\theta)$ a $sg(z'_\xi)$, pričom $\theta$ sú trénované váhy, $\xi$ je exponenciálny kĺzavý priemer váh $\theta$ a $sg$ značí stop-gradient. Na konci trénovania je všetko okrem $f_\theta$ zahodené a $y_\theta$ sa používa na prevedenie obrázkov do priestoru reprezentácií. Prevzaté z [14] a následne preložené.

## 2.2 Cvičenie jogy

Joga [21] je starodávna činnosť, ktorá má svoj pôvod pravdepodobne v Indii. Zahŕňa pohyb, meditáciu a dýchacie techniky na podporu duševnej a fyzickej pohody. V praxi existuje niekoľko druhov jogy a mnoho disciplín. V tejto práci bude pojem joga používaný ako moderná forma *Hatha jogy*, čo sú techniky slúžiace na zlepšenie fyzickej zdatnosti, zmiernenie stresu a relaxáciu založené na pozíciach (*asanas*).

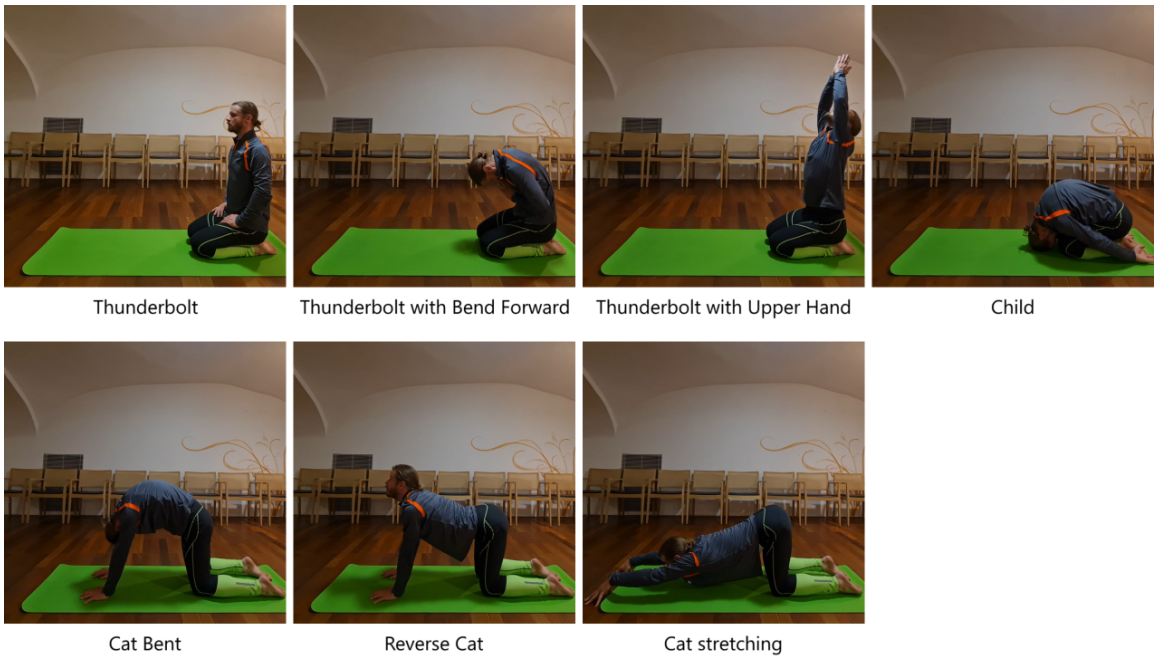
Cvičenie pozostáva z postupného zaujatia pozícií, pričom cvičiaci (tzv. jogín) sa v každej pozícií na niekoľko sekúnd zastaví. Jednotlivé pozície cvičenia môžu byť súčasťou sekvencie. Medzi známe sekvencie patrí napr. pozdrav slnku (*Sun salutation*, pozície sekvencie sú zobrazené na obrázku 2.4). Ďalšie sekvencie vyskytujúce sa v dátovej sade použitej v tejto práci sa nachádzajú na obrázkoch 2.5, 2.6 a 2.7. Nemusí platiť, že každá pozícia sa v jednej sekvencii vyskytne iba raz.




Obr. 2.4: Pozdrav slnku (*Sun salutation*)

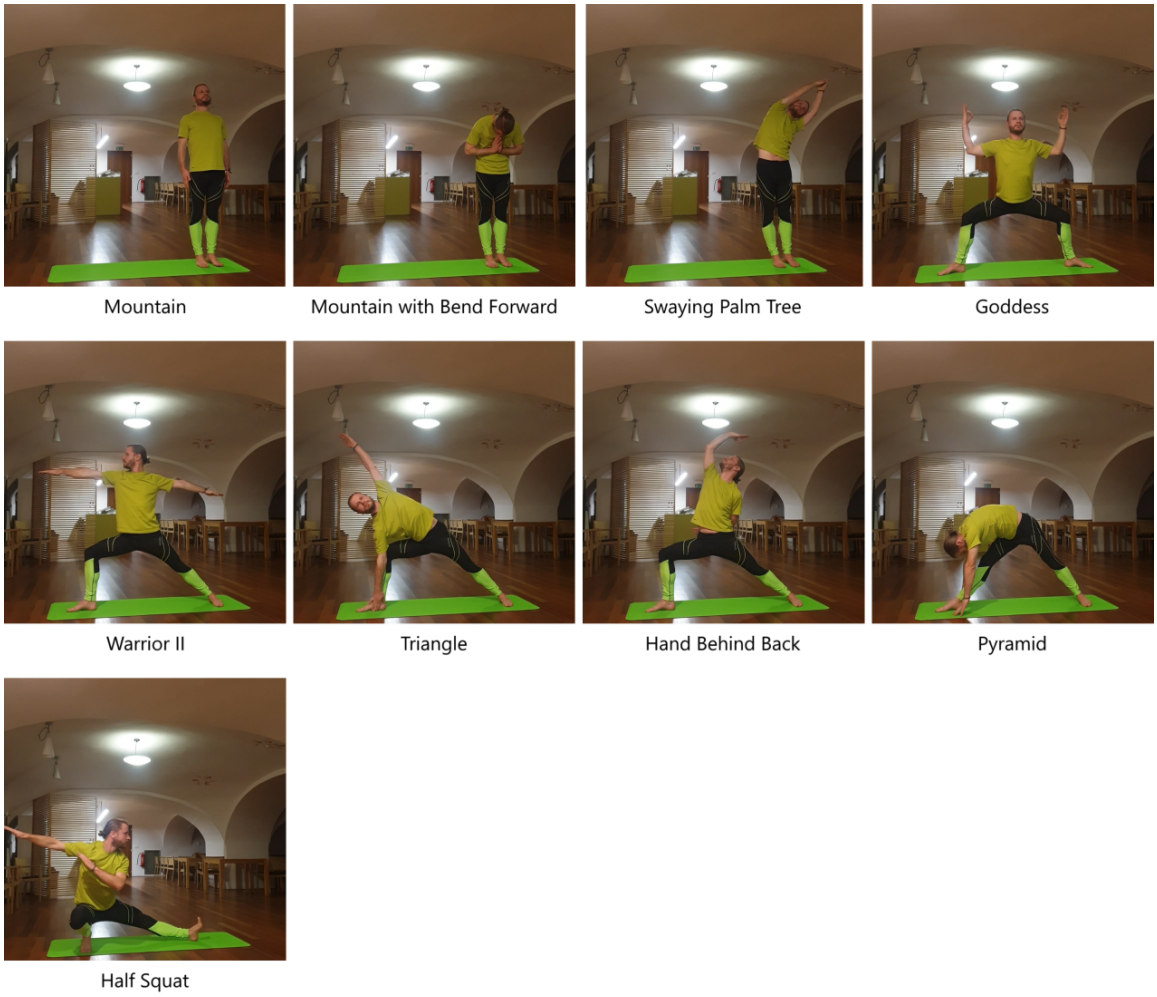
Dôležité je poznamenať, že neexistuje jeden predpísaný spôsob v akom poradí a aké presne pozície sa v sekvencii vyskytnú. V rôznych zdrojoch sa tieto informácie líšia. Rovnako aj každá pozícia môže mať rôzne názvy.

Jednotlivé pozície môžu byť cvičené viacerými spôsobmi, napríklad pozícia s názvom stolička (*Utkatasana*), ktorá je realizovaná v miernom podrepe s rukami buď vzpaženými, predpaženými alebo s dlaňami spojenými pri hrudi. Táto nejednoznačnosť komplikuje vytvorenie univerzálneho klasifikátora, ktorý dokáže rozpoznávať každú variáciu každej pozície. Z tohto dôvodu je užitočné, aby si používateľ dokázal sám definovať konkrétne pozície spôsobom, akým ich sám cvičí.




Obr. 2.5: Mačacia sekvencia (*Cat sequence*)


Obr. 2.6: Sekvencia bojovníka III (*Warrior III sequence*)




Obr. 2.7: Pozdrav mesiacu (*Moon salutation*)

# Kapitola 3

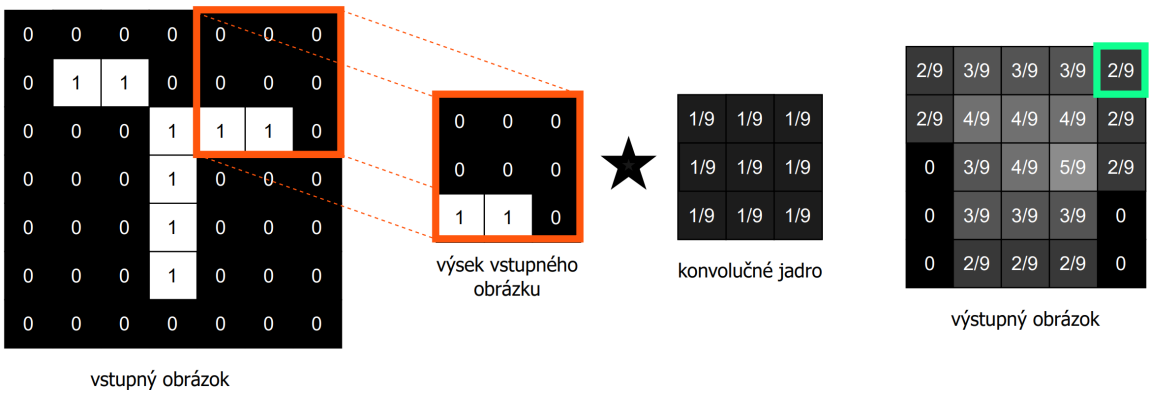
# Klasifikácia obrazu pomocou KNN

Konvolučné neurónové siete (KNN) sú neurónové siete, ktoré vo svojej štruktúre obsahujú konvolučné vrstvy. Konvolučné vrstvy, v kontexte klasifikácie obrázkov, umožňujú sieti využiť informácie o 2D štruktúre vstupných obrázkov s využitím operácie 2D konvolúcie (obrázok 3.1). Jednodimenzionálny analóg 2D konvolúcie sa nachádza v rovnici (3.1). Jednoduchšie siete pred spracovávaním musia vstupný obrázok sploštiť (*flatten*) do jednej dimenzie, čím sa stratí informácia o 2D štruktúre.

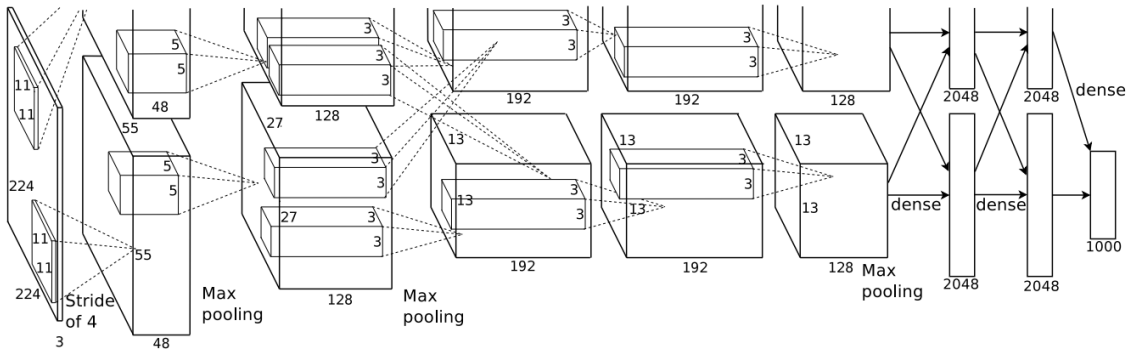
Konvolučnú vrstvu charakterizuje počet filtrov, veľkosť konvolučného jadra (*kernel size*) a veľkosť kroku (*stride*). Počet filtrov udáva počet výstupných kanálov, pričom každý z nich má vlastné trénovateľné parametre. Typicky používané veľkosti konvolučného jadra sú $3 \times 3$ alebo $5 \times 5$. Veľkosť kroku udáva počet pixelov, o ktorý sa, v rámci vstupného obrázka, posunie konvolučné jadro medzi každým krokom.

Aplikácie KNN siahajú od klasifikácie a segmentácie obrazu až po spracúvanie prirodzeného jazyka. Jedna z najznámejších architektúr KNN – *Alexnet* – je ukázaná na obrázku 3.2.

$$x[n] \star y[n] = \sum_{k=-\infty}^{\infty} x[k] \cdot y[n-k] \tag{3.1}$$




Obr. 3.1: Ukážka operácie 2D konvolúcie.




Obr. 3.2: Schéma jednej z najznámejších architektúr KNN – *Alexnet*. Obsahuje 5 konvolučných vrstiev s jadrami s veľkosťou $11 \times 11, 5 \times 5$ a $3 \times 3$. Prevzaté z [17].

## 3.1 Trénovanie neurónových sietí

Trénovanie neurónových sietí [3] je v princípe optimalizačná úloha, pri ktorej chceme maximalizovať schopnosť siete riešiť nami zadanú úlohu. Aby sa dala kvantifikovať úspešnosť siete, používa sa takzvaná stratová funkcia (*loss function*). Stratová funkcia vypočítavá vzdialenosť sieťou predpovedaného výstupu a očakávaného výstupu. Cieľom optimalizátora pri trénovaní siete je túto vzdialenosť minimalizovať.

Optimalizátor je algoritmus, ktorý slúži na aktualizovanie parametrov siete v priebehu učenia. Moderné neurónové siete typicky používajú na aktualizovanie parametrov gradientný zostup (*gradient descent)*. Gradientný zostup [5, 23] je iteratívny algoritmus, ktorý začína na náhodnom bode funkcie (pred trénovaním sa váhy typicky inicializujú na náhodné hodnoty) a pokračuje v smere gradientu k minimu.

Keďže počítanie gradientu je pre veľké množstvo dát pomalé, využíva sa náhodné vzorkovanie celej dátovej sady. Táto sada náhodne vybratých vzoriek sa nazýva minidávka (*minibatch*).

Po spracovaní každej dávky a vypočítaní gradientu sa váhy siete aktualizujú. Čím viac vzoriek je v dávke, tým presnejší bude odhad gradientu za cenu toho, že je potrebné vykonať väčšie množstvo predikcií pred každým aktualizovaním váh. Naopak, použitie menšej dávky vedie k menej presnému odhadu gradientu, ktorý viac závisí od konkrétnych vzoriek v dávke. Podľa veľkosti dávky sa metóda delí na:

- dávkový gradientný zostup (*batch gradient descent*) – veľkosť dávky je rovná celkovému množstvu trénovacích príkladov,
- stochastický gradientný zostup (*stochastic gradient descent*) – veľkosť dávky je rovná jednej,
- minidávkový gradientný zostup (*minibatch gradient descent*) – veľkosť dávky je medzi hodnotou 1 a veľkosťou celej trénovacej dátovej sady.

Konkrétnou stratovou funkciou, ktorá sa typicky používa pri klasifikácií s mnohými štítkami, pričom vstup má stále práve jeden štítok (*label*), je funkcia *Categorical cross-entropy* (3.2), ktorá je dobrou metrikou podobnosti správneho ($y$) a sieťou predpovedaného ($\hat{y}$) vektora pravdepodobností s veľkosťou $n$. Stratová funkcia sa ráta z výstupu siete, a teda výstupu aktivačnej funkcie poslednej vrstvy.

$$L = -\sum_{i=1}^{n} y_i \cdot \log \hat{y}_i \tag{3.2}$$

Typicky používanou aktivačnou funkciou poslednej vrstvy neurónovej siete pri klasifikácií obrázkov s práve jedným štítkom je softmax. Výpočet hodnoty softmax funkcie pre kategóriu $i$ z výstupného vektora $\vec{y}$ poslednej vrstvy siete pri počte $n$ kategórií je v rovnici (3.3). Softmax funkcia sa vyznačuje tým, že súčet hodnôt výstupného vektora je rovný 1, takže vstupné hodnoty sú aplikovaním funkcie normalizované na rozdelenie pravdepodobností jednotlivých kategórií.

$$\sigma(\vec{y})_i = \frac{e^{y_i}}{\sum_{j=1}^{n} e^{y_j}} \tag{3.3}$$

### 3.1.1 Augmentácia trénovacích dát

Častým problémom pri trénovaní neurónových sietí, v prípade, že trénovacia dátová sada je málo rozmanitá a vyvážená, je to, že sa sieť naučí klasifikovať na základe nesémantických informácií, napríklad farba oblečenia cvičiaceho, predmety v pozadí alebo výška kamery od zeme. Tento jav sa nazýva preučenie (*overfitting*) a prejavuje sa vysokou úspešnosťou klasifikácie trénovacích dát oproti dátam, ktoré sieť pri trénovaní nevidela.

Jednou z techník využívaných na predídenie preučenia je augmentácia trénovacích dát. Jej princíp spočíva v aplikovaní náhodných zmien na trénovacie obrázky, čím sa v trénovacej sade minimalizuje nesémantická informácia a zachováva sémantická. Často využívané augmentácie pri klasifikácií obrazu sa nachádzajú na obrázku 3.3.


Obr. 3.3: Ukážka rôznych augmentácií na obrázku jogovej pózy.

Augmentácia trénovacích dát sa taktiež často používa pri vytváraní pozitívnych obrázkov k bázovým obrázkom pri kontrastívnom aj nekontrastívnom učení. Obe uvedené metódy učenia sú popísané v kapitole 3.2.3.

## 3.2 Self-supervised learning

Tradičné prístupy učenia s učiteľom sa spoliehajú na veľké množstvo dostupných anotovaných dát, pričom anotácie slúžia na generovanie kontrolných signálov (*supervisory signals*). Self-supervised learning (SSL) [19] získava kontrolné signály zo štruktúry samotných údajov.

Typickou úlohou self-supervised learning je predpovedať akúkoľvek nepozorovanú alebo skrytú časť (alebo vlastnosť) vstupu z akejkoľvek pozorovanej alebo viditeľnej časti vstupu.

Napríklad, v prípade spracovania prirodzeného jazyka, môžeme skryť časť vety a predpovedať skryté slová zo zostávajúcich slov. Môžeme tiež predpovedať minulé alebo budúce snímky vo videu (skryté údaje) z aktuálnych (viditeľné údaje). Keďže self-supervised learning využíva štruktúru samotných údajov, môže využívať rôzne kontrolné signály naprieč modalitami (napr. video a zvuk) a naprieč veľkými súbormi údajov bez spoliehania sa na štítky.

Trénovanie siete využívajúcej SSL typicky pozostáva z dvoch etáp – samotného SSL vyžadujúceho iba neoznačené dáta a dolaďovacej fázy (*fine-tuning*), kedy sa model s využitím označených dát prispôsobuje konkrétnej úlohe.

### 3.2.1 Pomocná úloha

Pomocné úlohy (*pretext tasks, proxy tasks*) [16] pri self-supervised learning tvoria časť trénovacieho procesu, kedy sa sieť učí bez učiteľa. Počas tejto etapy sa sieť snaží naučiť reprezentovať údaje pomocou pseudoštítkov. Tieto pseudoštítky sú generované automaticky na základe atribútov nachádzajúcich sa v dátach, bez nutnosti poskytnutia explicitných štítkov.

Model naučený z pomocnej úlohy môže byť doladený učením s učiteľom a použitý na rôzne následné úlohy v počítačovom videní, ako je klasifikácia, segmentácia, detekcia atď. Pomocné úlohy môžu byť navrhnuté pre rôzne modality, ako je obraz, video, reč, signály, a tak ďalej.

### 3.2.2 Siamská neurónová sieť

Siamská neurónová sieť (*Siamese neural network*) [4, 12] je typ sieťovej architektúry, ktorá obsahuje dve alebo viac identických podsietí prijímajúcich odlišné vstupy, ktoré sú na ich konci spojené energetickou funkciou. Táto funkcia vypočítava určitú metriku medzi výstupmi podsietí v príznakovom priestore.

Parametre medzi podsieťami sú zdieľané, čo zaručuje, že dva veľmi podobné obrázky nemôžu ich príslušné siete zobraziť na výrazne odlišné miesta v príznakovom priestore, pretože každá sieť vypočíta rovnakú funkciu.

Siamské siete sa používajú na rôzne úlohy, ako je detekcia duplikátov, hľadanie anomálií a rozpoznávanie tváre.

### 3.2.3 Kontrastívne učenie

Jednou z konkrétnych aplikácií siamských neurónových sietí je kontrastívne učenie [15]. Predstavme si jednoduchú sieť, ktorú chceme naučiť, aby pre vstupné vzorky s rovnakým štítkom dávala na výstup čo najpodobnejšiu reprezentáciu v latentnom priestore.

Za týmto cieľom si môžeme vytvoriť siamskú sieť, ktorej budeme dávať na vstup dva obrázky s rovnakým štítkom a ako stratovú funkciu použijeme vzdialenosť v latentnom priestore medzi výstupmi, ktorú chceme minimalizovať. Avšak, táto sieť by bola veľmi náchylná na skĺznutie k triviálnemu riešeniu. Napríklad sieť, ktorá pre každý vstup dáva úplne rovnaký výstup, by bola optimálnym riešením tejto úlohy.

Riešenia, ktoré využívajú iba kladné dvojice síce existujú (napríklad prístup *Bootstrap your own latent* [14]), ale často sú, z dôvodu prevencie zrútenia sa, komplikovanejšie oproti sieťam, ktoré používajú aj negatívne vzorky. Tento prístup, pracujúci iba s pozitívnymi vzorkami, sa nazýva nekontrastívne učenie.

Naopak, pri kontrastívnom učení sa siamská sieť skladá z troch identických podsietí a pri trénovaní sa používajú trojice obrázkov – bázový (*anchor*, $A$), negatívny (*negative*, $N$) a pozitívny (*positive*, $P$). Pôvodný obrázok funguje ako báza, jeho transformovaná verzia pôsobí ako pozitívna vzorka a jeden zo zvyšných obrázkov z dávky (*batch*) alebo z celých trénovacích dát pôsobí ako negatívna vzorka.

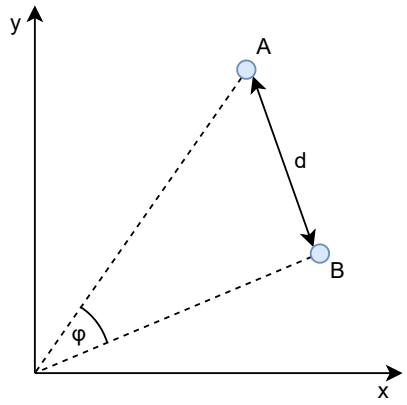
Kontrastívne učenie je diskriminatívny prístup, ktorého cieľom je minimalizovať vzdialenosť pozitívnych dvojíc a zároveň maximalizovať vzdialenosť negatívnych dvojíc v latentnom priestore. Za týmto účelom sa používa trojicová stratová funkcia (*triplet loss*) uvedená v rovnici (3.4) [22]. Najmä pre úlohy počítačového videnia sa kontrastívna stratová funkcia vyhodnocuje na základe reprezentácií obrázkov získaných extraktorom príznakov $f$.

$$L(A, P, N) = \max(dist(f(A), f(P)) - dist(f(A), f(N)) + m, 0) \tag{3.4}$$

V rovnici sa nachádza člen $m$, ktorého hodnota je konštantná. Tento člen sa nazýva marža (*margin*) a udáva požadovanú vzdialenosť bázovej a negatívnej vzorky.

Zvolená metrika vzdialeností dvojíc *dist* sa môže rôzniť. Typicky používanou je kosínusová vzdialenosť. Rovnica (3.5) uvádza výpočet kosínusovej podobnosti dvoch vektorov $\vec{x}$ a $\vec{y}$. Kosínusová vzdialenosť sa odvodí odčítaním kosínusovej podobnosti od hodnoty 1. Metrika euklidovskej vzdialenosti spolu so všetkými $L_k$ normami trpí tzv. kliatbou dimenzionality (*Curse of dimensionality*) [2], ktorá hovorí o tom, že euklidovská vzdialenosť je v mnoho-dimenzionálnom priestore medzi rôznymi bodmi veľmi malá. Ilustračný obrázok euklidovskej a kosínusovej vzdialenosti v 2D priestore sa nachádza na obrázku 3.4.

$$\cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}|| \cdot ||\vec{x}||} = \frac{\sum_{i=0}^{n-1} x_i \cdot y_i}{\sqrt{\sum_{i=0}^{n-1} x_i^2} \cdot \sqrt{\sum_{i=0}^{n-1} y_i^2}} \tag{3.5}$$




Obr. 3.4: Body $A$ a $B$ v 2D priestore a vyznačenie ich rôznych metrík vzdialeností. Úsečka $d$ reprezentuje euklidovskú ($L_2$) vzdialenosť a uhol $\varphi$ vyznačuje kosínusovú vzdialenosť bodov. Avšak, v kontexte latentného priestoru sa pracuje s výrazne viac dimenziami než dve.

Pri rozdelení trénovacieho procesu na dve úlohy – pomocnú a dotrénovaciu – sa kontrastívne učenie používa v prvej etape, kedy sa model učí reprezentácie vzoriek v latentnom priestore. Následne sa táto informácia procesom zvaným prenos vedomostí (*knowledge transfer*) zužitkuje v nasledujúcej fáze.

## 3.3 Few-shot learning

Aplikácie strojového učenia typicky vyžadujú veľké množstvo dát na trénovanie, čo umožňuje sieti zovšeobecniť svoje znalosti a úspešne ich aplikovať na testovaciu sadu. Existujú však aplikácie, kde je získanie dostatočného množstva trénovacích dát náročné. Cieľom few-shot learning (FSL) [11, 25] je vytvoriť schopné modely strojového učenia s menším množstvom potrebných trénovacích dát, čo sa viac blíži spôsobu, akým sa učia ľudia.

FSL sa používa najmä na úlohy učenia s učiteľom. Príklady aplikácií zahŕňajú klasifikáciu obrazu, rozpoznávanie objektov, klasifikáciu sentimentu z krátkeho textu, spracovanie prirodzeného jazyka a spracovanie zvuku.

## 3.4 Sieť *MoveNet*

*MoveNet* [13, 24] je konvolučná neurónová sieť, ktorá z obrázkov predpovedá polohu kľúčových bodov tela jednej osoby. Model je navrhnutý tak, aby bol spustiteľný v prehliadači pomocou knižnice *Tensorflow.js* alebo na mobilnom zariadení pomocou *TensorFlow Lite* v reálnom čase, na použitie na pohybové a fitness aktivity.

Variant `MoveNet.SinglePose.Lightning` je menej náročný model (v porovnaní s `MoveNet.SinglePose.Thunder`), ktorý dokáže bežať na frekvencii vyše 50 snímok za sekundu na väčšine moderných notebookoch pri dosahovaní dobrého výkonu.

Architektúra pozostáva z dvoch komponentov – extraktora príznakov a sady predikčných hláv. Schéma predikčných hláv rozširuje *CenterNet* [26] s výraznými zmenami, ktoré zlepšujú rýchlosť aj presnosť. Všetky modely sú trénované pomocou *TensorFlow Object Detection API*.

Extraktor príznakov v *MoveNet* využíva sieť *MobileNetV2* s dekodérom sietí príznakových pyramíd (*Feature Pyramid Network decoder*). K extraktoru príznakov sú pripojené štyri predikčné hlavy, ktoré sú zodpovedné za hustú predikciu nasledujúcich znakov:

- Teplotná mapa stredu osoby – predpovedá geometrický stred inštancií osôb.
- Pole regresie kľúčových bodov – predpovedá celú množinu kľúčových bodov pre osobu, čo sa používa pri zoskupovaní kľúčových bodov do inštancií osôb.
- Teplotná mapa kľúčových bodov osoby – predpovedá umiestnenie všetkých kľúčových bodov nezávisle od inštancií osôb.
- Pole 2D posunu každého kľúčového bodu – predpovedá presnú polohu každého kľúčového bodu.

Vstupom siete je video alebo obrázok, reprezentovaný ako tenzor typu `int32` a tvaru $192 \times 192 \times 3$. Poradie farebných kanálov je RGB v rozsahu hodnôt 0 až 255. Výstupom je tenzor typu `float32` s rozmermi $1 \times 1 \times 17 \times 3$.

Prvé dva kanály poslednej dimenzie predstavujú súradnice $y$ a $x$ (normalizované na rozmery snímky, t. j. v rozsahu 0,0 až 1,0) 17-tich kľúčových bodov (obr. 3.5) v nasledujúcom poradí – nos, ľavé oko, pravé oko, ľavé ucho, pravé ucho, ľavé rameno, pravé rameno, ľavý lakeť, pravý lakeť, ľavé zápästie, pravé zápästie, ľavé bedro, pravé bedro, ľavé koleno, pravé koleno, ľavý členok, pravý členok. V treťom kanáli poslednej dimenzie sa nachádza hodnota spoľahlivosti, ktorá vyjadruje to, aký si je model istý polohou kľúčového bodu.


Obr. 3.5: Jednotlivé kľúčové body tela vrátane ich indexov vo výstupnom vektore modelu. Farba bodu značí hodnotu spoľahlivosti na škále od zelenej farbe k červenej, pričom sýta zelená znamená, že si je model veľmi istý.

# Kapitola 4

# Knižnica *TensorFlow*

*TensorFlow* [1] je knižnica od Googlu s otvoreným zdrojovým kódom určená na strojové učenie a umelú inteligenciu. Poskytuje viacero vrstiev abstrakcie, vrátane vysokoúrovňového rozhrania *Keras API* [10], ktoré minimalizuje množstvo používateľského úsilia pri implementácii aplikácií strojového učenia a má obsiahlu dokumentáciu a návody pre vývojárov. Modely z *Keras* sa dajú, okrem iného, exportovať do jazyka JavaScript, kde budú bežať priamo v prehliadači, alebo do *TensorFlow Lite* formátu, v ktorom sa dajú vkladať do aplikácií pre iOS a Android.

Zatiaľ čo *TensorFlow* je vrstva infraštruktúry pre diferencovateľné programovanie, ktorá sa zaoberá tenzormi, premennými a gradientami, *Keras* je používateľské rozhranie pre hlboké učenie, ktoré sa zaoberá vrstvami, modelmi, optimalizátormi, stratovými funkciami, metrikami a ďalšími. *Keras* slúži ako vysokoúrovňové API pre *TensorFlow*.

## 4.1 Vrstvy

Trieda vrstvy (`Layer`) [9] je základnou abstrakciou v *Keras*. Vrstva zapuzdruje stav a niektoré výpočty. Vrstva je jednoduchá transformácia, ako napríklad škálovanie alebo orezanie. Napríklad tu uvedená vrstva je lineárna projekcia, ktorá mapuje svoje vstupy do 16-rozmerného príznakového priestoru – `dense = keras.layers.Dense(units=16)`. Základné typy vrstiev, ktoré sa typicky používajú pri klasifikácií obrázkov, sú:

- Vstupná vrstva (`Input`).
- Plne prepojená vrstva (`Dense`) – implementuje operáciu
  `výstup = aktivačná_funkcia(skalárny_súčin(vstup, váhy) + bias)`.
- 2D konvolučná vrstva (`Conv2D`) – táto vrstva vytvorí konvolučné jadro, ktoré vykonáva operáciu 2D konvolúcie so vstupom.
- Vrstva dropout (`Dropout`) – Vrstva náhodne nastavuje vstupné vzorky na 0 s nastavenou frekvenciou, čo pomáha predísť preučeniu. Vstupné vzorky nenastavené na 0 sú kompenzačne zvýšené, aby sa súčet všetkých vstupov nezmenil.
- Združovacia vrstva (`MaxPooling, AveragePooling`) – Podvzorkuje vstupnú vrstvu pozdĺž jej priestorových dimenzií (výška a šírka) tým, že vezme zo vzorky vstupu v rámci okienka maximálnu, resp. priemernú hodnotu.

## 4.2 Model

Model je orientovaný acyklický graf vrstiev. Dá sa predstaviť ako väčšia vrstva, ktorá zahŕňa viacero podvrstiev a ktorá sa dá trénovať po poskytnutí dát [8].

### 4.2.1 Sekvenčný model

Jedným zo spôsobov ako zostaviť model v *Keras* je tzv. sekvenčný model. Sekvenčný model sa definuje ako zoznam jednotlivých vrstiev. Je použiteľný v prípade, že sieť má lineárnu topológiu a má práve jeden vstupný a práve jeden výstupný tenzor. Príklad definície jednoduchého modelu s tromi plne prepojenými vrstvami:

```
model = keras.Sequential([
    layers.InputLayer(input_shape=(16)),
    layers.Dense(32, activation="tanh", name="dense1"),
    layers.Dense(32, activation="tanh", name="dense2"),
    layers.Dense(32, activation="tanh", name="output")
])
```

Príkazom `model.summary()` dokážeme získať informácie o vytvorenom sekvenčnom modeli, vrátane počtu trénovateľných a netrénovateľných parametrov jednotlivých vrstiev aj celkovo. Výstupom volania metódy `summary()` na model definovaný vyššie je:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense1 (Dense)              (None, 32)                544
 dense2 (Dense)              (None, 32)                1056
 output (Dense)              (None, 32)                1056
=================================================================
Total params: 3,133
Trainable params: 3,133
Non-trainable params: 0
```

### 4.2.2 Trénovací proces

Typický proces učenia neurónovej siete pozostáva z:

- trénovania,
- priebežnej validácie predom odloženou časťou dát,
- evaluácie na testovacích dátach.

Model je pred trénovaním nutné zostaviť príkazom `model.compile()`, ktorého argumenty sú, okrem iných, používaný optimalizátor, stratová funkcia a metriky.

Celá dátová sada sa typicky rozdeľuje na tri časti – trénovacie dáta, validačné dáta a testovacie dáta. Sieť sa pri trénovaní snaží predpovedať štítok vstupných trénovacích dát, na základe úspešnosti predikcie získa spätnú väzbu a pomocou gradientného zostupu sa jej parametre upravia. Progresívne sa zlepšujúca sieť typicky vidí rovnaké trénovacie položky viackrát v priebehu svojho trénovania – počet prechodov celou dátovou sadou sa nazýva počet epoch.

Validačná časť dátovej sady slúži, okrem iného, na monitorovanie preučenia (*overfitting*). Z validačnej sady sieť nezískava spätnú väzbu, ale po každej epoche sa na nej vyhodnotí

úspešnosť modelu. Ak je úspešnosť na validačnej sade výrazne nižšia ako na trénovacej sade, pravdepodobne dochádza k preučeniu.

Trénovanie siete sa spustí príkazom `model.fit()` s nasledujúcimi argumentami – trénovacia a validačná sada, počet epoch, veľkosť dávky (koľko trénovacích položiek sa má evaluovať pred upravením parametrov modelu) a iné. Funkcia na výstup priebežne vypisuje stav siete v jednotlivých epochách – trénovaciu a validačnú stratu, trénovaciu a validačnú presnosť a iné zvolené metriky.

Pri trénovaní často chceme sledovať vyvíjajúce sa parametre a na ich základe ovplyvňovať trénovací proces alebo zaznamenávať údaje o trénovaní. *Keras* implementuje vo funkcií `model.fit()` argument `callback`, kde sa dá vložiť zoznam vlastných alebo knižnicou preddefinovaných funkcií, ktoré budú invokované počas trénovacieho procesu. Významnou funkciou je `EarlyStopping()`, ktorá slúži na predčasné zastavenie trénovania v prípade, že sa splní preddefinovaná podmienka, napríklad strata prestane klesať, alebo dosiahneme požadovanú presnosť. Tento argument taktiež môže slúžiť na zaznamenávanie trénovacích dát v čase a ich neskoršie využitie na analýzu trénovacieho procesu.

Vytrénovaný model sa s pomocou testovacej sady dá evaluovať príkazom `model.evaluate()`. Pri využití v praxi je často potrebné použiť model na klasifikáciu jednej samostatnej položky bez potreby zisťovať stratu a spájať položky do dávok. Na to slúži príkaz `model.predict()`, ktorý po poskytnutí vstupných dát dá na výstup vektor štítkov spolu s ich pravdepodobnosťami.

### 4.2.3 Siamské siete v *Keras*

Siamské siete [12] obsahujú dve alebo viac identických podsietí, ktoré spolu zdieľajú aktuálne hodnoty trénovateľných parametrov. Pri implementácií je nutné si najprv vytvoriť základnú sieť (*embedding network*) a následne vrstvu, ktorá tieto siete bude spájať. V prípade kontrastívneho učenia s trojicami vstupov bude táto vrstva počítať kontrastívnu stratu medzi výstupmi troch identických podsietí.

*Keras* sekvenčný model nedokáže implementovať nelineárnu topológiu siamskej siete. Alternatívou je použitie funkcionálneho API, ktoré umožňuje vytvárať flexibilnejšie modely, ktoré môžu obsahovať nelineárnu topológiu, zdieľané vrstvy a viacero vstupných alebo výstupných vrstiev. Príklad definície siamskej siete na kontrastívne učenie:

```
# extraktor priznakov, ktory ma linearnu topologiu a moze byt definovany aj sekvencnym API
embedding = keras.Sequential(...)

# definicia troch vstupnych vrstiev pre kazdy prvok kontrastivnej trojice
anchor_input = layers.Input(name="anchor", shape=(34))
positive_input = layers.Input(name="positive", shape=(34))
negative_input = layers.Input(name="negative", shape=(34))

# vytvorenie topologie troch identickych extraktorov priznakov na konci spojenych vrstvou
# vypocitavajucou vzdialenost prvkov trojic, z ktorych sa nasledne rata kontrastivna strata
distances = DistanceLayer()(
    embedding(anchor_input),
    embedding(positive_input),
    embedding(negative_input)
)

siamese_network = Model(
    inputs=[anchor_input, positive_input, negative_input], outputs=distances
)
```

# Kapitola 5

# Spracovanie dátovej sady

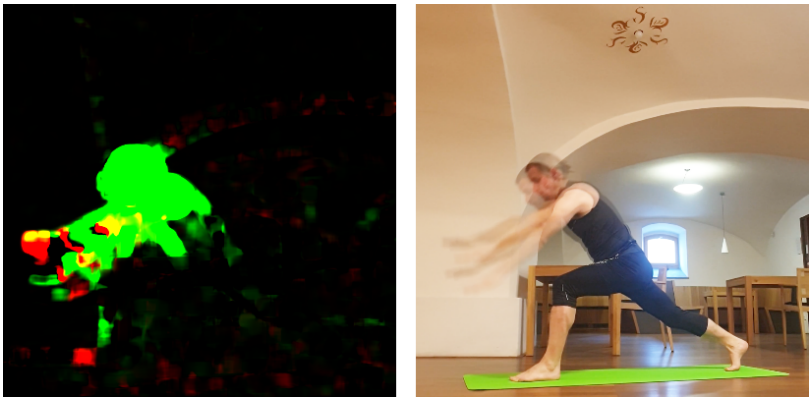
Nespracovaná dátová sada pozostáva z 234 videí rôznej dĺžky (1 – 20 min). Každé z videí má rozlíšenie $1440 \times 1440$ pixelov a frekvenciu 20 snímok za sekundu. Každé z jednotlivých cvičení jogy bolo zaznamenávané z 2 až 4 staticky umiestnených kamier z rôznych uhlov, pričom výška kamery bola v rozsahu niekoľko desiatok centimetrov od zeme až do výšky očí cvičiaceho. Kamera stále mierila na ventrálnu polrovinu cvičiaceho. Videá boli natáčané na rôznych miestach v rámci jednej miestnosti, s rôznym pozadím, osvetlením a farbou oblečenia cvičiaceho. Niekoľko vzoriek z dátovej sady sa nachádza na obrázku 5.1.

Obr. 5.1: Ukážka snímok z videí dátovej sady.

## 5.1 Synchronizácia videí

Videá jedného cvičenia zaznamenané z rôznych uhlov bolo pred ďalším spracovaním nutné synchronizovať. Synchronizácia by mohla prebiehať ručne, avšak to by bolo časovo náročné

oproti programovému spracovaniu. Fakt, ktorý môžeme zužitkovať je, že zjavné množstvo pohybu pri cvičení je, až na malé výnimky, z každého uhla približne rovnaké. Jednou z metrík na kvantifikovanie pohybu v obraze je optický tok (*optical flow*), ktorý je vizualizovaný na obrázku 5.2.

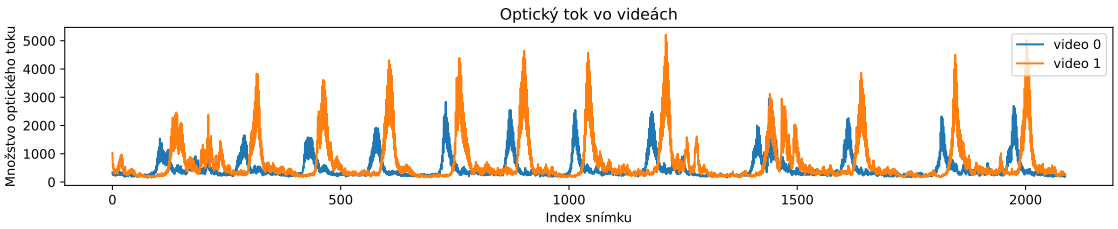


Obr. 5.2: Vizualizácia optického toku v 2D priestore vypočítaného z dvoch po sebe idúcich snímok videa. Zelený farebný kanál reprezentuje vertikálny pohyb, červený kanál horizontálny.

Optický tok sa vypočítava z dvoch po sebe idúcich snímok s rozmermi $X \times Y$ pixelov a udáva odhad vektora pohybu každého pixelu ($tok_x$ a $tok_y$ obsahujú horizontálnu a vertikálnu zložku vektora v rozsahu -1 až 1). Agregáciou dát z každého pixelu sa získa optický tok daného videa pre daný časový okamih (5.1).

$$M = \sum_{y=0}^{Y-1} \sum_{x=0}^{X-1} \sqrt{tok_x[x,y]^2 + tok_y[x,y]^2} \tag{5.1}$$

Ak tento proces opakujeme pre každú dvojicu po sebe idúcich snímok, dostaneme signál (obr. 5.3), ktorý reprezentuje optický tok v čase pre dané video.




Obr. 5.3: Množstvo optického toku v čase jedného cvičenia z dvoch rôznych uhlov. Odchýlka videí by sa z obrázka dala, očividne, odhadnúť aj ručne. Vidíme, že cvičiaci v periodických intervaloch menil pozíciu, v ktorej sa následne malú chvíľu nehýbal.

Po vypočítaní optického toku pre každé video sa nad párom signálov spočíta nevychýlený odhad korelačných koeficientov (5.2). V prípade viac ako dvoch videí sa každé video koreluje s prvým videom. Index najväčšieho prvku vektora korelačných koeficientov udáva najpravdepodobnejšie posunutie daného videa oproti prvému videu.

$$\hat{R}[k] = \frac{1}{N - |k|} \sum_{n=0}^{N-1} x[n]x[n+k] \tag{5.2}$$

Pre prípadné ďalšie anotovanie sa videá pomocou nástroja `ffmpeg`[1] upravia, aby začínali v rovnaký moment a mali rovnakú dĺžku.

## 5.2 Extrakcia neoznačených snímok z videí

Na vytvorenie kontrastívnych trojíc je potrebné zo synchronizovaných videí extrahovať bázový, negatívny a pozitívny obrázok. Bázový a negatívny obrázok pochádzajú z rovnakého videa, ale v rôznom čase. Naopak bázový a pozitívny obrázok pochádzajú z rôznych videí (uhlov pohľadu na cvičenie), ale sú v rovnakom čase, a teda reprezentujú rovnakú pozíciu a mali by byť blízko seba v latentnom priestore.

Je evidentné, že aj napriek tomu, že je medzi snímkami z jedného videa časový rozostup, tak sa môže stať, že sa cvičiaci v medzičase nepohol, čo je v prípade cvičenia jogy veľmi časté – preto potrebujeme nejakým spôsobom kvantifikovať pohyb a vyberať úseky, kde je jeho množstvo veľké.

Na tento účel nám poslúži, rovnako ako pri synchronizácií, optický tok. Vypočítaný tok z predošlého kroku stačí posunúť pre každé video o hodnotu zistenú v predošlom kroku, aby boli synchrónne. Toky jednotlivých videí spriemerujeme a na získanom signáli hľadáme úseky, kde je po dlhšiu dobu jeho hodnota vysoká (obr. 5.4).

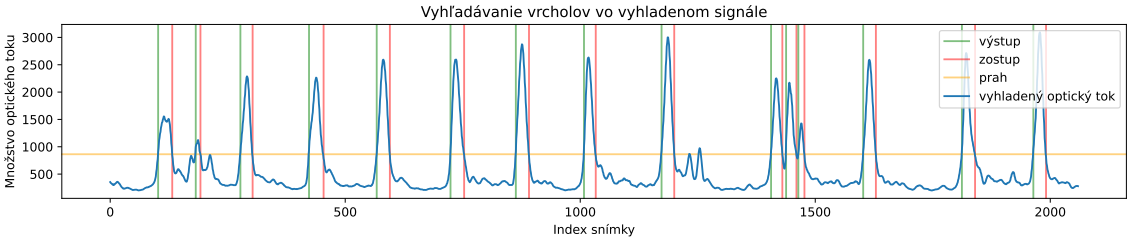
Pred hľadaním vrcholov sa na signál aplikuje dolnopriepustný (*low-pass*) filter, realizovaný pomocou operácie konvolúcie, aby bol signál menej zašumený. Následne sa určí prahová hodnota, ktorá bude oddeľovať vrcholy od zvyšku signálu. Experimentálne zistená hodnota, ktorá fungovala dobre, bola 1,4-násobok priemernej hodnoty signálu.

Okrem minimálnej výšky musia vrcholy mať aj dostatočnú šírku. Algoritmus na určenie vrcholov a ich šírky je nasledovný:

```python
# vstupom je vyhladeny 1D signal
# vystupom su dva zoznamy udavajuce indexy snimok zaciatkov a koncov vrcholov
def findPeaks(smoothedFlow):
    ascents = []
    descents = []

    # ak je prva vzorka nad prahom, zacina vrchol na prvej vzorke
    if smoothedFlow[0] >= flowThreshold:
        ascents.append(0)

    for i in range(len(smoothedFlow) - 1):
        # ak je aktualna vzorka pod prahom a nasledujuca nad, zacina vrchol
        if smoothedFlow[i] < flowThreshold and smoothedFlow[i + 1] >= flowThreshold:
            ascents.append(i + 1)
        # ak je aktualna vzorka nad prahom a nasledujuca pod, konci vrchol
        elif smoothedFlow[i] >= flowThreshold and smoothedFlow[i + 1] < flowThreshold:
            descents.append(i + 1)

    # ak je posledna vzorka nad prahom, konci vrchol na poslednej vzorke
    if smoothedFlow[len(smoothedFlow) - 1] >= flowThreshold:
        descents.append(len(smoothedFlow) - 1)

    return ascents, descents
```

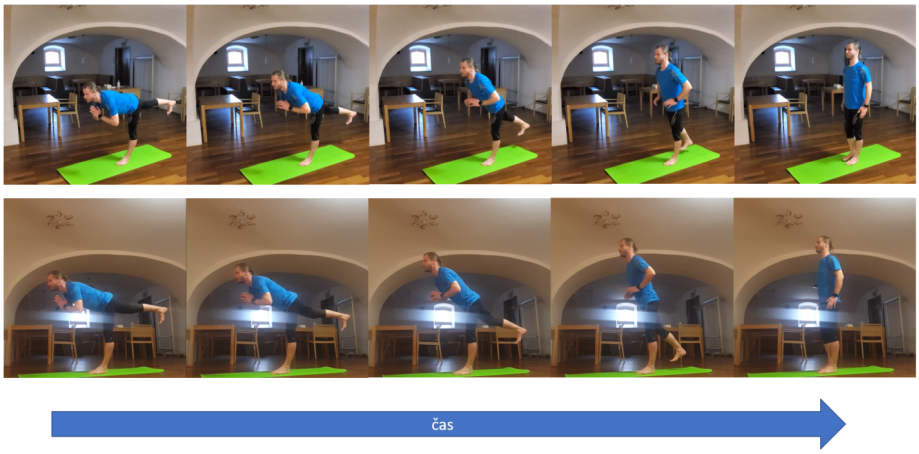
[1] https://ffmpeg.org/

Odčítaním indexu snímku začiatku vrcholu od konca získame šírku vrcholu. Vrcholy, ktoré sú príliš úzke, môžeme vylúčiť.




Obr. 5.4: Označené vrcholy (medzi zelenou a červenou čiarou) vo vyhladenom signáli.

Minimálna časová vzdialenosť medzi bázovým a negatívnym obrázkom bola nastavená na 10 snímok, čo reprezentuje 500 milisekúnd. Po celej šírke vrcholu sa zo všetkých videí extrahujú snímky s krokom 10 snímok (obr. 5.5). Trojice sa vyberajú tak, aby pozitívna a bázová snímka mali rovnaký čas a aby negatívna a bázová snímka boli z rovnakého videa (obr. 5.6 a 5.7).




Obr. 5.5: Sekvencia snímok s rozostupom 500 milisekúnd z dvoch uhlov záznamu jedného cvičenia v mieste s vysokým optickým tokom.

## 5.3 Extrakcia označených snímok z videí

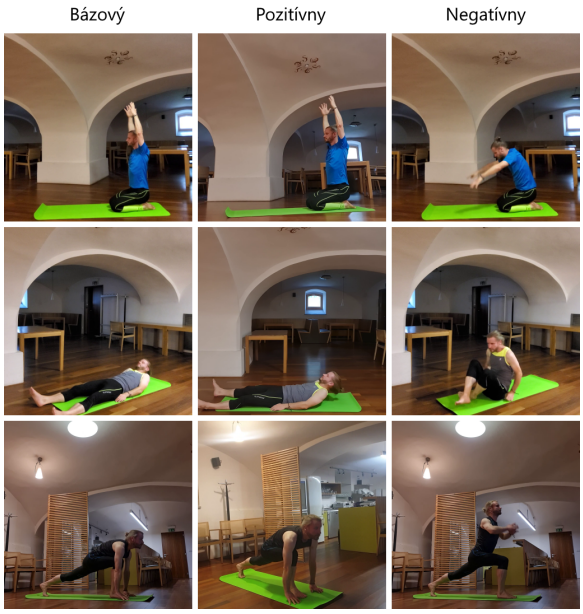
Na anotovanie dát bol použitý webový nástroj[2], zobrazený na obrázku 5.8, ktorý umožňuje videá nahrané na Google Disk načítať. Webový nástroj bol vytvorený mnou v rámci predošlého projektu, ako vylepšenie anotačného nástroja z práce *Detection of a Yoga Poses in Image* [18]. Medzi vylepšenia patrí možnosť anotovať viacero videí jedného cvičenia naraz, ručne synchronizovať videá (v tejto práci sa táto funkcionalita nevyužíva, lebo sa videá synchronizujú automaticky skriptom `extractLabelled.py` s využitím optického toku) a načítavať videá z Google Disku, čo umožňuje jednoduchšiu integráciu s ďalšími nástrojmi.

[2]Dostupný na online na `http://video-anno.herokuapp.com/` s otvoreným zdrojovým kódom na `https://github.com/leSamo/Video-Annotation`.

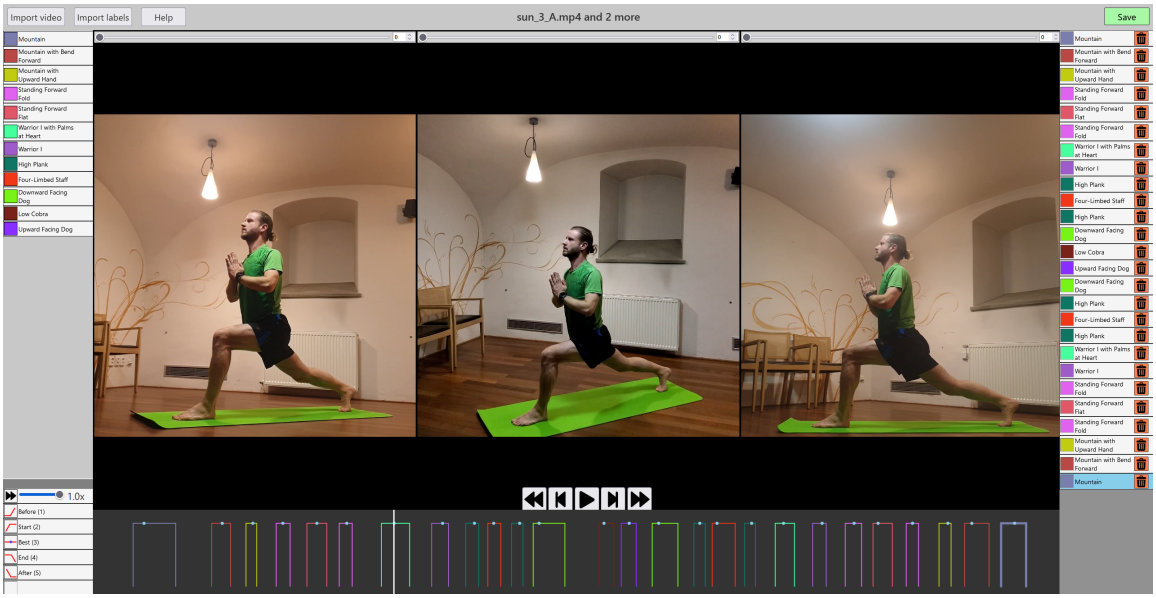

Obr. 5.6: Spôsob vyberania kontrastívnych trojíc zo sekvencií.

Po zvolení vstupných videí z Google Disku a načítaním JSON súboru s definovanými pózami sa jednotlivým pózam udeľuje štítok a trvanie od a do. Podrobnejší manuál k použitiu aplikácie sa nachádza v prílohe B. Aplikácia na konci pre každé video vygeneruje JSON súbor s anotáciami.

JSON súbor s anotáciami spolu s videom sú vstupom skriptu `extractLabelled.py`, ktorý na základe anotácií extrahuje z videa obrázky jogových pozícií a tie potriedi podľa označenej pozície do priečinkov. Tieto priečinky s obrázkami sú v správnom tvare na použitie pri trénovaní a evaluovaní siete pomocou ďalších skriptov. Z každej anotácie vzniknú tri snímky – jedna z času daného začiatkom anotácie, ďalšia z času presného stredu a tretia z času konca anotácie. V priebehu jednej anotácie sa síce úmyselne cvičiaci nehýbe, ale mierne kolísanie nastať môže, a to hlavne pri náročnejších pózach (obr. 5.9).




Obr. 5.7: Príklad troch trojíc vybraných z trénovacej dátovej sady.




Obr. 5.8: Ukážka webového anotátora, v ktorom sa nachádza jedno cvičenie zaznamenané z troch rôznych uhlov. V dolnej časti obrazovky sa nachádza časová os, na ktorej sú zaznačené anotácie.

Obr. 5.9: Tri snímky jedného vykonávania polohy preložené cez seba. Táto póza vykonávaná na jednej nohe je menej stabilná, a preto sa v priebehu jej realizácie mierne menia polohy kľúčových bodov. Z tohto dôvodu sa oplatí z jednej realizácie pozície extrahovať viac ako jeden snímok.

# Kapitola 6

# Návrh a implementácia neurónovej siete

Výsledné riešenie je implementované v jazyku Python s využitím knižníc *OpenCV* a *TensorFlow*. Pri implementácií boli časti kódu prebraté z dokumentácie ku knižnici *Tensorflow* [1] a *Keras* [8, 9, 10, 12].

Hotový model *MoveNet Lightning*, používaný na detekciu kľúčových bodov tela z obrázkov, je importovaný knižnicou *TensorFlow Hub*. Extrakcia kľúčových bodov spolu s augmentáciou obrázkov je súčasťou fázy predspracovania obrázkov. Do trénovania a evaluácie neurónovej siete vstupujú už iba kľúčové body tela.

Definuje sa kódovacia sieť (*encoder, embedding network*), ktorá má lineárnu topológiu a jej úlohou je z kľúčových bodov získať príznakový vektor, ktorý by mal byť pre rovnakú jogovú pozíciu približne rovnaký a ktorý bude slúžiť na neskoršiu klasifikáciu. Tri inštancie tejto siete, ktoré zdieľajú trénovateľné parametre, sú na konci spojené vzdialenostnou vrstvou, čím vytvárajú siamskú sieť. Vzdialenostná vrstva z troch príznakových vektorov podsietí ráta kontrastívnu stratovú funkciu (sekcia 3.2.3).

Kódovacia sieť, ktorej parametre boli natrénované kontrastívnym učením, je schopná lepšie charakterizovať pozíciu z obrázka oproti výstupu zo siete *MoveNet*, ktorý má čisto 2D informáciu. V tomto bode sieť dokáže rovnakým pózam dávať podobné ohodnotenie, ale nedokáže klasifikovať. V druhej etape trénovania sa na kódovaciu sieť zozadu napojí jednoduchý klasifikátor, ktorého výstupom je vektor pravdepodobností jednotlivých jogových póz. Parametre kódovacej siete sa zmrazia a trénuje sa čisto iba klasifikátor. Vďaka tomu, že kódovacia sieť dáva klasifikátoru užitočnejšie informácie oproti kľúčovým bodom v 2D priestore, vyžaduje klasifikátor pri trénovaní iba veľmi málo trénovacích dát.

## 6.1 Predspracovanie obrázkov a extrakcia kľúčových bodov

Pred použitím vstupných obrázkov pri trénovaní je nutné ich predspracovať. Predspracovanie pozostáva z nasledujúcich krokov:

- augmentácia vstupných dát,
- výpočet kľúčových bodov z obrázka,
- výpočet obdĺžnikovej obálky (*axis-aligned minimum bounding box, AABB*) z kľúčových bodov cvičiacej osoby z obrázka,

- orezanie obrázka na základe obálky,
- výpočet kľúčových bodov z orezaného obrázka.

Opakované spracovanie obrázkových dát pri iteratívnom vývoji a experimentovaní so sieťou zaberá veľa času, a preto sa extrahované a augmentované kľúčové body zálohujú do binárneho súboru pomocou funkcie `np.save()` z knižnice *NumPy*, odkiaľ sa pri nasledujúcom spustení načítajú pomocou `np.load()`.

### 6.1.1 Augmentácia obrázkov

Často používanými [6, 7, 14] augmentáciami pri učení neurónových sietí klasifikujúcich obrázky bývajú náhodné horizontálne preklopenie, náhodné orezanie, náhodná zmena obrázka do odtieňov sivej a náhodná zmena jasu, kontrastu a saturácie.

V práci bolo experimentálne zistené, že najväčší vplyv na úspešnosť modelu má použitie náhodného horizontálneho preklopenia. Či už išlo o použitie pri trénovaní extraktora príznakov (extraktor sa naučí, že horizontálne preklopenie nemení sémantickú informáciu, keďže každá jogová pozícia rozpoznávaná v tejto práci je symetrická podľa sagitálnej roviny tela), alebo pri trénovaní klasifikátora, kedy nepreklopený a preklopený obrázok sú uvedené ako dva varianty danej pozície.

Použitie náhodného orezu je v prípade detekcie pozície problematické, a to hneď z dvoch dôvodov. Sieť na extrakciu kľúčových bodov tela *MoveNet* dokáže pracovať iba s obrázkami, kde je vidno celú osobu. *MoveNet* stále predpokladá, že na obrázku je osoba celá a preto sa pokúsi chýbajúce časti tela v obraze nájsť aj keď tam nie sú, čo má za následok nesprávnu detekciu. Druhým problémom, ktorý sťažuje detekciu pozície oveľa viac ako napríklad klasifikáciu druhov zvierat, je to, že je takmer nemožné odvodiť pozíciu celého tela z orezu, na rozdiel od klasifikácie zvierat, kde sa v náhodnom oreze nachádza oveľa viac sémantickej informácie – farba, textúra, obrys, atď.

Náhodné otočenie obrázka o niekoľko stupňov je invariantné sémantickej informácii a preto sa dá použiť na obohatenie dátovej sady a prevenciu preučenia. Konkrétne použité hodnoty otočenia sú v rozsahu -10° až 10°. Centrálnym bodom otáčania je stred obrázka a vzniknuté diery v obrázku sú vyplnené zrkadlením priľahlých pixelov.

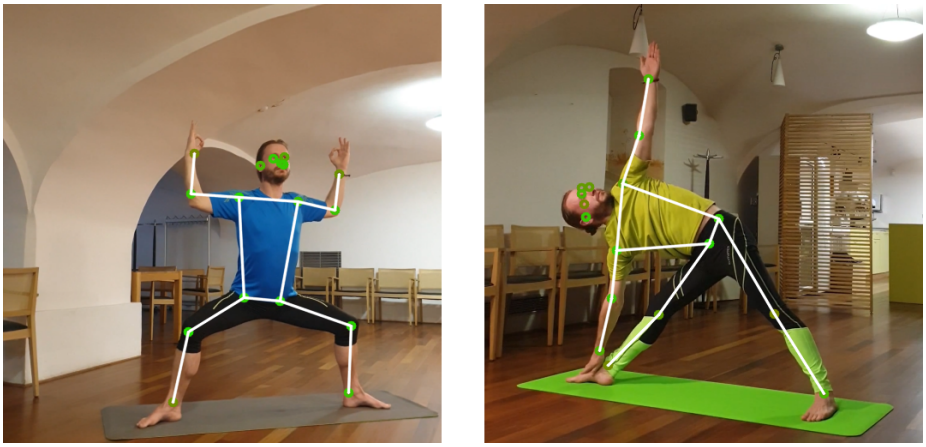
Farebné augmentácie (prevod obrázku do odtieňov sivej, zmena jasu, kontrastu a saturácie) neboli použité, keďže by extraktoru kľúčových bodov nijak nepomohli, práve naopak, detekcia kľúčových bodov by bola o niečo náročnejšia, čo bolo potvrdené aj experimentálne. Extraktor z dát odstráni informáciu o farbe, takže sa nepropaguje ďalej a nemá ako inak ovplyvniť trénovací proces.

Inšpiráciou k použitiu veľmi malých náhodných transformácií kľúčových bodov po osi $x$ aj $y$ bol projekt *YogAI: Smart Personal Trainer* [20]. V tejto práci sa používajú náhodné posuny všetkých bodov rovnakým smerom aj mierne náhodné posuny každého bodu iným smerom, čo typicky nezmení význam pózy a obohatí trénovaciu sadu.

### 6.1.2 Výpočet kľúčových bodov

Spracovaním augmentovaného obrázka sieťou *MoveNet* získame 14 kľúčových bodov tela cvičiaceho. Každý bod obsahuje súradnice $x$ a $y$ normalizované do rozsahu 0 až 1 v rámci 2D priestoru obrázka a taktiež hodnotu spoľahlivosti (*confidence*), ako si je sieť *MoveNet* polohou istá. Hodnota spoľahlivosti sa v ďalších výpočtoch nepoužíva. Na ladiace účely je v projekte priložený skript `skeleton.py`, ktorý každý obrázok v zadanom vstupnom

priečinku evaluuje sieťou *MoveNet* a uloží kópiu obrázka do výstupného priečinka s vykreslenými kľúčovými bodmi (hodnota istoty je reprezentovaná farbou bodu na škále medzi zelenou a červenou, pričom zelená značí vysokú spoľahlivosť), kosťami (biele čiary spájajúce kľúčové body) a obdĺžnikovou obálkou. Ukážka označenia bodov a kostí sa nachádza na obrázku 6.1.



Obr. 6.1: Obrázok jogových póz s označenými kľúčovými bodmi a kosťami získanými ladiacim skriptom `skeleton.py`. Môžeme vidieť, že väčšina bodov má sýtu zelenú farbu, čo značí, že si je sieť istá, že určila polohu bodov správne. V prípade, že nejaký bod splýva s pozadím alebo je z pohľadu kamery skrytý, sieť typicky nastavuje hodnotu spoľahlivosti na nízku hodnotu.

### 6.1.3 Získanie obdĺžnikovej obálky

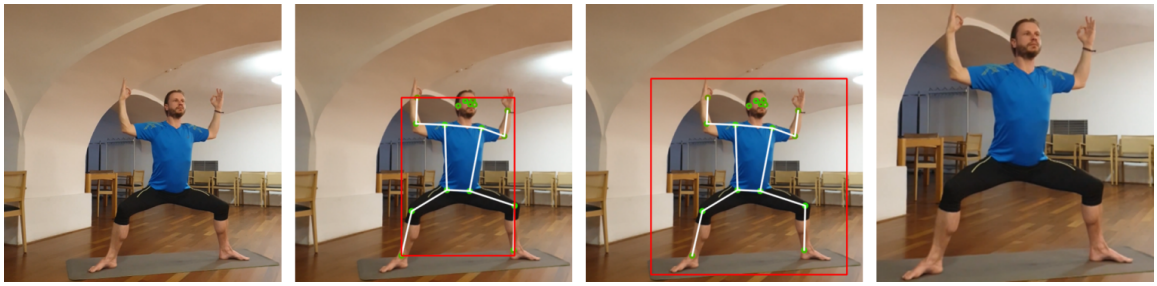
Vzhľadom na to, že *MoveNet* akceptuje vstupný obrázok o veľkosti $192 \times 192$ pixelov a to, že cvičiaci v obrázkoch je často ďaleko od kamery a nevypĺňa efektívne celý 2D priestor snímky, by sa oplatilo snímku najprv orezať.

To, ktorú časť obrázka môžeme odrezať tak, aby cvičiaci ostal v obraze, môžeme určiť tým, že neorezanú snímku evaluujeme pomocou siete *MoveNet* a okolo získaných bodov nakreslíme obdĺžnikovú obálku. Tento obdĺžnik je potrebné o niekoľko percent expandovať do všetkých štyroch strán, inak by, napríklad, nebolo vidno chodidlá (najnižšie kľúčové body sú členky). Ohraničenie v tvare obdĺžnika rozšírime na štvorec a podľa neho orezaný obrázok vložíme znovu do siete *MoveNet* a výsledné kľúčové body využívame v nasledujúcich krokoch (obr. 6.2 a 6.3).

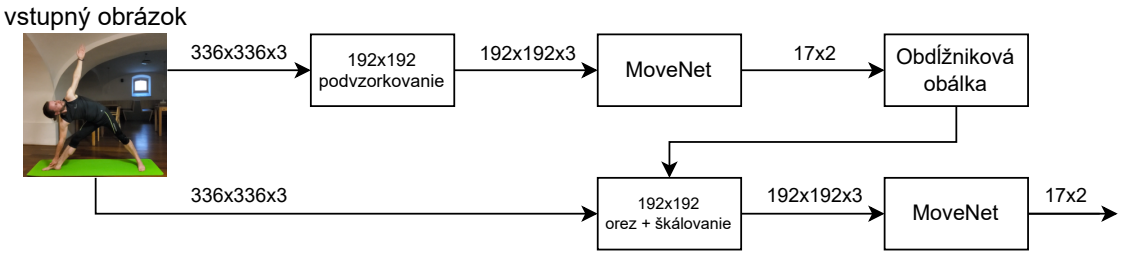
Táto metóda má oproti jednému prechodu sieťou *MoveNet* až dve výhody – prvou je, že *MoveNet* v druhom prechode dokáže presnejšie určiť vzájomnú polohu jednotlivých kľúčových bodov, lebo cvičiaci zaberá väčšiu plochu na vstupnom obrázku; druhou výhodou je to, že pozície kľúčových bodov budú v rámci obrázka lepšie normalizované.

## 6.2 Štruktúra modelu

V prvej fáze trénovania sa používa siamská sieť zložená z troch inštancií extraktora príznakov a vzdialenostnej vrstvy. V druhej trénovacej etape, a zároveň aj pri testovaní, sa používa sieť s lineárnou topológiou zložená z jednej, už natrénovanej, inštancie extraktora príznakov a klasifikačnej siete.



Obr. 6.2: Jednotlivé kroky orezania obrázku. Na prvom obrázku sa nachádza pôvodný obrázok, v ďalšom sú označené kľúčové body a ich obdĺžniková obálka. Následne sa obálka rozšíri o niekoľko percent do všetkých strán a doplní na štvorec. Na poslednom obrázku sa nachádza orezaný obrázok, ktorého kľúčové body sa znovu zistia a používajú v procese trénovania.




Obr. 6.3: Diagram dvojitého spracovania obrázka sieťou *MoveNet* s cieľom sémantického orezu.

### 6.2.1 Extraktor príznakov

Vstupom extraktora príznakov (kódovej siete) je tenzor s 34 premennými (17 kľúčových bodov, ktorých význam je definovaný v sekcii 3.4, každý má dve súradnice $x$ a $y$) normalizovanými do rozsahu 0 až 1. Tento tenzor s veľkosťou $17 \times 2$ sa splošťovacou vrstvou pretransformuje na vektor s veľkosťou 34. Jednodimenzionálny vektor môže byť následne spracovaný štvoricou plne prepojených vrstiev s aktivačnou funkciu `tanh`. Posledná plne prepojená vrstva má 128 výstupov a tieto výstupy reprezentujú zobrazenie vstupu do latentného priestoru.

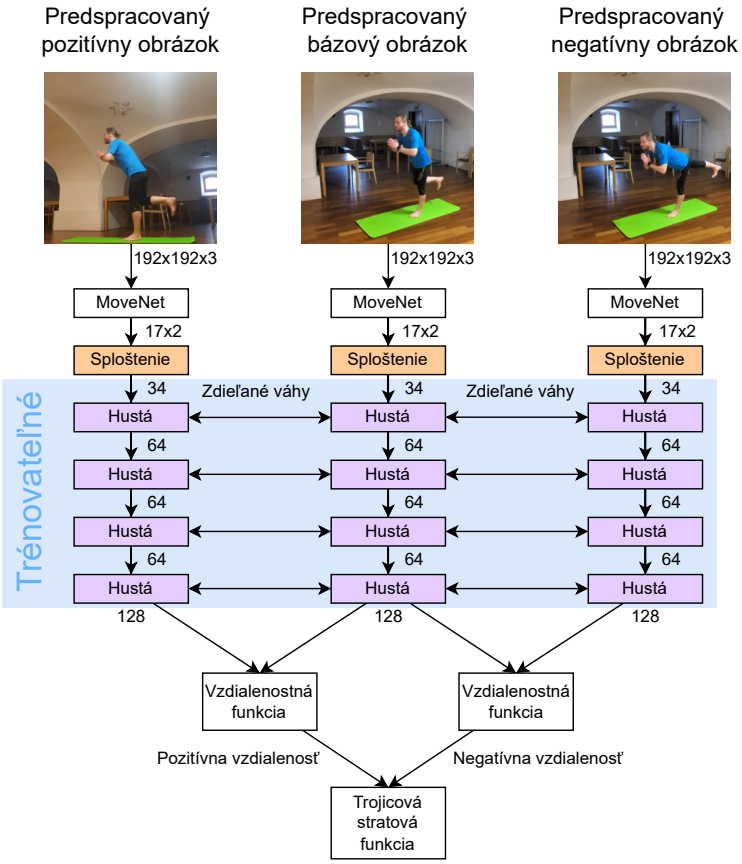
### 6.2.2 Siamská sieť

Trojica kódovacích sietí je prepojená vzdialenostnou vrstvou, ktorá slúži na počítanie pozitívnej a negatívnej vzdialenosti medzi prvkami kontrastívnych trojíc. Nová vrstva v *Keras* sa definuje ako trieda rozširujúca bázovú triedu `tf.keras.layers.Layer`. Pri implementácií vlastnej vrstvy je okrem konštruktoru nutné definovať iba metódu `call()`, ktorá transformuje vstupy (parametre metódy) na výstupy (návratová hodnota metódy).

Konkrétne implementované vzdialenostné vrstvy sú `CosineDistanceLayer`, ktorá ráta kosínusovú vzdialenosť (3.5) a `EuclideanDistanceLayer` rátajúcu euklidovskú vzdialenosť pre pozitívny a negatívny pár kontrastívnych trojíc.

Trojica inštancií extraktora prepojená zvolenou vzdialenostnou vrstvou je obalená do inštancie triedy `SiameseModel`. Táto trieda rozširuje bázovú triedu `tf.keras.Model` o implementácie metód umožňujúce kontrastívne učenie. Konkrétne ide o implementáciu metódy

`_compute_loss()`, aby sieť dokázala z pozitívnej a negatívnej vzdialenosti získanej z poslednej (vzdialenostnej) vrstvy vypočítať stratu. Keďže typický model pri trénovaní očakáva vstupné dáta na evaluáciu a zároveň aj štítok, musíme predefinovať metódu `train_step()`, kde nastavíme, že nami definovaná funkcia `_compute_loss()` bude rátať stratu namiesto typicky používanej stratovej funkcie vyžadujúcej štítky ako napríklad MSE. Jednotlivé vrstvy použité na zostavenie siamskej siete sú viditeľné na obr. 6.4.




Obr. 6.4: Diagram siamskej siete, ktorá bude použitá na trénovanie kódovej siete. Vzdialenostná funkcia je realizovaná euklidovskou alebo kosínusovou vzdialenostnou vrstvou.

### 6.2.3 Klasifikátor

Klasifikátor slúži na zobrazenie bodu zo 128-dimenzionálneho latentného priestoru na štítok. Pozostáva z troch plne prepojených vrstiev oddelených vrstvou typu dropout, ktorá náhodne nastavuje vstupy nasledujúcej vrstvy na 0 s pravdepodobnosťou 50 %, aby predišla preučeniu. Aktivačnou funkciou poslednej vrstvy je softmax, takže výstup klasifikátora je normalizovaný a spĺňa vlastnosti rozdelenia pravdepodobností jednotlivých štítkov.

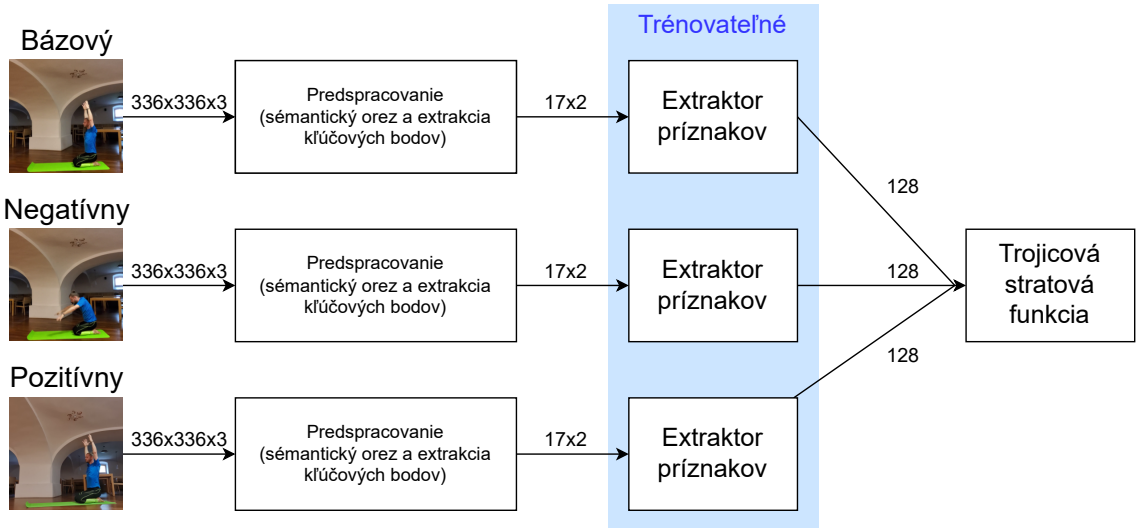
## 6.3 Pomocná úloha

Cieľom pomocnej úlohy je naučiť kódovaciu sieť zobrazovať rovnaké pozície na rovnaký bod v latentnom priestore. Táto časť je čisto self-supervised, a teda ako kontrolný signál pri

učení sa využívajú informácie obsiahnuté v štruktúre dát. Vo fáze extrakcie dát z videí boli pripravené trojice obrázkov, pričom informácia o tom, ktorý z trojice je bázový, negatívny a pozitívny vytvára jediné kontrolné signály, ktoré sa v tejto fáze využívajú.

Pri trénovaní sa každý z trojice obrázkov evaluuje svojou podsieťou siamskej siete a zo vzdialenosti príznakových vektorov sa vypočíta trojicová strata (obr. 6.5). Parametre procesu trénovania riadi optimalizátor `Adam` s počiatočnou mierou učenia (*learning rate*) $10^{-4}$. Počet epoch trénovania nie je fixný, ale je monitorovaný funkciou skorého zastavenia, ktorý trénovací proces ukončí, keď deteguje, že hodnota validačnej stratovej funkcie prestala klesať. Trojice sú rozdelené na trénovaciu a validačnú časť v pomere 4 : 1. Použitá veľkosť dávky je 32, čo bol dobrý kompromis medzi stabilitou stratovej funkcie a rýchlosťou trénovania.

V tejto fáze sieť nemá žiadne informácie o tom, aké pozície sa chystajú byť klasifikované. Dôležitým hyperparametrom, ktorý udáva dimenzionalitu latentného priestoru je veľkosť príznakového vektora. Experimentálne zistená vhodná hodnota je 128. To je zároveň počet premenných, koľkými spolu interagujú kódovacia a klasifikačná časť siete.




Obr. 6.5: Diagram trénovania siete pomocnou úlohou s neoznačenými dátami s využitím siamskej siete, ktorá zahŕňa tri identické trénovateľné extraktory príznakov a ktorá používa kontrastívnu stratovú funkciu.

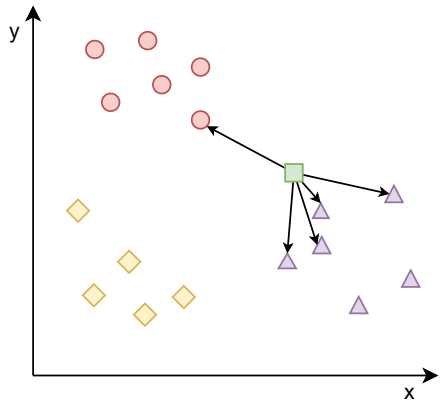
## 6.4 Dolaďovanie siete

Pri dolaďovaní siete (obr. 6.7) sa spolieha na to, že sa kódovacia sieť naučila dostatočne dobre reprezentovať podobné pozície na body blízko seba v latentnom priestore. Na základe toho táto trénovacia fáza vyžaduje výrazne menej označených dát oproti spôsobu, kedy by sa klasifikovalo čisto na základe kľúčových bodov.

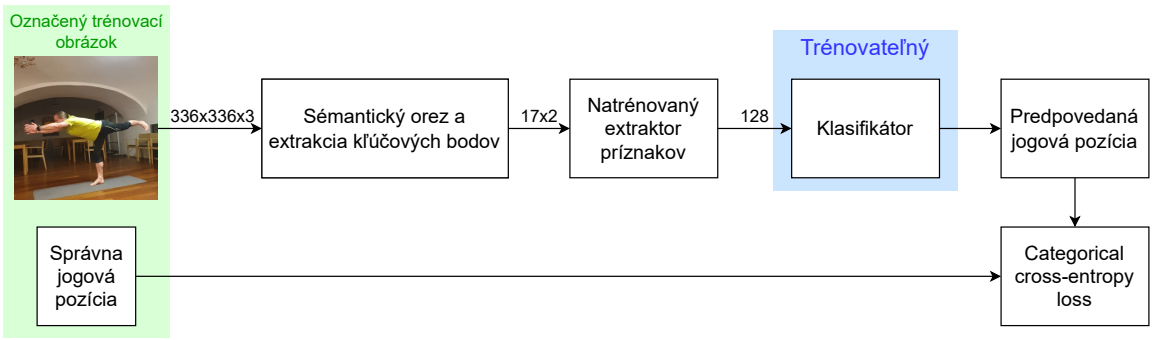
V tomto kroku sa k jednotlivým zhlukom v latentnom priestore priradia štítky. Dva spôsoby ako to dosiahnuť sú využitie zhlukovania podľa algoritmu k-najbližších susedov (*k-nearest neighbors*, obr. 6.6) alebo využitie klasifikačnej siete. Na základe experimentálnych výsledkov bol zvolený prístup využívajúci klasifikačnú sieť.

Výstupom klasifikátora je sieťou odhadovaný vektor pravdepodobnosti jednotlivých pozícií. Ako stratová funkcia je použitá funkcia *Categorical crossentropy*. Trénovací proces

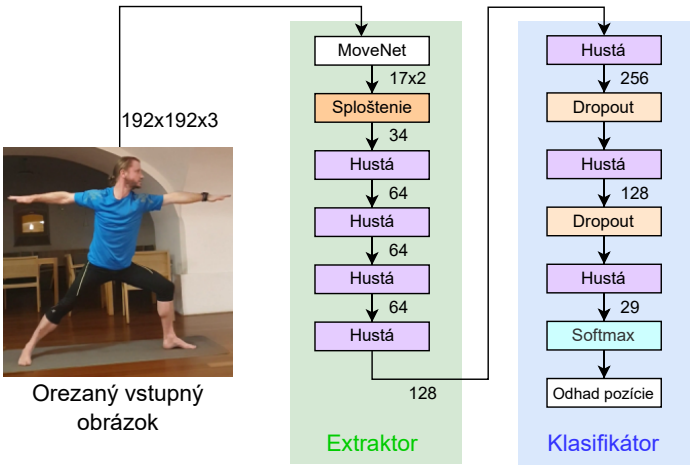
riadi optimalizátor `Adam` s počiatočnou mierou učenia $10^{-5}$ a veľkosťou dávky 16. Rovnako ako pri prvej fáze trénovania nie je počet epoch fixný, ale proces je zastavený funkciou skorého zastavenia, keď deteguje, že stratová funkcia začala stagnovať. Jednotlivé vrstvy extraktora a klasifikátora sú zobrazená na obr. 6.8.




Obr. 6.6: Ukážka zhlukovania v 2D priestore podľa algoritmu k-najbližších susedov pre $k = 5$. Červenou, žltou a fialovou farbou sú vyznačené tri rôzne zhluky bodov. Zelenou je označený bod, ktorý chceme priradiť k jednému zo zhlukov. Hodnota $k$ udáva počet najbližších evaluovaných susedov, takže sa zistí štítok piatich bodov najbližších k zelenému bodu (označené šípkou). Keďže väčšina najbližších bodov patrí k fialovému štítku, je aj evaluovaný bod klasifikovaný fialovým štítkom. V našom kontexte by body reprezentovali príznakové vektory jednotlivých obrázkov a priestor by mal 128 dimenzií namiesto dvoch.




Obr. 6.7: Diagram dolaďovacej fázy, kedy sa zmrazia parametre extraktora príznakov a označenými dátami sa trénuje jednoduchý klasifikátor, ktorého vstupom je výstup extraktora príznakov.




Obr. 6.8: Jednotlivé vrstvy siete použitej pri dolaďovacej etape, pozostávajúcej z dvoch častí – extraktora príznakov a klasifikátora.

# Kapitola 7

# Experimenty

Pri experimentovaní bol skúmaný vplyv množstva označených trénovacích dát a rôznych augmentácií na presnosť modelu v klasifikácií správnej jogovej pozície z 29-tich možných. Okrem úspešnosti modelu bolo sledované, s akými konkrétnymi obrázkami mal model problém.

## 7.1 Použitá dátová sada

Celá použitá dátová sada (tab. 7.1) pozostáva z 54 cvičení nahrávaných z dvoch až štyroch uhlov, dokopy 177 videí. Cvičenia boli dopredu rozdelené na neoznačenú trénovaciu sadu, označenú trénovaciu sadu a testovaciu sadu.

Neoznačená sada je zložená z 2771 kontrastívnych trojíc extrahovaných z 16-tich cvičení, spolu 58 videí, ktoré neboli anotované. Trojice boli extrahované z videí v miestach s veľkým množstvom optického toku, medzera medzi bázovou a negatívnou vzorkou je 500 ms. Trojice sú v pomere 4 : 1 rozdelené na trénovaciu a validačnú časť.

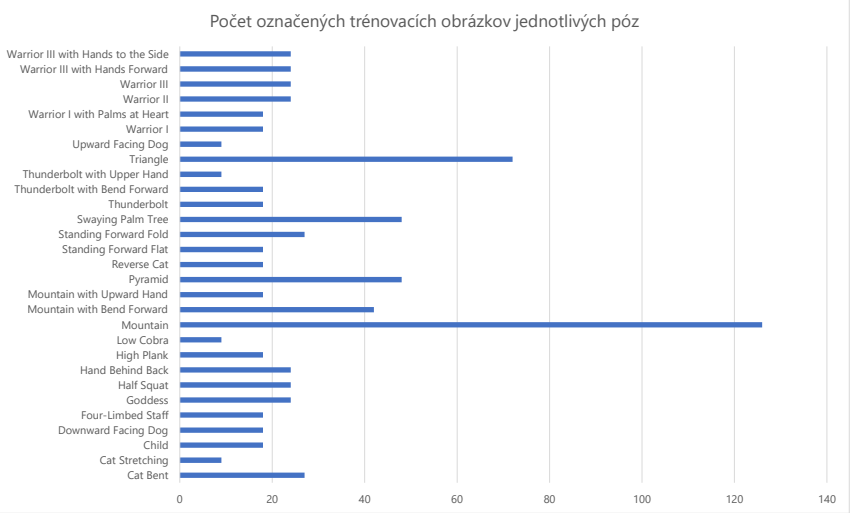
Experimenty sa najmä snažia dokázať úspešnosť modelu s malým množstvom anotovaných dát a preto označená dátová sada pozostáva z malého množstva obrázkov. Experimentuje sa s tromi označenými sadami rôznej veľkosti:

- Sada A pozostávala z jedného trénovacieho obrázku na pózu (*one-shot learning*).
- Sada B pozostávala z jedného trénovacieho obrázku z každého uhla videa, t.j. 3 až 4 obrázky na pózu.
- Sada C pozostávala zo všetkých extrahovaných trénovacích obrázkov, t.j. 9 až 126 obrázkov na pózu, podľa toho, ako často sa póza vyskytovala; spolu 792 obrázkov na všetky pózy.

Vzhľadom na to, že niektoré pózy sa vo videách vyskytovali výrazne častejšie ako iné, napríklad póza *Mountain*, ktorá je súčasťou troch zo štyroch sekvencií, nemá každá póza rovnaký počet označených trénovacích obrázkov (obr. 7.1). Či tento fakt výrazne škodí úspešnosti sa dá vyčítať z matice zámen (*confusion matrix*, obr. 7.3).

Testovaciu dátovú sadu tvorí 6120 obrázkov zo 105 videí 34 cvičení.

Experimentovanie prebiehalo nasledovne – model bol natrénovaný neoznačenou trénovacou sadou. Tento model sa uložil a následne sa vytrénoval označenou sadou A, evaluovala sa jeho úspešnosť na testovacej dátovej sade a model sa obnovil na stav pred trénovaním označenými dátami, aby sa mohlo trénovať a evaluovať so sadou B, nezávisle na sade A.




Obr. 7.1: Počet trénovacích obrázkov na každú pózu z označenej trénovacej sady C.

| | | počet cvičení | počet videí | počet obrázkov |
|---|---|---|---|---|
| **neoznačené trénovacie** | | 16 | 58 | $3 \times 2771$ |
| **označené trénovacie** | **sada A** | 4 | 4 | 29 |
| | **sada B** | 4 | 14 | 109 |
| | **sada C** | 4 | 14 | 792 |
| **testovacie** | | 34 | 105 | 6120 |
| **spolu** | | 54 | 177 | 15225 |

Tabuľka 7.1: Počet cvičení, videí a obrázkov, ktoré tvoria jednotlivé podmnožiny dátovej sady. Čísla v riadku „spolu“ nie sú súčtom čísel z vyšších riadkov, pretože sada A $\subset$ sada B $\subset$ sada C.

Nápodobne sa pokračovalo pre poslednú sadu C. Keďže trénovacích dát je tak málo, tak sa na každú snímku aplikovalo 20 rôznych náhodných augmentácií, čo výrazne sadu zväčšilo.

## 7.2 Popis experimentov

Experimenty boli pre každú konfiguráciu spustené päťkrát a výsledná úspešnosť bola spriemerovaná, aby sa vyhladili malé výkyvy v rozdieloch úspešností medzi rôznymi behmi, pričom každý z piatich experimentov pozostával z evaluácie úspešnosti s trénovacou sadou A, B a C. V prvom experimente je overované či je lepšie použiť euklidovskú alebo kosínusovú vzdialenosť pri vypočítavaní trojicovej stratovej funkcie, pričom sú povolené všetky augmentácie. V druhej sade experimentov je sledovaný vplyv rôznych augmentácií na úspešnosť modelu na overenie, že žiadna z augmentácií neškodí úspešnosti modelu.

## 7.3 Výsledky

Úspešnosť siete je závislá od schopnosti extraktora kľúčových bodov dostatočne presne určovať kľúčové body. Preto je nevyhnutné pri vyhodnocovaní riešenia sledovať s akými

| | sada A | sada B | sada C |
|---|---|---|---|
| **všetky augmentácie** | 63,24 % | 75,57 % | 82,26 % |
| **bez horizontálneho preklopenia** | 9,77 % | 28,47 % | 52,24 % |
| **bez rotácie** | 51,24 % | 74,30 % | 84,16 % |
| **bez sémantického orezu** | 49,25 % | 71,36 % | 78,54 % |

Tabuľka 7.2: Úspešnosť pre rôzne konfigurácie augmentácií pri použití euklidovskej vzdialenosti v trojicovej strate.

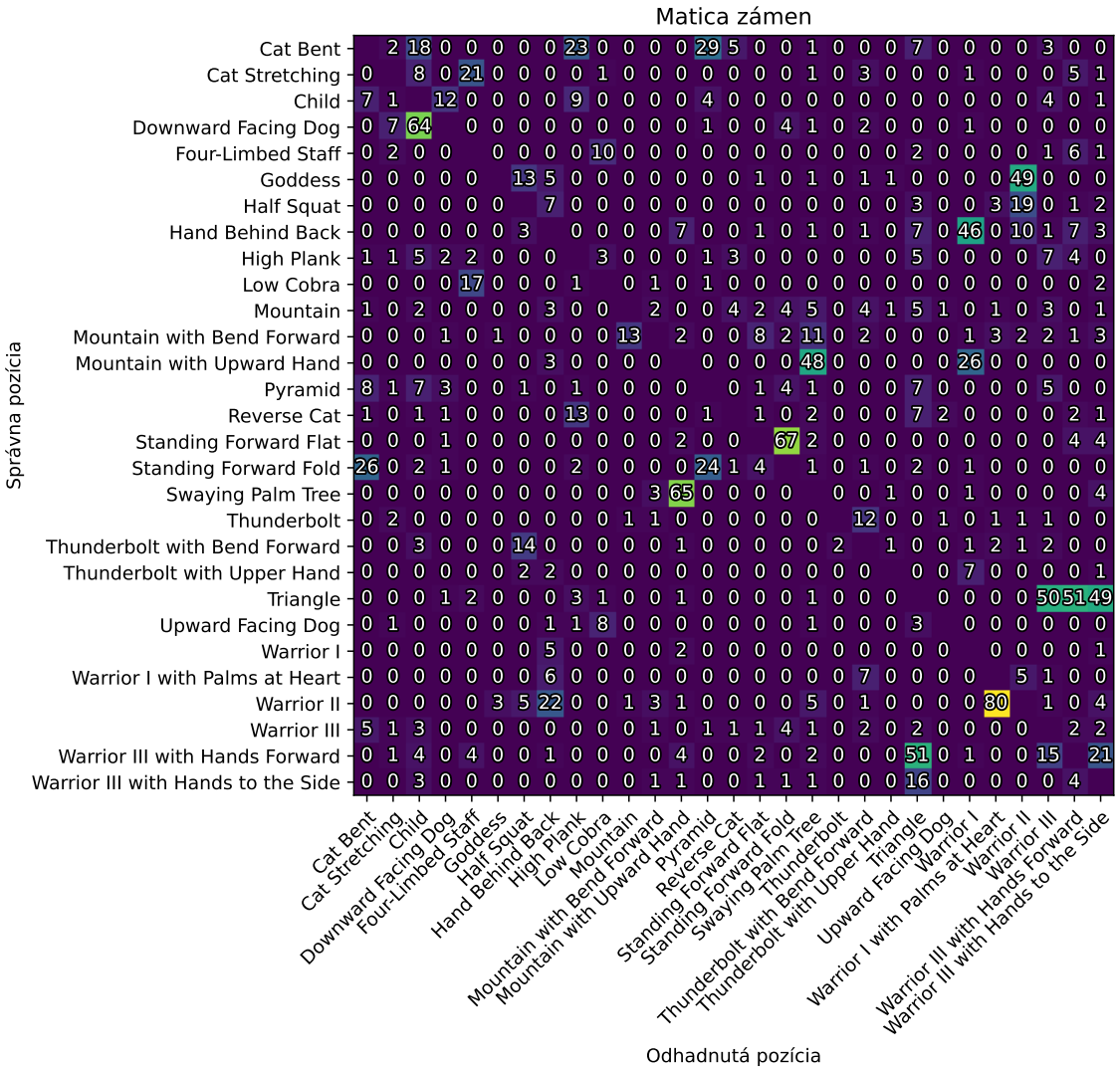
vstupmi mal problém. Po skúmaní nesprávne ohodnotených obrázkov sa prišlo k záveru, že väčšina dôvodov prečo neuspel sa radí do štyroch kategórií, viď obr. 7.2.

Rozdiel úspešnosti pre euklidovskú a kosínusovú vzdialenosť bol nasledovný – pri trénovacej sade C je rozdiel úspešností zanedbateľný – 82,26 % pre euklidovskú a 82,87 % pre kosínusovú. V prípade sady B už začína byť vyše jednopercentný rozdiel medzi metrikami – 75,57 % pre euklidovskú a 74,39 % pre kosínusovú. Pri sade A už je rozdiel veľmi výrazný – 63,24 % pre euklidovskú a 57,44 % pre kosínusovú. Z toho sa dá vyvodiť, že pre menšie množstvo označených trénovacích dát je lepšia euklidovská vzdialenosť, a preto sa v nasledujúcich experimentoch používa iba euklidovská vzdialenosť. Zaujímavosťou je, že na dotrénovanie po použití kosínusovej vzdialenosti bol potrebný približne dvojnásobný počet epoch ako po použití euklidovskej vzdialenosti.

Pri testovaní augmentácií sa evaluovala úspešnosť tak, že sa jedna z augmentácií vypla a výsledok sa porovnával výsledkom dosiahnutým so všetkými augmentáciami. Z výsledkov v tabuľke 7.2 vyplýva, že augmentácie zlepšujú výsledok najviac pri sade A, ktorá je najmenšia, čo sa dalo očakávať – vo väčšej dátovej sade je väčšia šanca, že sa vyskytne póza horizontálne preklopená a s rôznymi rotáciami aj bez augmentácií. Z konkrétnych augmentácií umožnila najväčšie zlepšenie augmentácia náhodného horizontálneho preklopenia, ktorá aj v sade C zlepšila úspešnosť o 33 %. Ďalšia augmentácia, ktorá priniesla výrazné zlepšenie vo všetkých sadách je sémantický orez. Čo sa týka rotácie, tá mala pozitívny efekt v menších sadách A a B a spôsobila mierne zhoršenie v sade C.

Model berie do úvahy iba obrázky samostatne, bez ohľadu na uchovávanie stavu medzi snímkami. Pri zaznamenávaní videa by mohlo byť vylepšením brať ohľad aj na predošlé snímky, čím by sa mohla využiť temporálna informácia o pohybe cvičiaceho a na jej základe by sa zlepšila presnosť.


Obr. 7.2: Typické dôvody nepresného odhadu kľúčových bodov sieťou *MoveNet*. Sieť *MoveNet* bola trénovaná iba na obrázkoch, kde je celá osoba v snímke a takto aj očakáva, že budú vstupné dáta vyzerať. Avšak v dátovej sade sa pozície presahujúce okraj snímky vyskytujú. Takže riešením by bolo, aby cvičiaci stál ďalej od kamery, čo by síce znížilo vo všeobecnosti vyplnenie 2D priestoru snímky cvičiacim, ale jednou z augmentácií je sémantické orezanie, ktoré to napraví. Sémantické orezanie potrebuje väčšie rozlíšenie vstupných dát ako vstupné rozlíšenie siete *MoveNet*, ktoré je $192 \times 192$. Ďalšie problémy spôsobujúce pomýlenie boli nízky kontrast cvičiaceho s pozadím a obrazové artefakty spôsobené svetlom.




Obr. 7.3: Matica zámen (*confusion matrix*) zobrazujúca v každej bunke počet nesprávne klasifikovaných obrázkov – stĺpec udáva predpovedanú pózu a riadok udáva správnu pózu. Obsiahnuté dáta boli získané z experimentu s použitím všetkých augmentácií s testovacou sadou B. Tento typ diagramu je vhodným nástrojom na ladenie siete a zisťovanie jej nedostatkov. Vďaka matici je vidno, že jedna póza, ktorú model často nesprávne klasifikoval je póza *Triangle*. Model si ju mýlil s variantami pozície *Warrior III*.

# Kapitola 8

# Záver

Práca demonštrovala spôsob využitia extraktora kľúčových bodov osoby z obrázka a siamskej neurónovej siete učenej kontrastívnym učením na klasifikáciu jogových pozícií, s cieľom minimalizovať množstvo potrebných trénovacích dát.

Trénovacie a testovacie obrázky boli extrahované z dátovej sady videí cvičení jogy, pričom každé cvičenie bolo zaznamenávané z viacerých uhlov. Videá niektorých cvičení boli označené anotačným nástrojom a následne extrahované snímky boli použité v časti trénovacieho procesu, kde sa sieť trénovala učením s učiteľom. Väčšina cvičení ostala neoznačených a táto časť sady bola používaná v self-supervised časti trénovacieho procesu, kde sa kontrolný signál pri učení generoval zo štruktúry dát, konkrétne z informácií o časovom rozostupe snímok a z informácií, či je uhol pohľadu na cvičenie rovnaký alebo nie.

Obrázky boli pomocou siete *MoveNet* pred ďalším spracovaním konvertované na vektor súradníc reprezentujúcich polohu kľúčových bodov tela cvičiaceho. Toto zmenšenie priestoru vstupných parametrov zjednodušilo úlohu kódovania pozícií do latentného priestoru tým, že odfiltrovalo veľkú časť nesémantickej informácie, ktorá sa v obrázkoch nachádzala.

Trénovací proces bol rozdelený do dvoch etáp. V prvej etape sa trénoval extraktor príznakov, ktorý sa učil reprezentovať vstupné kľúčové body v latentnom priestore pomocou kontrastívneho učenia využívajúceho trojicovú stratovú funkciu (*triplet loss*) a neoznačené trénovacie dáta. Za týmto účelom sa tri inštancie extraktora príznakov so zdieľanými parametrami spojené do siamskej siete. V nasledujúcej fáze sa parametre natrénovaného extraktora príznakov zmrazia a pripojí sa naň jednoduchá klasifikačná sieť, ktorá sa s použitím malého množstva označených dát naučí zobrazenie z latentného priestoru na štítok – jogovú pozíciu.

Úspešnosť finálnej siete bola závislá aj od použitých augmentácií na obrázkové dáta. Konkrétne použité náhodné augmentácie boli horizontálne preklopenie, rotácia a mierne posunutie kľúčových bodov. Na dátovej sade so štyrmi obrázkami na každú jogovú pózu bola s využitím augmentácií dosiahnutá úspešnosť 76 %. Na väčšej dátovej sade s 800 obrázkami na všetky pozície je úspešnosť 82 %. Finálna architektúra je výpočtovo nenáročná, skladá sa zo siete na detekciu kľúčových bodov *MoveNet*, ktorá bola vytvorená pre použitie na mobilných zariadeniach, extraktora príznakov a klasifikátora, pričom posledné dve spomenuté pozostávajú iba z malého množstva plne prepojených vrstiev. V tejto práci bol model natrénovaný na detekciu jogových pozícií ale po získaní vhodnej dátovej sady by sa model dal naučiť na iný šport s využitím neoznačenej dátovej sady a následne doučiť učením s učiteľom na konkrétne pozície na klasifikáciu.

# Literatúra


[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Dostupné z: http://tensorflow.org/.

[2] Aggarwal, C. C., Hinneburg, A. a Keim, D. A. On the Surprising Behavior of Distance Metrics in High Dimensional Space. In: Bussche, J. Van den a Vianu, V., ed. *Database Theory — ICDT 2001*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, s. 420–434. ISBN 978-3-540-44503-6.

[3] Baruah, I. D. How to train Neural Networks like a pro. *Towards Data Science*. Júl 2021. Dostupné z: https://towardsdatascience.com/how-to-train-neural-networks-like-a-pro-1d2362768c1.

[4] Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., LeCun, Y. et al. Signature Verification Using A "Siamese"Time Delay Neural Network. *IJPRAI*. 1993, zv. 7, č. 4, s. 669–688. Dostupné z: https://dblp.org/rec/journals/ijprai/BromleyBBGLMSS93.

[5] Brownlee, J. How to Control the Stability of Training Neural Networks With the Batch Size. *Machine Learning Mastery*. Január 2019. Dostupné z: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/.

[6] Chen, T., Kornblith, S., Norouzi, M. a Hinton, G. E. A Simple Framework for Contrastive Learning of Visual Representations. *CoRR*. 2020, abs/2002.05709. Dostupné z: https://arxiv.org/abs/2002.05709.

[7] Chen, X. a He, K. Exploring Simple Siamese Representation Learning. *CoRR*. 2020, abs/2011.10566. Dostupné z: https://arxiv.org/abs/2011.10566.

[8] Chollet, F. Introduction to Keras for Engineers. Apríl 2020. Dostupné z: https://keras.io/getting_started/intro_to_keras_for_engineers/.

[9] Chollet, F. Introduction to Keras for Researchers. Apríl 2020. Dostupné z: https://keras.io/getting_started/intro_to_keras_for_researchers/.

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

[11] Dilmegani, C. What is Few-Shot Learning? Methods & Applications in 2022. *AI Multiple*. Marec 2022. Dostupné z: https://research.aimultiple.com/few-shot-learning.

[12] Essam, H. a Valdarrama, S. L. Image similarity estimation using a Siamese Network with a triplet loss. Marec 2021. Dostupné z: https://keras.io/examples/vision/siamese_network/.

[13] Google. *Movenet/singlepose/lightning*. Dostupné z: https://tfhub.dev/google/movenet/singlepose/lightning/4.

[14] Grill, J., Strub, F., Altché, F., Tallec, C., Richemond, P. H. et al. Bootstrap Your Own Latent: A New Approach to Self-Supervised Learning. *CoRR*. 2020, abs/2006.07733. Dostupné z: https://arxiv.org/abs/2006.07733.

[15] Jaiswal, A., Babu, A. R., Zadeh, M. Z., Banerjee, D. a Makedon, F. A Survey on Contrastive Self-Supervised Learning. *Technologies*. 2021, zv. 9, č. 1. DOI: 10.3390/technologies9010002. ISSN 2227-7080. Dostupné z: https://www.mdpi.com/2227-7080/9/1/2.

[16] Jing, L. a Tian, Y. Self-supervised Visual Feature Learning with Deep Neural Networks: A Survey. *CoRR*. 2019, abs/1902.06162. Dostupné z: http://arxiv.org/abs/1902.06162.

[17] 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/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[18] Kutálek, J. *Detekce jógových pozic v obraze*. Brno, CZ, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Dostupné z: https://www.fit.vut.cz/study/thesis/24128/.

[19] LeCun, Y. a Misra, I. Self-supervised learning: The dark matter of intelligence. *Meta AI*. Marec 2021. Dostupné z: https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/.

[20] Mayorquin, S. a Rodriguez, T. YogAI: Smart Personal Trainer. *Hackster.io*. Február 2019. Dostupné z: https://www.hackster.io/yogai/yogai-smart-personal-trainer-f53744.

[21] Nichols, H. *How does yoga work?* Dostupné z: https://www.medicalnewstoday.com/articles/286745.

[22] Schroff, F., Kalenichenko, D. a Philbin, J. FaceNet: A Unified Embedding for Face Recognition and Clustering. *CoRR*. 2015, abs/1503.03832. Dostupné z: http://arxiv.org/abs/1503.03832.

[23] Srinivasan, A. V. Stochastic Gradient Descent — Clearly Explained. *Towards Data Science*. September 2019. Dostupné z: https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.

[24] Votel, R. a Li, N. Next-Generation Pose Detection with MoveNet and TensorFlow.js. Máj 2021. Dostupné z: https://blog.tensorflow.org/2021/05/next-generation-pose-detection-with-movenet-and-tensorflowjs.html.

[25] Wang, Y. a Yao, Q. Few-shot Learning: A Survey. *CoRR*. 2019, abs/1904.05046. Dostupné z: http://arxiv.org/abs/1904.05046.

[26] Zhou, X., Wang, D. a Krähenbühl, P. Objects as Points. *CoRR*. 2019, abs/1904.07850. Dostupné z: http://arxiv.org/abs/1904.07850.

# Príloha A

# Obsah priloženého pamäťového média

V súbore `readme.md` sa nachádza podrobná dokumentácia k celému obsahu pamäťového média a ku všetkým skriptom, vrátane opisu vstupných parametrov.

Priečinok `scripts/` obsahuje skripty v jazyku Python. Zoznam potrebných závislostí sa nachádza v súbore `requirements.txt` v koreňovom adresári. Jednotlivé skripty sú:

- `extractTraining.py` – pre každé cvičenie vypočíta optický tok v čase, vypočíta korelačné koeficienty a zistí odstup jednotlivých videí, následne synchronizované videá uloží do nového priečinka a extrahuje z nich neoznačené trénovacie dáta v podobe kontrastívnych trojíc obrázkov,
- `extractLabelled.py` – pre každé video s JSON súborom vygenerovaným anotačnou aplikáciou extrahuje z videa obrázky jogových pozícií a obrázky potriedi podľa označenej pozície do priečinkov,
- `model.py` – slúži na vytvorenie, trénovanie a evaluáciu modelu,
- `skeleton.py` – každý vstupný obrázok spracuje sieťou *MoveNet* a vykreslí do obrázka jednotlivé kľúčové body, kosti medzi kľúčovými bodmi a obdĺžnikovú obálku,
- `augmentation.py` – pomocná trieda, ktorá obsahuje metódy na augmentácie obrázkov – horizontálne preklopenie, otočenie, náhodné orezanie, sémantické orezanie a modifikácia farieb.

Priečinok `labels/` obsahuje JSON súbory, v ktorých sú definované jednotlivé jogové pozície vo forme štítkov vo formáte, ktorý podporuje anotačná aplikácia. Použitie tohto súboru a podrobný návod na použitie anotačnej aplikácie sa nachádza v jej dokumentácií[1].

Priečinok `data/` obsahuje podmnožinu neoznačených dát `data/training` trénovacích dát
`data/poses/train` a testovacích dát `data/poses/test`.

[1] https://github.com/leSamo/Video-Annotation#readme

```
CD-ROM
├── data/
│   ├── labelled/
│   ├── poses/
│   │   ├── test/
│   │   │   ├── pose A/
│   │   │   └── pose B/
│   │   └── train/
│   │       ├── pose A/
│   │       └── pose B/
│   ├── training/
│   │   ├── anchor/
│   │   ├── negative/
│   │   └── positive/
│   ├── videosSynced/
│   └── videosUnsynced/
├── labels/
│   ├── Cat sequence.json
│   ├── Moon salutation.json
│   ├── Sun salutation.json
│   └── Warrior III.json
├── latex/
├── scripts/
│   ├── augmentation.py
│   ├── extractLabelled.py
│   ├── extractTraining.py
│   ├── filter.py
│   ├── model.py
│   └── skeleton.py
├── poster.pdf
├── readme.md
├── requirements.txt
├── Video annotation web app link
└── video.mp4
```

# Príloha B

# Manuál na použitie anotačnej aplikácie

Anotačná aplikácia kompatibilná so skriptami na pamäťovom médiu sa nachádza na adrese http://video-anno.herokuapp.com/ so zdrojovým kódom na https://github.com/leSamo/Video-Annotation. Detailnejší manuál sa nachádza v súbore `README.md` v repozitári alebo sa k nemu dá dostať kliknutím na tlačidlo nápovedy (`Help`) vnútri aplikácie.

### Zdieľanie priečinka s videami

Pred použitím aplikácie si musíte na Google Disku vytvoriť priečinok s videami, ktoré chcete anotovať a zdieľať ho s anotačnou aplikáciou.

1. Vytvorte si nový priečinok na svojom Google Disku (https://drive.google.com/).
2. Do priečinka vložte všetky videá z dátovej sady (podporované prípony videí sú `mp4, mov, mkv, wmv, flv, avi, webm, mpg, ogv, 3gp`; za predpokladu, že ich podporuje aj vami použitý prehliadač)
3. Zdieľajte priečinok s e-mailovou adresou `annotation.tester@gmail.com` a uistite sa, že ste vybrali oprávnenia „Editor“ (nevyhnutné na nahranie výsledných anotačných údajov vo formáte JSON späť do tohto priečinka)

### Importovanie štítkov a videí

Použite tlačidlo importovania štítkov (`Import labels`) a vyberte JSON súbor s definíciou štítkov. Vzorové súbory štítkov sa nachádzajú na pamäťovom médiu v priečinku `labels/`.

Použite tlačidlo importovania videí (`Import video`) na otvorenie modálneho okna. Vložte ID priečinka na Google Disku (ID priečinka možno získať skopírovaním časti adresy URL za poslednou lomkou, keď ste v priečinku na Disku Google, je to reťazec dlhý 33 znakov vo formáte base64).

Ak ste zadali platné ID priečinka, mali by ste vidieť v ľavom paneli všetky dostupné videá z priečinka. Vpravo sa môžete rozhodnúť vytvoriť nový súbor JSON s výstupnými údajmi alebo importovať existujúci výstupný súbor JSON, ak je k dispozícii. Môžete vybrať až 6 videí, ktoré chcete anotovať vedľa seba. Jediným obmedzením je, že aplikácia nedokáže rozpoznať počet snímok za sekundu (FPS) aktuálneho videa, takže pri načítaní videí sa dá hodnota ručne nastaviť. Je zrejmé, že anotovanie viacerých videí naraz bez toho, aby mali všetky rovnakú hodnotu FPS, nie je možné.

### Vzájomné synchronizovanie videí

Viaceré videá vedľa seba je možné synchronizovať pomocou jazdcov nad každým videom. Výstupné údaje JSON budú správne odrážať posun každého videa. Avšak ručné nastavenie typicky nie je potrebné, keďže skript `extractTraining.py` dokáže videá synchronizovať automaticky pred nahratím videí na Google Disk.

### Prehrávanie videí

Jednotlivé videá nemajú vlastné ovládacie prvky, pretože sa prehrávajú synchrónne. Aktuálna pozícia vo videách je znázornená bielym pruhom (bežec) na sivom pozadí (časová os) v spodnej časti stránky. Prehrávanie je možné ovládať pomocou tlačidiel nad časovou osou alebo pomocou klávesových skratiek[1]. Rýchlosť prehrávania je možné zmeniť pomocou bežca v ľavej dolnej časti stránky.

### Anotovanie videí

Kliknutím na položku v ľavom paneli štítkov sa vytvorí nová anotácia, ktorá sa zobrazí vpravo na paneli histórie. Anotácie sú na časovej osi znázornené farebnými lichobežníkmi. Jednotlivé body lichobežníka sa nazývajú, v poradí zľava doprava, „pred“ (*before*), „začiatok“ (*start*), „najlepší“ (*best*), „koniec“ (*end*), „po“ (*after*). Skript na extrakciu vyberá snímky medzi bodmi „začiatok“ a „koniec“. Body „pred“, „najlepší“ a „po“ v aktuálnej práci nie sú využité, ale môžu byť užitočné v budúcich projektoch.

Po pridaní anotácie ju nie je vidno na časovej osi, kým nebude mať aspoň 2 kľúčové body. Kľúčové body anotácií sa pridávajú na aktuálnu pozíciu vo videu pomocou tlačidiel v ľavom dolnom rohu alebo pomocou klávesových skratiek.

Predchádzajúca anotácia môže byť upravená po jej označení v paneli histórie. Vybranú anotáciu je možné odstrániť kliknutím na ikonu odpadkového koša vedľa nej v paneli histórie alebo pomocou klávesy `del`.

Stlačením tlačidla uloženia (`Save`) sa výstupný súbor JSON uloží.

### Výstupný JSON súbor

Výstupný súbor vo formáte JSON sa nahrá do rovnakého priečinka na Disku Google, kde sú zdrojové videá. Ak anotujete viacero videí, vytvorí sa správne synchronizovaný súbor JSON pre každé video, pomenovaný rovnako ako video, ku ktorému patrí. Výstupný súbor neskôr môžete upraviť tak, že ho vyberiete pri importovaní videa. Následne by sa zobrazili všetky existujúce anotácie a umožnilo by to mazanie a pridávanie nových anotácií.

[1] Zoznam klávesových skratiek je uvedený v rozšírenej dokumentácií v repozitári.