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

MOBILNÁ APLIKÁCIA PRE PODPORU TRÉNOVANIA SILOVÝCH ŠPORTOV

MOBILE APPLICATION FOR ASSISTANCE IN STRENGTH TRAINING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. SIMON KOŠINA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. ROMAN JURÁNEK, Ph.D.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

156727

Ústav:	Ústav počítačové grafiky a multimédií (UPGM)
Student:	**Košina Simon, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Strojové učení
Název:	**Mobilní aplikace pro podporu trénování silových sportů**
Kategorie:	Mobilní aplikace
Akademický rok:	2023/24

Zadání:

1. Seznamte se se současným stavem aplikací pro podporu trénování silových sportů.
2. Navrhněte mobilní aplikaci pro velocity-based trénink.
3. Implementujte aplikaci, která bude sledovat uživatele při tréninku, monitorovat jeho pohyb a poskytovat zpětnou vazbu.
4. Otestujte aplikaci na několika uživatelích a porovnejte ji s existujícími řešeními.
5. Diskutujte silné a slabé stránky vaší aplikace.
6. Vytvořte materiály prezentující vaši práci (video, web, atd.)

Literatura:

- Tober, Jacob (2022-05-21). "Reliability and validity of MetricVBT beta".
- Cetin, Onat; Isik, Ozkan (2021-11-11). "Validity and Reliability of MyLift App in Determining 1-RM for Deadlift and Back Squat Exercises". European Journal of Human Movement.

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

Vedoucí práce:	**Juránek Roman, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, prof. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	9.11.2023

Abstrakt

Cieľom práce je vytvoriť mobilnú aplikáciu pre zariadenia s operačným systémom Android, poskytujúcu športovcom spätnú väzbu počas silového cvičenia vo forme rýchlostných metrík pre jednotlivé opakovania, vykonané v rámci série určitého cviku. Tréning na základe využitia rýchlostných metrík sa stáva čoraz populárnejším, jednak v praxi, ale aj vo výskume, kde sa ukázalo, že pomocou týchto objektívnych metrík je možné odhadnúť náročnosť tréningu. Výsledná aplikácia využíva metódy strojového učenia na detekciu závaží naložených na olympijskej ose v snímkoch z kamery mobilného zariadenia a sleduje tak trajektóriu ich pohybu. Kalibrácia prejdenej vzdialenosti je vykonaná na základe známej veľkosti váhových kotúčov. Algoritmus funguje v reálnom čase a poskytuje teda užívateľom spätnú väzbu už počas cvičenia, vo forme zvukového signálu, pri dosiahnutí stanoveného prahu vybranej rýchlostnej metriky.

Abstract

The aim of this work is to create a mobile application for Android devices that provides athletes with real-time feedback during strength training in the form of velocity metrics for individual repetitions within a set of a certain exercise. Velocity based training is becoming increasingly popular both in practical applications and in research, where it has been demonstrated that these objective metrics can be used to estimate the intensity of a given set. The resulting application utilizes machine learning methods to detect weights plates loaded on a barbell in frames coming from the mobile device's camera and tracking their movement trajectory. Known size of the weight plates is used to calibrate the travelled distance. The algorithm operates in real-time, providing users with feedback during exercise sessions in the form of an auditory signal when a predefined threshold of selected velocity metric is reached.

Kľúčové slová

tréning na základe rýchlosti pohybu, silový tréning, detekcia objektov, mobilná aplikácia, Android, Jetpack Compose, TensorFlow Lite

Keywords

velocity based training, strength training, object detection, mobile application, Android, Jetpack Compose, TensorFlow Lite

Citácia

KOŠINA, Simon. *Mobilná aplikácia pre podporu trénovania silových športov.* Brno, 2023. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Roman Juránek, Ph.D.

Mobilná aplikácia pre podporu trénovania silových športov

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením pana Ing. Romana Juránka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Simon Košina
15. mája 2024

Poďakovanie

Veľmi ďakujem vedúcemu mojej práce za vypísanie a vedenie vlastného zadania, cenné rady a pomoc na konzultáciách. Taktiež by som chcel poďakovať pánovi Mgr. Janovi Štastnému, Ph.D., z Centra športových aktivít, za ochotu a výpomoc pri zbieraní referenčných dát systémom pre sledovanie pohybu Qualysis. Vďaka patrí aj mojim rodičom a priateľke, za ich trpezlivosť a podporu počas celého štúdia.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Následujúca práca sa zaoberá vývojom aplikácie schopnej poskytnúť užívateľom spätnú väzbu počas silového tréningu. V rámci silového tréningu sa v poslednej dobe pomaly začína využívať a skúmať tréning na základe rýchlosti pohybu, ktorý pomocou objektívnych rýchlostných metrík posudzuje náročnosť vykonávaných sérií, oproti čisto subjektívnej spätnej väzbe športovcov. Využitie rôznych rýchlostných metrík vidíme zatiaľ v prevažne u komplexnejších cvikoch s olympijskou osou, akými sú napr. drepy, bench press alebo mŕtvy ťah, zameraných na rozvoj sily.

Vzhľadom na to, že väčšina dostupných komerčných riešení je pomerne drahá a kvalita existujúcich aplikácií pre zariadenia s operačným systémom Android nie je veľmi vysoká, alebo tieto aplikácie neponúkajú určité vymoženosti, je tu priestor pre vytvorenie aplikácie bez takýchto obmedzení.

V rámci práce je teda vytvorená aplikácia pre systém Android, ktorá pomocou kamery mobilného zariadenia detekuje závažia naložené na činke a vďaka tomu následne sleduje trajektóriu činky počas vykonávania série opakovaní. Takto získané dáta sú analyzované v reálnom čase, pričom algoritmus detekuje opakovania, vykonané v rámci série a s využitím kalibrácie pomocou známej veľkosti naložených váhových kotúčov pre dané opakovania počíta rýchlostné metriky. Aplikácia umožňuje užívateľom vybrať si pred cvičením konkrétnu rýchlostnú metriku a hodnotu jej prahu, o ktorého prekročení sú počas cvičenia informovaní pomocou zvukového signálu.

Kapitola 2 informuje čitateľa o tom, ako vyzerá typický tréningový plán zameraný na rozvoj sily a akým spôsobom je možné v rámci neho využiť rôzne rýchlostné metriky, ponúkajúce objektívnu spätnú väzbu o výkone. Prieskum existujúcich riešení a špecifikácia požiadavkov je uvedená v kapitole 3. Kapitola 4 sa venuje teoretickým základom vývoja aplikácií pre platformu Android, s využitím frameworku Jetpack Compose a kapitola 5 je venovaná princípom detekcie objektov v obraze a sledovaniu ich trajektórie. Návrh užívateľského rozhrania aplikácie a štruktúry databáze, je uvedený v kapitole 6. Kapitola 7 opisuje trénovanie modelov strojového učenia pre detekciu závaží v snímkoch, ich vyhodnotenie a taktiež samotnú implementáciu aplikácie, spolu s algoritmom pre analýzu sérií opakovaní a výpočet rýchlostných metrík. Implementovaný algoritmus bol následne porovnaný so softvérom Kinovea a taktiež s profesionálnym systémom Qualysis, určeným pre sledovanie pohybu. Výsledky porovnania a zhodnotenie je uvedené v kapitole 8.

Kapitola 2

Silový tréning na základe rýchlosti pohybu

Silový tréning alebo posilňovanie, zvyčajne znamená vykonávanie určitých fyzických cvikov, typicky za účelom zvýšenia svalovej hmoty a sily [45]. Silový tréning má však ďalšie výhody, ako je napr. zníženie rizika pádov u starších ľudí [41], prevencie a zníženie rizika úrazov [27], možné zlepšenie zdravia kostí a zníženie rizika osteoporózy [3], či zlepšenie flexibility a mobility [28]. V poslednej dobe sa ukázalo, že silový tréning môže mať taktiež pozitívny vplyv na kognitívne schopnosti jedincov [52], zlepšenia celkovej nálady [17], či zníženie úzkostí [18]. Podľa zámerov jednotlivca je možné vykonávať cviky s vlastnou váhou, s činkami, za pomoci rôznych posilňovacích strojov alebo iných prostriedkov s dostatočnou rezistenciou.

Kapitola popisuje typickú štruktúru tréningového plánu zameraného predovšetkým na rozvoj sily v rámci špecifických cvikov a možnosti uplatnenia princípov tréningu založeného na rýchlosti pohybu (*velocity based training*, VBT) v rámci tréningového plánu, spolu s vysvetlením jednotlivých rýchlostných metrík.

V rámci nasledujúcej kapitoly, aj pri návrhu a implementácii aplikácie, sa zameriame prevažne na cviky s činkami, ktoré sú súčasťou rôznych silových súťaží a často sa princípy tréningu na základe rýchlosti pohybu uplatňujú práve u nich.

2.1 Typická štruktúra silového tréningového plánu

Športovci, ktorí vykonávajú silový tréning kvôli zlepšeniu atletického výkonu, alebo ľudia, ktorí súťažia v rôznych silových disciplínach, ako napríklad silový trojboj[1], či vzpieranie[2], často v rámci tréningov nasledujú určitý tréningový plán. Takýto tréningový plán môže mať predpísané jednotlivé cviky, váhy, s akými budú vykonávané, počet sérií a opakovania vykonané v rámci jednotlivých sérií.

Zdvíhané váhy a počty opakovaní je možné odvodiť a predpísať napr. ako určité percento maximálnej váhy, ktorú jedinec dokáže zdvihnúť v rámci daného cviku (*one repetition maximum*, 1RM), buď na základe historických údajov alebo ako informovaný odhad. Aj keď sa počet opakovaní, ktorý sú ľudia schopní vykonať pri danom percente 1RM, môže líšiť [39], využitím tejto metódy by sme v rámci tréningového planú zameraného primárne na rozvoj sily, mohli predpísať napr. 3 série po 4 opakovaniach s 85% 1RM. Nevýhodou tejto

[1] Wikipedia – Powerlifting
[2] Wikipedia – Olympic Weightlifting

metódy je fakt, že maximálny výkon môže byť ovplyvnený rôznymi faktormi, ako je napr. stres, či únava, a teda striktné dodržanie takto dopredu stanoveného plánu nemusí mať očakávaný efekt a môže spôsobiť nežiadúce účinky v prípade, že tréning je pre daného športovca v určitý deň menej alebo viac náročný, ako bolo úmyslom.

Pred niekoľkými rokmi sa teda začalo skúmať a využívať v praxi plánovanie tréningov založené na škále vnímanej námahy (*rate of perceived exertion*, RPE), aby došlo k zmierneniu vyššie popísaných negatívnych efektov [53]. S využitím tejto škály (tab. 2.1) je potom možné v rámci tréningu namiesto špecifického percenta z 1RM, predpísať napr. 3 série po 4 opakovania pri hodnote RPE 8.

Tabuľka 2.1: Škála RPE.

RPE	Popis vnímanej námahy
10	maximálne vyčerpanie
9.5	nedokáže vykonať ďalšie opakovanie, dokázal by však pridať na váhe
9	1 opakovanie v rezerve
8.5	1-2 opakovanie v rezerve
8	2 opakovanie v rezerve
7.5	2-3 opakovanie v rezerve
7	3 opakovanie v rezerve
5-6	4-6 opakovaní v rezerve
3-4	nízka námaha
1-2	žiadna až nízka námaha

Síce využitie takéhoto prístupu už netrpí rovnakými nedostatkami, ako je predpisovanie tréningu na základe percenta z 1RM, keďže tréning je lepšie prispôsobený danému jedincovi a berie do úvahy, ako sa športovec cíti v jednotlivé dni a jeho úroveň únavy. Jedná sa však o subjektívne hodnotenie výkonu a športovci nemusia vždy byť schopní presne ohodnotiť ich vnímanú námahu v rámci série, aj keď sa ukazuje, že odhad jedincov sa s postupom času a získanými skúsenosťami zlepšuje [44].

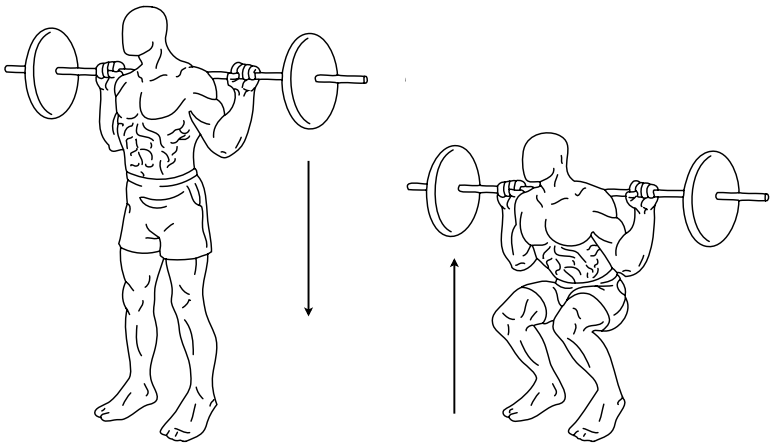
Vyššie uvedené limitácie sa snaží minimalizovať tréning na základe rýchlosti pohybu [35], kde predpíšeme napr. 3 série po 4 opakovaniach pri minimálnej rýchlosti 0.30 ms^{-1}, namiesto špecifického percenta z 1RM, či hodnotenia RPE. V rámci štúdii sa však ukázalo, že je potrebné, aby predpísané rýchlosti boli individualizované pre daného športovca, či dokonca vykonávaný cvik [20]. Takto predpísaný tréning je vhodné kombinovať s predchádzajúcimi metódami a získať tak presnejší prehľad o výkone športovcov, keďže metriky namerané počas tréningu môžme spolu s nameranými testovacími hodnotami využiť na predikciu 1RM [2], či určenie hodnoty RPE danej série [32].

2.2 Rýchlostné metriky

Pri tréningu na základe rýchlosti je možné využiť niekoľko metrík a spôsobov ako definovať rýchlosť jednotlivých opakovaní v rámci série. Mimo iné sa využíva napr. priemerná koncentrická rýchlosť (*average concentric velocity*, AVC), priemerná pohonná rýchlosť (*mean propulsive velocity*, MPV) a maximálna rýchlosť (*peak velocity*, PV) [6]. ACV je priemerná rýchlosť počas celej koncentrickej fázy pohybu, viď obr. 2.1. MPV je definovaná ako prie-

merná rýchlosť od počiatku koncentrickej fázy až do momentu, kedy zrýchlenie je pomalšie ako gravitačné zrýchlenie a PV odpovedá maximálnej rýchlosti dosiahnutej počas koncentrickej fázy pohybu.

V prípade predikcie 1RM, pomocou lineárnej regresie, na základe závislosti rýchlosti vykonaného opakovania na váhe, sa však ukázalo, že najspoľahlivejšou metrikou je ACV, ktorá dosiahla najvyššej hodnoty $R^2 = 0,989$ oproti $R^2 = 0,983$ v prípade metriky MPV a $R^2 = 0,974$ v prípade metriky PV [6]. Vidíme však, že všetky hodnotené metriky dosiahli pomerne vysokého koeficientu determinácie a môžu nám ponúknuť rôzne informácie.



Obr. 2.1: Vľavo je ukázaný príklad excentrickej kontrakcie, kde sa svaly zapojené v rámci cviku postupne predlžujú, ako jedinec klesá do drepu. Vpravo je potom príklad koncentrickej kontrakcie v rámci cviku, pri ktorej sa zúčastnené svaly skracujú. Typicky pri cvikoch s činkami nastáva excentrická fáza pohybu, keď činka klesá dole a koncentrická fáza nastáva, ak jedinec dvíha činku smerom nahor.

2.3 Predikcia 1RM, skóre RPE a plánovanie tréningu s ich využitím

Na predikciu 1RM sa využíva individualizovaná tabuľka rýchlostí prvých opakovaní pre určitý cvik. V rámci zberu dát jedinec vykoná niekoľko sérií s postupne sa zvyšujúcimi váhami, až sa dopracuje k jeho 1RM, pričom z každej série si poznamenáme rýchlosť prvého opakovania (napr. ako metriku ACV). Pomocou lineárnej regresie potom získame závislosť medzi percentom z 1RM a rýchlosťou prvého opakovania v sérii pre daný cvik, ktorú môže športovec využiť v rámci budúcich tréningov.

Tabuľka 2.2: Tabuľka rýchlosti prvých opakovaní, príklad otestovania 1RM jedinca spolu so zahrievacími sériami. Tabuľka prevzatá z [21].

Séria	Váha [kg]	ACV [ms^{-1}]	% 1RM
1	20	1.20	6.7
2	105	0.95	35
3	165	0.70	55
4	210	0.52	70
5	232,5	0.40	77.5
6	255	0.30	85
7	270	0.24	90
8	285	0.20	95
9	300	0.16	100
10	302.5	zlyhanie	zlyhanie

V rámci tréningového plánu môže mať jedinec predpísané napr. 3 série po 4 opakovaniach s 85% 1RM. Na základe prechádzajúcich tréningov a rýchlostí nameraných počas rozvičovacích sérií, dokáže jedinec určiť jeho 1RM pre prvú sériu cviku. Po jej vykonaní sa môže pozrieť na namerané metriky a váhu poprípade upraviť pre následujúce série.

Tabuľka 2.3: Ukážka tréningu, kde cieľom bolo vykonať 3 série po 4 opakovaniach s 85% 1RM. Na základe rýchlosti z prvého opakovania v rámci prvej série a vyššie uvedenej tabuľky (tab. 2.2) vidíme, že je potreba znížiť danú váhu na 85% z predikovaného 1RM, aby bola dodržaná naplánovaná intenzita. Príklad prevzatý z [21].

Séria	Váha [kg]	ACV prvého opakovania [ms^{-1}]	% 1RM	Predikované 1RM [kg]
1	255	0,29	∼87,55	291,27
2	247,58	0,3	∼85,00	291,27
3	247,58	0,31	∼84,16	294,08

Pre predikciu skóre RPE je potrebné namerať individualizovanú tabuľku rýchlostí posledných opakovaní pre určitý cvik. V takomto prípade si jedinec vyberie váhu, s ktorou v rámci daného cviku dokáže vykonať približne 10-12 opakovaní a vykoná danú sériu do zlyhania, pričom sú zaznamenané rýchlosti všetkých opakovaní. Vďaka nameraným hodnotám je potom možné určiť závislosť medzi rýchlosťou posledného opakovania v rámci série a skóre RPE danej série.

Tabuľka 2.4: Tabuľka rýchlosti posledných opakovaní na základe testovacej série. Tabuľka prevzatá z [21].

Opakovanie	RPE	ACV [ms^{-1}]
11	10	0.16
10	9	0.19
9	8	0.22
8	7	0.25
7	6	0.30
6	5	0.35
5	4	0.40
4	3	0.45
3	2	0.52
2	1	0.60
1	–	–

Podobne ako v predchádzajúcom prípade, môže mať športovec naplánované napr. 3 série po 4 opakovaniach pri hodnote RPE 8. Na základe historických údajov, či percenta z 1RM, dokáže jedinec určiť váhu pre prvú pracovnú sériu. Po jej vykonaní získame nameranú rýchlosť posledného opakovania a dokážeme z vyššie spomenutej tabuľky odvodiť skutočnú hodnotu RPE a poprípade upraviť váhu na následujúce série tak, aby bola dodržaná požadovaná intenzita tréningu.

Tabuľka 2.5: Ukážka tréningu, kde cieľom bolo vykonať 3 série po 4 opakovaniach pri hodnote RPE 8. Na základe rýchlosti z prvého opakovania v rámci prvej série a vyššie uvedenej tabuľky (tab. 2.4) vidíme, že je potreba znížiť danú váhu, aby bola dodržaná naplánovaná intenzita. Príklad prevzatý z [21]

Séria	Váha [kg]	ACV prvého opakovania [ms^{-1}]	RPE
1	255	0,205	8.5
2	250	0,22	8
3	250	0,26	7

Iné možnosti plánovania tréningu môžu vyžadovať, aby športovec ukončil sériu v prípade, že dosiahol určité skóre RPE, či rýchlosť opakovaní klesla pod určitú hodnotu a preto je výhodné, ak zariadenia určené na meranie rýchlosti dokážu poskytnúť športovcovi spätnú väzbu, napr. v podobe zvukového signálu.

Kapitola 3

Prieskum existujúcich riešení a špecifikácia požiadavkov

Aj napriek tomu, že analýza výkonu pomocou rýchlostných metrík je čoraz populárnejšia, mnoho rekreačných športovcov ju stále nevyužíva v rámci tréningu. Väčšina komerčne dostupných zariadení je pomerne drahá, či vyžaduje čas na nastavenie a spárovanie s mobilným telefónom. Špecializované zariadenia je potrebné neustále prenášať so sebou a vhodne ich zakaždým nastaviť. Ceny týchto zariadení nie sú veľmi prívetivé, keďže sa typicky pohybujú v stovkách dolárov, počnúc okolo hranice 400$. Vytvorenie užívateľsky prívetivej aplikácie, ktorá jednoducho na základe kamery sníma pohyb a analyzuje rýchlostné metriky v rámci tréningu, by teda mohlo byť prínosné.

Následujúca kapitola sa venuje prieskumu komerčne dostupných zariadení, akými sú lineárne prevodníky polohy, či bezdrôtové senzory a k nim dostupným aplikáciam. Taktiež bol vykonaný prieskum už existujúcich aplikácií, ktoré sú založené na princípoch počítačového videnia a následne špecifikujeme požiadavky na navrhovanú aplikáciu.

3.1 Lineárne prevodníky polohy

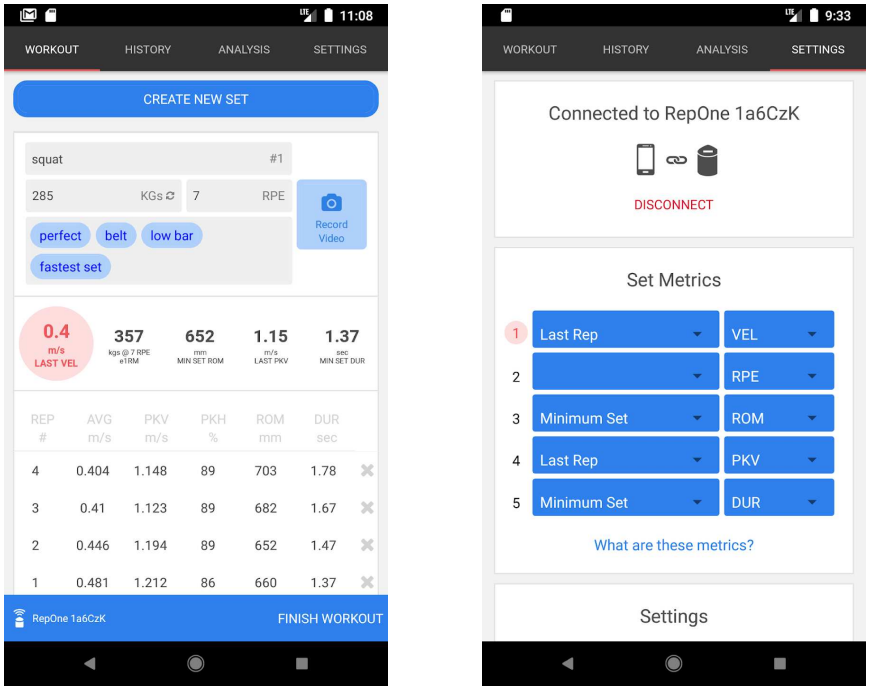
V súčasnej dobe je u väčšiny cvikov s činkou možné spoľahlivo merať rýchlosť pomocou komerčne dostupných lineárnych prevodníkov polohy (*linear positional transducer*). Príkladmi takýchto produktov sú GymAware RS[1], Vitruve Encoder[2], RepOne Tether[3] a iné.

K zariadeniam sú typicky ponúkané aplikácie, ktoré po spárovaní s prevodníkom umožňujú užívateľom zaznamenávať jednotlivé metriky. Medzi takto ponúkané aplikácie patrí napr. Vitruve [4], RepOne Personal[5], či Enode Pro[6]. Zvyčajne aplikácie ponúkajú možnosť nahratia metrík pre danú sériu opakovaní, pričom pred spustením samotného nahrávania môže užívateľ zadať názov cviku, váhu, s ktorou cvičí a ďalšie informácie, na základe ktorých môže neskôr filtrovať medzi historickými dátami a analyzovať tak svoj výkon. Je vhodné, ak aplikácie taktiež ponúkajú možnosť nahrať spolu s metrikami videozáznam, na základe, ktorého môže daný športovec, či tréner analyzovať techniku prevedenia cviku. Užívateľ následne spustí zaznamenávanie, odcvičí sériu a nahrávanie ukončí, pričom sa mu zobrazia namerané

[1] GymAware RS
[2] Vitruve Encoder
[3] RepOne Tether
[4] Google Play – Vitruve
[5] Google Play – RepOne Personal
[6] Google Play – Enode Pro

metriky pre jednotlivé opakovania, ktoré vykonal v rámci série, či rôzne agregované metriky pre celú sériu. Lepšie aplikácie potom ponúkajú rôzne možnosti analýzy historických dát, predikcie 1RM, či nastavenie zobrazovaných metrík.

Obr. 3.1: Ukážka lineárneho prevodníku polohy RepOne Tether. Obrázky prevzaté z [38].



Obr. 3.2: Ukážka aplikácie RepOne Personal. Vľavo je ukázaný detail zaznamenanej série spolu s vybranými metrikami pre každé opakovanie. Vpravo vidíme obrazovku nastavení, kde si užívateľ môže predvoliť vhodné metriky.

Vybrané aplikácie však nemajú príliš dobré hodnotenie v rámci služby Google Play. K januáru 2024 má aplikácia RepOne Personal hodnotenie 3.6 z 5, aplikácie Vitruve má 3.9 z 5 a Enode Pro má len 2.9 z 5. Množstvo recenzií je pomerne obmedzené, aplikácia Vitruve

nemala dostatok recenzií, aby bolo možné si utvoriť určitý prehľad, no recenzie u ďalších dvoch aplikácií je možné zhrnúť následovne:

- RepOne Personal – užívatelia kladne hodnotia zariadenie, no dizajn aplikácie a užívateľské rozhranie by mohlo byť prívetivejšie. Merané metriky v rámci aplikácie by mohli byť lepšie pomenované, chýba ich vysvetlenie a po zadaní nového cviku, si ho aplikácia nie vždy uloží do zoznamu známych cvikov, z ktorých si užívateľ vyberá pri popise série.
- Enode Pro – dlhá doba párovania zariadenia, meranie spotrebováva príliš veľa baterky a často sa stáva, že opakovania nie sú zaznamenané v prípade, že jedinec ich vykonáva s pomalším tempom, či pauzami medzi fázami cviku. Užívatelia sa taktiež sťažujú na časté havárie aplikácie.

3.2 Bezdrôtové senzory

Príkladom bezdrôtového senzoru je zariadenie Flex[7], ktoré je možné nasadiť na koniec olympijskej tyče. Funkciu zabezpečuje pole laserov umiestnené na senzore a špeciálna reflektívna podložka, ktorú je potrebné umiestniť na zem pod daný senzor, pre získanie presných dát.

Obr. 3.3: Bezdrôtový senzor Flex, umiestnený na konci olympijskej tyče. Obrázok prevzatý z [49].

K zariadeniu je opäť dostupná aplikácia s názvom Flex Stronger[8], ktorá má hodnotenie 4.8 z 5 v službe App Store, hodnotenie v službe Google Play nebolo k januáru 2024 dostupné. Užívatelia si chvália prevažne funkcionalitu, analýzu progresu v rámci jednotlivých cvikov a taktiež sa im pozdáva zobrazenie trasy tyče v rámci jednotlivých opakovaní.

3.3 Aplikácie založené na princípoch počítačového videnia

V sekcii sa budeme venovať aplikáciam MetricVBT[9], RepSpeed[10] a WL Analysis[11]. Takéto aplikácie snímajú jednotlivca pomocou kamery mobilného zariadenia a následne využívajú

[7] FLEXStronger
[8] App Store – Flex Stronger
[9] App Store – MetricVBT
[10] Google Play – RepSpeed
[11] Google Play – WL Analysis

rôzne algoritmy počítačového videnia na detekciu a sledovanie činky, rozlíšenie jednotlivých opakovaní v rámci série a výpočet rýchlosti, či iných metrík.

Aplikácie MetricVBT

Aplikácia MetricVBT je dostupná len pre zariadenia s operačným systémom iOS a nepodporuje Android, na základe recenzií však možno usúdiť, že užívatelia sú s aplikáciou pomerne spokojní. K januáru 2024 dosahuje hodnotenia 4.8 z 5, užívateľom sa pozdáva prevažne jednoduchosť ovládania a samotná funkcionalita aplikácie. Mimo iné aplikácia ponúka zaznamenávanie rýchlostných metrík a trasy tyče v rámci série, analýzu výkonu na základe historických trendov, prípadne nahrávanie videozáznamu, či taktiež možnosť manuálne si zapísať výkon aj pre cviky, u ktorých nie je nahrávanie rýchlostných metrík podporované, aby užívatelia mali všetko na jednom mieste. Ďalšou výhodou je publikovaný článok [46], kde bola nezávisle otestovaná presnosť a spoľahlivosť aplikácie na cvikoch ako je mŕtvy ťah, drep, či bench press, pričom aplikácia dosiahla pomerne dobré výsledky.

Aplikácie RepSpeed a WL Analysis

Aplikácie RepSpeed a WL Analysis už majú o čosi horšie hodnotenia. Po vyskúšaní oboch aplikácií a na základe recenzií je možné zhodnotiť, že užívateľské rozhranie týchto aplikácií je už menej prívetivejšie ako v prípade aplikácie MetricVBT. Aplikácia WL Analysis, zatiaľ nedokáže poskytnúť spätnú väzbu v reálnom čase, keďže funkcionalita je ešte len vo vývoji. Aplikácia RepSpeed síce danú funkcionalitu poskytuje, ale má určité nedostatky. Počas zaznamenávania aplikácia totiž nezobrazuje detekovaný objekt, či trasu činky a taktiež nie je možné využiť prednú kameru telefónu. Obe aplikácie dokážu spätne analyzovať nahraté videá, tie však musí užívateľ predom upraviť tak, aby tesne na začiatku bolo prvé opakovanie série a na konci posledné. Aplikácie sa teda nevedia vysporiadať s prípadnou manipuláciou činky pred začatím a po skončení samotnej série, napr. pri drepe je potrebné, aby jedinec zdvihol činku zo stojanu, urobil niekoľko krokov vzad, vykonal sériu a následne vrátil činku naspäť do stojanu. WL Analysis taktiež neanalyzuje jednotlivé opakovania a ponúka len agregované metriky pre celú sériu.

3.4 Špecifikácia základnej funkčnosti aplikácie

Výsledná aplikácia predovšetkým musí implementovať funkcionalitu analýzy rýchlostných metrík jednotlivých sérii v reálnom čase pomocou kamery mobilného zariadenia.

Je potrebné, aby algoritmus využitý na analýzu bol schopný sám detekovať jednotlivé opakovania v rámci série a bol odolný voči manipulácii s činkou pred začiatkom a po skončení série. Aplikácia by taktiež mala byť schopná poskytnúť užívateľovi počas cvičenia spätnú väzbu podľa jeho preferencií, napr. ako zvukový signál, ak v rámci série klesne rýchlosť opakovania pod stanovený prah. Výhodou by bolo, ak by aplikácia ponúkala možnosť nahrať videozáznam spolu s analýzou a zobrazovala aspoň detekovaný objekt alebo trajektóriu tyče počas cviku, aby užívateľ vedel, že aplikácia funguje. Kvôli komfortu užívateľov je potrebné umožniť používanie prednej aj zadnej kamery mobilného zariadenia.

Aplikácia by mala poskytovať široké spektrum jasne definovaných metrík, pričom užívateľ by mal byť schopný si vybrať, ktoré metriky ho zaujímajú a tie budú následne v rámci aplikácie zobrazené.

Keďže užívateľov budú často zaujímať ich predchádzajúce výkony, aplikácia by im mala vhodným spôsobom umožniť popísať jednotlivé série pred nahrávaním a následne na základe týchto informácií implementovať filtrovanie historických dát. Príkladom takýchto údajov môže byť názov cviku, zdvíhaná váha, počet vykonaných opakovaní, dátum a čas vykonania série, či rôzne štítky na označenie sérií. Dôležité je, aby implementácia bola intuitívna a umožňovala užívateľom spomínané údaje rýchlo a jednoducho zadať počas tréningu.

Základnú funkcionalitu je ďalej možné rozšíriť napríklad o predikciu 1RM, či odhadnutie skóre RPE na základe historických dát užívateľa pre daný cvik. Taktiež by bolo možné zobraziť užívateľom rôzne trendy v ich výkone, ktoré by ich mohli zaujímať, poprípade umožniť spätnú analýzu nahraných videozáznamov.

Kapitola 4

Android a vývoj aplikácií

Android je open source mobilný operačný systém založený na báze Linuxu určený primárne pre mobilné zariadenia, akými sú chytré telefóny, či tablety, no je taktiež využívaný ako operačný systém chytrých televízorov, hodiniek a ďalších zariadení. V rámci následujúcej kapitoly sa budeme venovať predovšetkým vývoju aplikácií pre mobilné zariadenia, pre ktoré je výsledná aplikácia prevažne určená.

Vývoj Androidu vedie firma Google pod záštitou konzorcia firiem *Open Handset Alliance* v rámci projektu *Android Open Source Project* (AOSP), ktorý zverejňuje zdrojové kódy Androidu, aby ich mohol ktokoľvek použiť. Android sám o sebe (tak ako je vyvíjaný AOSP) však zaisťuje len základný beh systému a väčšina zariadení beží na proprietárnej verzii Androidu, ktorá poskytuje ďalšie dôležité služby.

Na začiatku následujúcej kapitoly stručne zhrnieme architektúru operačného systému Android. Zbytok kapitoly sa potom venuje základom vývoja aplikácií pre platformu Android a platforme Jetpack Compose, ktorá bola využitá pre tvorbu aplikácie. Text je čerpaný prevažne z oficiálnej Android dokumentácie [12], ale taktiež z knihy [43].

4.1 Architektúra Androidu

Pri vývoji aplikácií je vhodné aspoň do určitej mieri porozumieť architektúre Androidu, zloženej z niekoľkých, navzájom komunikujúcich vrstiev, kde každá z nich má určitú rolu. Jedná sa o následovné vrstvy:

- **Jadro (Linux kernel)** – základ architektúry tvorí Linuxové jadro, ktoré slúži ako úroveň abstrakcie medzi hardvérom mobilného zariadenia a vyššími vrstvami architektúry. Mimo iné, jadro zabezpečuje preemptívny multitasking, nízko úrovňové služby, akými sú napr. správa pamäte, procesov a správa napájania zariadenia. Taktiež zabezpečuje základnú sieťovú vrstvu a ovládače pre jednotlivé zariadenia, ako napr. display, Wi-Fi, či audio.

- **Android Runtime (ART)** – vrstva zabezpečujúca beh a preklad aplikácií. Aplikácie sú pri inštalácii dopredu preložené z bytekódu do natívnych inštrukcií (*ahead-of-time compilation*). Vrstva následne využíva modifikovanú *Java Virtual Machine* navrhnutú s ohľadom na parametre mobilných zariadení a prispôsobenú pre beh niekoľkých aplikácií súčasne.

- **Knižnice** – ďalšiu vrstvu tvoria knižnice, typicky napísane v C/C++, Jave alebo Kotline, ktoré sú neodkladnou súčasťou pri vývoji aplikácií. Jedná sa napríklad o knižnice

Graphics, *Libc*, *SSL*, *SQLite*, *Webkit*, *OpenGL*, *Surface Manager*. V rámci vývoja často nepracujeme priamo s natívnymi knižnicami, ale skôr využívame ich nadstavby napísané v Jave alebo v Kotline. V prípade potreby, je však možné využiť *Android Native Development Kit* a pristupovať k natívnym knižniciam (napísaných v C alebo C++) priamo v rámci kódu aplikácie napísaného v Jave alebo v Kotline.

- **Aplikačný Framework** – nad vrstvou knižníc je implementovaný aplikačný framework napísaný v Jave, ktorý poskytuje aplikáciam rôzne vysoko-úrovňové služby prostredníctvom svojho API. Android aplikácie sú teda tvorené zo znovu-použiteľných a zameniteľných komponent. Medzi poskytovanými službami sú napríklad
  - *Activity Manager* – spravuje životný cyklus aplikácií a zásobník aktivít,
  - *Resource Manager* – ponúka prístup k definovaným zdrojom, ako sú textové reťazce, nastavenia farieb a layouty používateľského rozhrania, ktoré nie sú súčasťou kódu,
  - *View System* – sada prostriedkov pre tvorbu užívateľských rozhraní

  a iné.
- **Aplikácie** – tvorí najvyššiu vrstvu architektúry operačného systému a je zložená zo samotných aplikácií nainštalovaných na danom zariadení.

4.2 Základné princípy Android aplikácií

Aplikácie pre systém Android je možné písať v rôznych programovacích jazykoch ako napr. Kotlin, Java, či C++. Pomocou sady nástrojov Android *Software Development Kit* (SDK) je možné kód preložiť a spolu s ďalšími dátami a zdrojmi vytvoriť súbor `.apk` (*Android Package*), či `.aab` (*Android App Bundle*). APK archív využívajú zariadenia s operačným systémom Android na inštaláciu aplikácií, keďže tento archív obsahuje všetko potrebné pre spustenie a beh danej aplikácie. Archív typu `.aap` slúži na zverejnenie aplikácie napr. pomocou služby Google Play. Tento archív taktiež obsahuje všetky kompilované súbory a dáta, no generovanie a podpis APK archívu však ponecháva napr. na samotný Google Play, ktorý dokáže pre užívateľov automaticky vygenerovať a poskytnúť k stiahnutiu optimalizovaný APK archív pre ich konkrétne zariadenie.

Android aplikácie bežia izolovane od ostatných aplikácií v rámci vlastného virtuálneho stroja (*Virtual Machine*, VM). V rámci Androidu figuruje každá aplikácia ako samostatný užívateľ systému s vlastným *user ID*, ktoré však aplikácie nepoznajú a je viditeľné len operačnému systému. Operačný systém následne nastaví oprávnenia k súborom tak, aby aplikácie mohli pristúpiť len k vlastným súborom. Typicky Android aplikácie bežia ako vlastný proces, ktorý je spustený vždy, keď je potrebné vykonať určitú komponentu aplikácie. Procesy sú ukončené, ak už nie sú potrebné alebo v prípade, že systém potrebuje uvoľniť pamäť a zdroje pre iné aplikácie.

Komponenty aplikácie

Android aplikácie sa skladajú z komponent, pričom každá komponenta predstavuje vstupný bod aplikácie pomocou, ktorého ju užívateľ alebo systém môže spustiť. Existujú štyri typy komponent:

- **Aktivity** (*Activities*) – jedná sa o triedy, ktoré užívateľovi prezentujú obsah a umožňujú mu interakciu s aplikáciou. Typicky sa aplikácie skladali z niekoľkých aktivít, ktoré predstavovali jedinú obrazovku aplikácie spolu s užívateľským rozhraním. V rámci Jetpack Compose je stále možné vytvoriť niekoľko aktivít v rámci aplikácie, no typicky sa v projekte nachádza jediná primárna aktivita a jednotlivé obrazovky sú implementované pomocou kompozitných funkcií, viď 4.4.

- **Služby** (*Services*) – sú to komponenty, vykonávajúce dlhotrvajúce operácie v pozadí. Služby neponúkajú užívateľské rozhranie. Majú pomerne dlhú životnosť a pretrvávajú aj po prepnutí do inej aplikácie. Služba môže napríklad obsluhovať sieťové transakcie, prehrávať hudbu, vykonávať vstupno-výstupné operácie so súbormi, alebo interagovať s poskytovateľom obsahu, a to všetko na pozadí.

- **Broadcast recievers** – komponenta umožňujúca komunikáciu medzi aplikáciami mimo bežného užívateľského toku, fungujúca na princípe návrhového vzoru *publish-subcribe*. Pri zaslaní správy sa systém automaticky postará o ich doručenie zaregistrovaným aplikáciam. Registrované aplikácie teda môžu reagovať na rôzne systémové udalosti, o ktorých ich informuje operačný systém pomocou broadcastového vysielania, alebo môžu reagovať na správy vytvorené ďalšími aplikáciami.

- **Poskytovatelia obsahu** (*Content Providers*) – poskytujú aplikáciam rozhranie pre správu a prístup k vlastným dátam, alebo k dátam iných aplikácií a teda umožňujú zdieľať dáta medzi viacerými aplikáciami. Zapuzdrujú dáta a implementujú rôzne bezpečnostné mechanizmy pre ich ochranu a správu prístupu.

Jednotlivé typy komponent majú špecifické účely a životné cykly, ktoré, mimo iné, určujú ako a kedy je daná komponenta vytvorená a ukončená.

Súbor manifest

Je potrebné aby v rámci každej aplikácie bol v koreňovom adresári zdrojového kódu definovaný súbor s názvom `AndroidManifest.xml`, ktorý obsahuje základné údaje o aplikácii. Mimo iné, v rámci tohto súboru sú uvedené všetky komponenty, ktoré aplikácia využíva a ich popis. V súbore sú uvedené všetky povolenia, ktoré aplikácia vyžaduje, ako napr. prístup k internetovému pripojeniu, či možnosť vidieť zoznam kontaktov. Ďalej tu špecifikujeme hardvérové, či softvérové požiadavky, ako je napr. prístup ku kamere. V rámci súboru taktiež definujeme minimálnu úroveň API viazanú s verziou operačného systému, potrebnú pre chod aplikácie.

4.3 Perzistentné ukladanie dát

Android ponúka niekoľko možností perzistentného ukladania dát. V rámci aplikácie nie sú využívané dáta iných užívateľov a preto si vystačíme s lokálnymi úložiskami dát, konkrétne využijeme knižnicu *Room*[1] a *DataStore*[2].

Room knižnica predstavuje abstrakčnú vrstvu nad relačnou SQLite databázou. Ponúka vývojárom možnosť jednoducho definovať štruktúru databázy pomocou anotovaných entitných tried, reprezentujúcich jednotlivé tabuľky databázy. DataStore slúži na robustné a

[1] Android dokumentácia – Room
[2] Android dokumentácia – DataStore

flexibilné ukladanie údajov typu kľúč-hodnota alebo typovaných objektov a je určený primárne pre uchovávanie menších objemov dát, ako sú užívateľské preferencie, či nastavenia aplikácie.

4.4 Framework pre vývoj natívneho užívateľského rozhrania

Jetpack Compose[3] je moderná sada nástrojov, vytvorená firmou Google, slúžiaca pre tvorbu natívnych užívateľských rozhraní pre Android. Jetpack Compose využíva programovací jazyk Kotlin[4] a jeho cieľom je zjednodušiť vývoj užívateľských rozhraní využitím deklaratívneho prístupu vďaka, ktorému je možné stručnejšie popísať jednotlivé časti užívateľského rozhrania. Informácie v následujúcej sekcii sú čerpané prevažne z oficiálnej dokumentácie [13] a kurzov [10].

Kompozitné funkcie

Kompozitné (*composable*) funkcie definujú užívateľské rozhranie a sú neoddeliteľnou súčasťou framework-u Jetpack Compose. Každá z takýchto funkcií reprezentuje špecifickú časť užívateľského rozhrania. Takto definované funkcie je potom možné rôznym spôsobom kombinovať a zanorovať tak, aby sme popísali celé užívateľské rozhranie. Výhodou takéhoto prístupu je jednoduchá znovu-použiteľnosť a flexibilita funkcií.

Kompozitné funkcie sú definované anotáciou `@Composable` a je možné ich volať len v rámci iných kompozitných funkcií, čím tvoríme určitú hierarchiu volaní a popisujeme užívateľské rozhranie organizovaným a modulárnym spôsobom. Jedná sa o deklaratívne funkcie, ktoré popisujú, ako má užívateľské rozhranie vyzerať na základe stavu dát a framework už zabezpečí znovu-vykreslenie potrebných elementov, v prípade, že dôjde k zmene stavu.

Pri kompilácií kódu dochádza k transformácií kompozitných funkcií na natívne elementy užívateľského rozhraniu podporované platformou Android.

Architektúra aplikácií

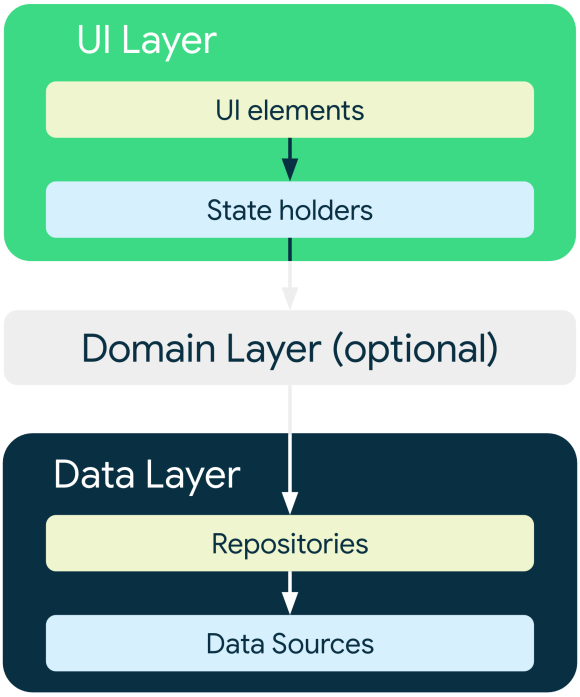
Dobre navrhnutá architektúra aplikácie nám pomáha správne rozdeliť zodpovednosť medzi jednotlivé triedy v kóde, umožňuje jednoduchšie škálovanie aplikácie a pridávanie novej funkcionality. Doporučená architektúra Android aplikácia sa snaží dodržať určité princípy, ako napr. *Separation of Concerns* (SoC), *Single Source of Truth* (SSoT) a *Unidirectional Data Flow* (UDF). Ďalšie princípy a ich význam je popísaný v [14].

Typicky je aplikácia štruktúrovaná do troch vrstiev a to:

- **Prezentačná vrstva** (*UI Layer*) – úlohou prezentačnej vrstvy je zobraziť užívateľom dáta na obrazovke mobilného zariadenia a taktiež reagovať na ich zmenu spôsobenú interakciou užívateľa so zariadením, či inými faktormi. Prezentačná vrstva je ďalej rozdelená na dve časti. Jedná sa o elementy užívateľského rozhrania, ktoré vykresľujú dáta na obrazovku. V Jetpack Compose ich definujeme pomocou kompozitných funkcií. Druhá časť implementuje logiku aplikácie a dočasne uchováva dáta, ktoré sprístupňuje jednotlivým elementom užívateľského rozhrania. Zvyčajne sa jedná o *ViewModel*[5] triedy.

[3] Jetpack Compsoe
[4] Kotlin
[5] Android dokumentácia – ViewModel

- **Doménová vrstva (*Domain Layer*)** – jedná sa o voliteľnú vrstvu medzi prezentačnou a dátovou vrstvou, ktorú je vhodné využiť v prípade, že v rámci aplikácie implementujeme komplexnejšiu business logiku, ktorú chceme zapuzdriť do vlastných tried a funkcií. Doménová vrstva sa taktiež využíva v prípade, že v rámci prezentačnej vrstvy opakovane implementujeme určitú logiku, ktorú by bolo možné vyňať zvlášť a následne len opätovne používať túto implementáciu v rámci ViewModel tried.
- **Dátová vrstva (*Data Layer*)** – dátová vrstva aplikácie má na starosti dáta aplikácie a business logiku. Skladá sa z tried, nazývaných repozitáre, ktoré pracujú s jedným, či až niekoľkými dátovými zdrojmi. Typicky vytvárame jednu repozitárnu triedu pre každý typ dát, s ktorým v rámci aplikácie pracujeme. Dátové zdroje sú taktiež reprezentované triedami a tie už priamo pracujú s jedným určitým zdrojom dát, ako napr. lokálna databáza, súbor, či iné vzdialené zdroje.



Obr. 4.1: Vrstvy typickej architektúry aplikácie, kde šipky znázorňujú závislosti medzi vrstvami, napr. prezentačná vrstva závisí na doménovej vrstve. Ilustrácia prevzatá z [14].

Kapitola 5

Detekcia a sledovanie objektov

Podstata detekcie objektov spočíva v lokalizácii a klasifikácii objektov v obrázkoch, či v rámci jednotlivých snímkov videa, pričom detekovaný objekt je typicky reprezentovaný obdĺžnikovým, ohraničujúcim, rámčekom, tzv. *bounding box*. Spolu s polohou, nás často zaujíma trieda, do ktorej daný objekt patrí, či skóre reprezentujúce dôveryhodnosť tejto detekcie. Detekcia objektov predstavuje dôležitú oblasť počítačového videnia a má široké uplatnenie jednak vo vedeckom výskume, ale aj v produkcii. Typicky sa jedná o oblasti ako detekcia tvári, textu, detekcia chodcov a automobilov z kamerových záznamov, či detekcia rôznych objektov v snímkoch z lekárskej oblasti. Na sledovanie pohybu detekcií v čase, je potom potrebné využiť *tracker*, ktorého cieľom je priradiť detekcie objektov k patričným trajektóriam.

V rámci kapitoly sú stručne vysvetlené spôsoby detekcie objektov. Nasleduje popis základných architektúr neurónových sieti pre spracovanie obrazu, ktoré sú súčasťou detektorov, spolu s vysvetlením detektoru EfficientDet a detektoru EfficientDet-Lite určeným pre mobilné zariadenia, pričom práve tento detektor je využitý v rámci výslednej aplikácie na detekciu váhových kotúčov. Na konci kapitoly sú uvedené algoritmy slúžiace na sledovanie detekcií v čase a taktiež je popísaný princíp algoritmu SORT.

5.1 Detekcia objektov

Vo všeobecnosti môže byť detekcia objektov vykonaná pomocou konvenčných metód spracovania obrazu alebo pomocou metód založených na hlbokom učení. Konvenčnými metódami spracovania obrazu typicky myslíme algoritmy, ktoré využívajú manuálnu extrakciu čŕt (*features*) z obrazu. Ručný návrh takýchto algoritmov je však obtiažny, keďže je potrebné, aby dané algoritmy boli robustné a fungovali v meniacich sa podmienkach [51]. To môže zahŕňať napríklad premenlivé úrovne osvetlenia scény, rozličné pozadia, čiastočné zakrytie hľadaných objektov a iné komplexné situácie.

Na druhú stranu metódy založené na hlbokom učení, teda neurónové siete, sú sami schopné naučiť sa robustne reprezentovať potrebné vlastnosti hľadaných objektov a stávajú sa teda čoraz populárnejšími [51]. Počas trénovania týchto modelov, je však potrebné využívať veľké množstvá anotovaných dát a takýto tréning môže byť výpočetne náročný. Spolu s rastúcou výpočetnou silou to však nepredstavuje príliš veľký problém a v posledných rokoch vidíme rozmach metód založených na hlbokom učení [51].

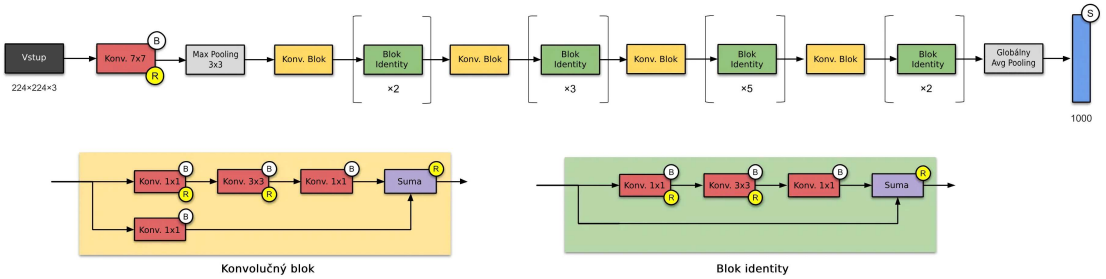
5.2 Architektúry neurónových sietí pre spracovanie obrazu

Základom modelov určených pre detekciu objektov je zvyčajne chrbticová (*backbone*) konvolučná neurónová sieť, viď [33], alebo jej varianty. Vstupom tejto siete je snímok, v rámci ktorého chceme detekovať jednotlivé objekty a jej výstupom sú extrahované črty daného snímku na rôznych úrovniach [34]. V počiatočných vrstvách tejto konvolučnej siete sme väčšinou schopní získať nízko-úrovňové rysy, avšak vo vysokom rozlíšení. Postupným propagovaním informácie cez jednotlivé vrstvy konvolučnej siete získavame rysy s čoraz nižším rozlíšením, ale vyššou informačnou hodnotou.

Príkladmi jednoduchších sietí sú siete AlexNet [26], či VGG-16 [42], zložené z prevažne konvolučných a poolingových vrstiev, nasledovaných niekoľkými plne prepojenými vrstvami. Ukázalo sa však, že využitie hlbších chrbticových sietí s väčším počtom vrstiev a parametrov, ako napr. ResNet [19], dokáže vylepšiť presnosť detekcií, pravdepodobne vďaka vyššej kvalite a informačnej hodnote extrahovaných čŕt. Okrem zvyšovania presnosti detekcií, nás však často zaujíma výpočetná náročnosť danej siete a to napr. v kontexte inferencie na mobilných zariadeniach za účelom sledovania pohybu v reálnom čase. Siete MobileNet [22], či EfficientNet [47], riešia tento problém a snažia sa minimalizovať svoju komplexitu pri zachovaní presnosti porovnateľnej s väčšími modelmi.

ResNet

Architektúry ResNet [19] využívajú reziduálne bloky, ktorých výstupom sú spracované dáta spolu s pripočítaným pôvodným vstupom, viď obrázok 5.1. Takéto zapojenie pomáha s propagáciou informácií cez jednotlivé vrstvy siete a to dopredným smerom počas inferencie, ale aj spätne v rámci upravovania váh algoritmom *gradient descent* počas tréningu. Neurónové siete tak môžu byť hlbšie, ako pri jednoduchších architektúrach, kde by sme počas trénovania mohli naraziť napríklad na problém miznúceho gradientu.

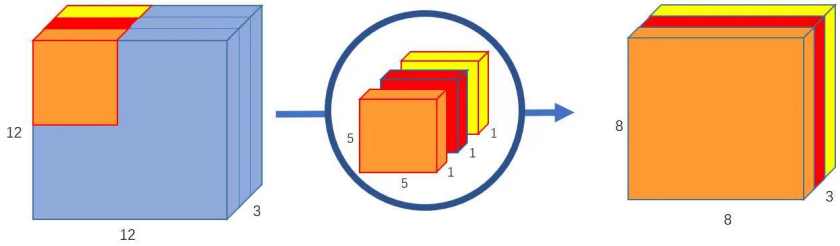


Obr. 5.1: Ukážka architektúry ResNet, kde `B` značí batch normalizáciu, `R` značí využitie aktivačnej funkcie ReLU a `S` znamená využitie sigmoidálnej aktivačnej funkcie. Obrázok prevzatý z [24].

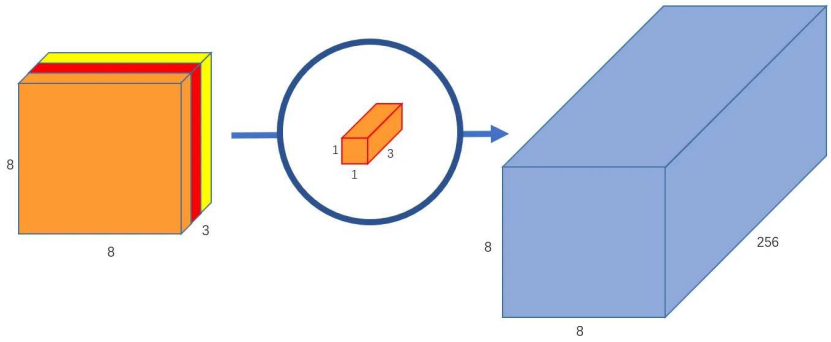
MobileNet

Modely MobileNet sa oproti predošlým architektúram sústredia na výpočetne limitované platformy, akými sú mobilné zariadenia, rôzne vstavané aplikácie, či robotika. Namiesto tvorenia väčších a komplikovanejších modelov, využívajú tzv. hĺbkovo separovateľné konvolúcie (*depthwise separable convolution*) [22].

Princíp hĺbkovo separovateľných konvolúcií spočíva v rozdelení bežnej operácie konvolúcie na dve časti. V prvom rade je vykonaná operácie konvolúcie bez zmeny hĺbky, teda počtu kanálov vstupu. To znamená, že máme rovnaký počet konvolučných jadier, ako kanálov na vstupe, pričom ich hĺbka je rovná jednej a každým jadrom spracujeme jeden odpovedajúci vstupný kanál (obr. 5.2). Na získaný výsledok v druhom kroku aplikujeme bodovú konvolúciu. Využívame jadrá, ktoré majú v prípade spracovania obrázkov rozmer 1×1 a ich hĺbka je rovná počtu vstupných kanálov. Takýchto jadier máme však niekoľko, v závislosti na požadovanej hĺbke výstupu (obr. 5.3).



Obr. 5.2: V prvom kroku vykonáme operáciu konvolúcie zvlášť pre každý vstupný kanál, pomocou troch kernelov o veľkosti $5 \times 5 \times 1$ teda transformujeme vstupný obrázok o rozmeroch $12 \times 12 \times 3$ na rozmery $8 \times 8 \times 3$. Obrázok prevzatý z [50].



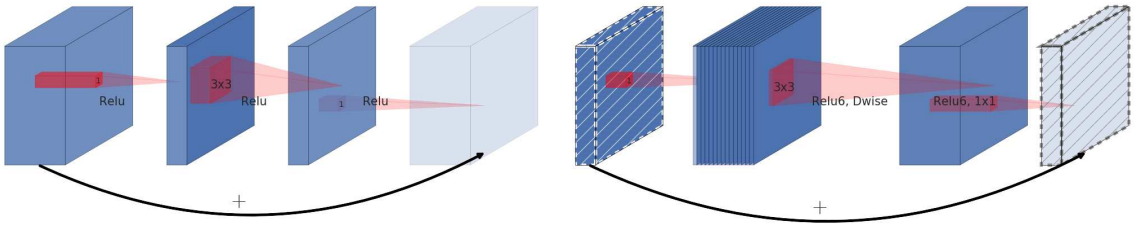
Obr. 5.3: V druhom kroku vykonáme bodovú konvolúciu pomocou 256 kernelov o veľkosti $1 \times 1 \times 3$, čím získame výstupný obrázok s 256 kanálmi. Obrázok prevzatý z [50].

Využitím hĺbkovo separovateľných konvolúcii boli autori schopný znížiť počet parametrov modelu a ich výpočetnú náročnosť, zatiaľ čo nameraná presnosť týchto modelov je porovnateľná s väčšími architektúrami [22]. Počty parametrov modelov MobileNet sa pohybujú len v jednotkách miliónov a priestor, ktorý zaberajú na disku, je taktiež rádovo menší v porovnaní s predchádzajúcimi modelmi.

MobileNetV2

Architektúra MobileNetV2 stavia na modeloch MobileNet, pričom využíva skratkové spojenia, podobné architektúre ResNet. Reziduálne bloky sú však invertované, čo znamená, že na začiatku a konci nájdeme úzke hradlá, na rozdiel od expandovaných reprezentácií vstupu v prípade tradičných reziduálnych blokov.

Vstupom takýchto blokov je teda nízko-dimenzionálny tenzor, ktorý je pomocou bodovej konvolúcie expandovaný do viac dimenzionálneho priestoru, vhodnejšieho pre aplikáciu nelineárnych aktivačných funkcií. Na tenzor teda následne aplikujeme aktivačnú funkciu ReLU6 [40], vykonáme hĺbkovú konvolúciu s jadrom o veľkosti 3×3 a opäť aplikujeme nelinearitu v podobe ReLU6. Takto spracovaný vysoko dimenzionálny tenzor je premietnutý späť do nízko-dimenzionálneho priestoru. Pri tejto projekcii však prirodzene dochádza k strate informácii a preto na vstup už ďalej neaplikujeme nelineárnu aktivačnú funkciu. Odstránenie poslednej nelinearity, podľa autorov, pomáha s uchovaním reprezentačnej schopnosti týchto blokov, čo v rámci pôvodného článku aj empiricky ukázali [40]. V prípade, že rozmery vstupu a výstupu invertovaných reziduálnych blokov sú totožné, môžeme k výstupu pripočítať pôvodnú hodnotu vstupu, rovnako ako v architektúrach ResNet.



Obr. 5.4: Porovnanie bežných reziduálnych blokov (vľavo) s invertovanými reziduálnymi blokmi (vpravo), využitými v architektúrach MobileNetV2. Obrázok prevzatý z [40].

EfficientNet

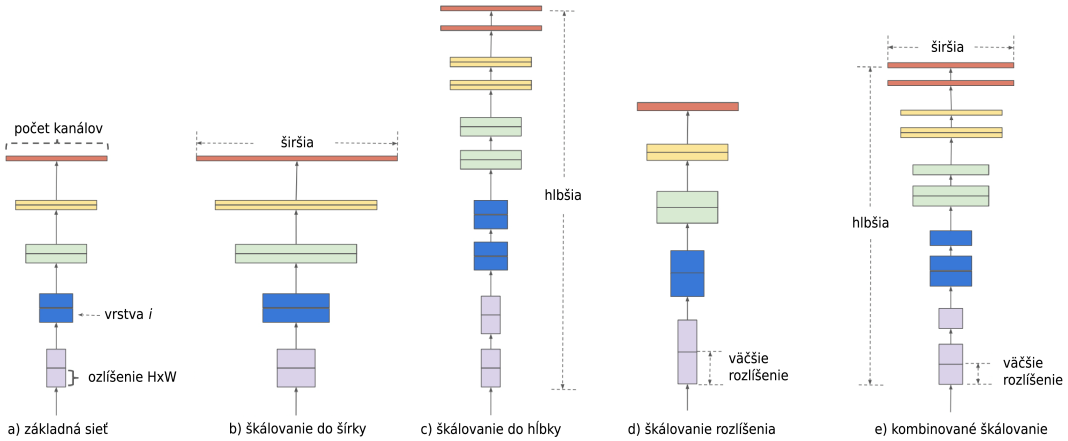
Ak chceme u modelov dosiahnuť väčšej presnosti, je zvyčajne potrebné ich určitým spôsobom škálovať, či už pridaním vrstiev (škálovanie do hĺbky) alebo zvýšením počtu kanálov v jednotlivých vrstvách (škálovanie do šírky). Taktiež je možné škálovať model zmenou rozlíšenia vstupných snímkov, kde z väčšieho rozlíšenia, by sme mali byť schopní získať komplexnejšie a detailnejšie črty. Škálovanie modelov len v jednej z dimenzií zvyčajne zlepšuje ich presnosť, no ladenie často vyžaduje manuálne upravovanie parametrov a zdĺhavé trénovanie.

Architektúra EfficientNet [47] sa spolieha na metódu nazývanú komplexné škálovanie (*compound scaling*), pri ktorej je model škálovaný vo všetkých dimenziách zároveň, teda do šírky, hĺbky a aj čo sa týka rozlíšenia vstupných snímkov, viď obr. 5.5. Škálovanie prebieha systematicky, vzhľadom na užívateľom definovaný škálovací koeficient φ, ktorým je schopný určiť celkovú komplexitu a výpočetnú náročnosť modelu. Vzhľadom na koeficient φ sú následne odvodené individuálne škálovacie koeficienty, teda koeficient pre škálovanie do hĺbky d, do šírky w a škálovanie rozlíšenia r, a to následovne

$$
\begin{aligned}
d &= \alpha^{\varphi}, \\
w &= \beta^{\varphi}, \\
r &= \gamma^{\varphi}, \\
&\text{kde } \alpha, \beta, \gamma \geq 1.
\end{aligned}
$$

Optimálne hodnoty α, β, γ sú nájdené vykonaním automatizovaného prehľadávania priestoru, so zafixovaným $\varphi = 1$ a s využitím určitého základného modelu, pričom cieľom je nájsť kombináciu parametrov, predstavujúcu najlepší kompromis medzi presnosťou modelu a jeho

výpočetnou náročnosťou, viď [47]. V prípade daného modelu, je možné vyjadriť nárast potrebných FLOPS ako $(\alpha \cdot \beta^2 \cdot \gamma^2)^\varphi$ [47]. Autori článku sa rozhodli obmedziť $(\alpha \cdot \beta^2 \cdot \gamma^2) \approx 2$ tak, aby pre zvolený škálovací koeficient φ, odpovedal nárast FLOPS modelu približne o 2^φ. Zafixovaním $\varphi = 1$ počas optimalizácie parametrov α, β, γ teda zväčšujeme model približne dvojnásobne. Pre základný model EfficientNet-B0 boli nájdené hodnoty $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.15$ a pomocou nich následne vytvorené modely EfficientNet-B1 až B7 [47].



Obr. 5.5: Ukážka rôznych typov škálovania. `a)` zobrazuje základnú sieť, `b)-d)` predstavujú bežné metódy škálovania, zväčšujúce len jednu dimenziu siete, `e)` ukazuje kombinované škálovanie všetkých dimenzií so zafixovaným pomerom, využité v architektúre EfficientNet. Obrázok prevzatý z [47].

Architektúra EfficientNet sa skladá z blokov nazývaných *Mobile Inverted Bottleneck*, ktoré sú podobné invertovaným reziduálnym blokom, vysvetlených v predchádzajúcej sekcii 5.2. Model taktiež využíva optimalizáciu *squeeze-and-excitation* [23] [47].

5.3 Spôsoby detekcie objektov

Prístupy k detekcii objektov založené na hlbokom učení vieme typicky rozdeliť do dvoch kategórií. V prvom prípade sa jedná o dvojstupňové (*two-stage*) detektory, ktoré najprv pomocou chrbticovej neurónovej siete navrhnú možné pozície objektov, tzv. regióny. Navrhnuté regióny, spolu s ich extrahovanými príznakmi sú následne klasifikované a navrhnutý bounding box môže byť ďalej spresnený [34]. Príkladmi takýchto detektorov sú R-CNN [8], Fast R-CNN [7], Faster R-CNN [37], FPN [29] a iné.

Jednostupňové (*one-stage*) detektory vykonávajú detekciu objektov priamo z obrázkov, bez prvého kroku, v ktorom sú vygenerované návrhy regiónov. Príkladmi takýchto detektorov sú YOLO [36], SSD [31] a iné.

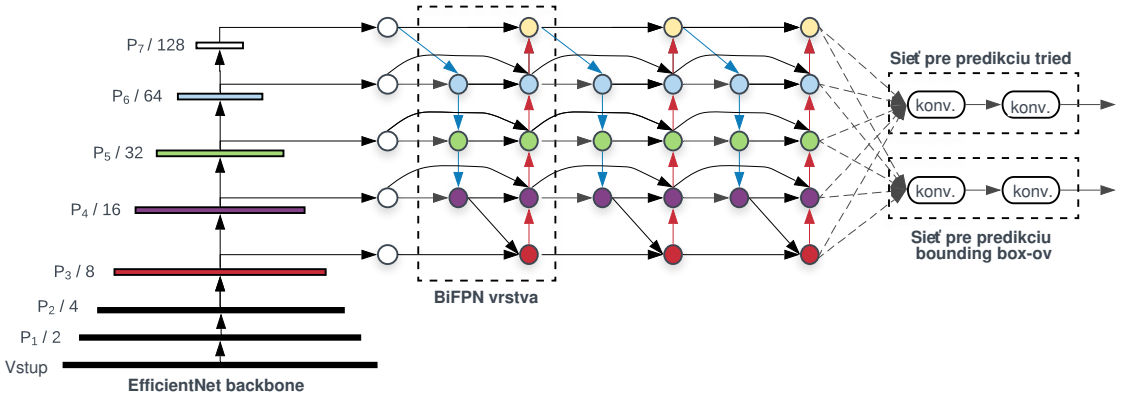
Typicky je presnosť dvojstupňových detektorov lepšia, ako v prípade jednostupňových detektorov, no jednostupňové detektory zvyčajne ponúkajú rýchlejšiu inferenciu [34].

EfficientDet

Motiváciou za vytvorením architektúry EfficientDet [48] pre detekciu objektov bola opäť minimalizácia výpočetnej náročnosti, pri zachovaní presnosti porovnateľnej s väčšími mo-

delmi. Autori sa inšpirovali architektúrou EfficientNet, konkrétne výsledkami dosiahnutými vďaka komplexnému škálovaniu modelu vo všetkých dimenziách zároveň a aplikovali podobný prístup na celú sieť určenú pre detekciu objektov.

Výsledný detektor využíva EfficientNet [47] ako chrbticovú sieť, po ktorej nasleduje niekoľko BiFPN [48] vrstiev pre extrakciu príznakov a ich výstup je spracovaný sieťami pre klasifikáciu a predikciu bounding boxov, viď obr. 5.6.



Obr. 5.6: Architektúra EfficientDet využíva EfficientNet ako chrbticovú sieť, BiFPN pre extrakciu príznakov a siete pre predikciu tried a bounding boxov. Vrstvy BiFPN a vrstvy siete pre výslednú predikciu sú opakované niekoľko krát v závislosti na konkrétnej verzii siete, viď [48]. Obrázok prevzatý z [48].

V článku boli taktiež predstavené vrstvy zvané *Bidirectional Feature Pyramid Network* (BiFPN) [47], ktoré extrahujú príznaky z niekoľkých vrstiev chrbticovej siete, pričom umožňujú obojsmerný tok informácii. Výstupy z jednotlivých úrovni chrbticovej siete však nemusia byť rovnako dôležité pre extrakciu príznakov a preto im sú v prípade BiFPN priradené váhy, ktoré sa model učí počas trénovania.

Jednotlivé verzie architektúry EfficientDet majú značenie D0 až D7, pričom číslice odpovedajú škálovaciemu koeficientu φ, t.j. $\varphi = 0$ pre EfficientDet-D0 a $\varphi = 7$ pre EfficientDet-D7 Modeli D1 až D7 boli vytvorené škálovaním základného modelu EfficientDet-D0. Presný výpočet nových rozmerov jednotlivých častí modelu je uvedený v [48].

EfficientDet-Lite

Modely EfficientDet-Lite [16] vznikli zmenšením a kvantizáciou architektúry EfficientDet tak, aby boli vhodné pre využitie na mobilných zariadeniach v reálnom čase. Google zverejnil niekoľko predtrénovaných TensorFlow Lite checkpointov, konkrétne sa jedná o modely EfficientDet-Lite0 až Lite-4. Tie je možné priamo využiť na detekciu určitých objektov, ale pomocou dodaných nástrojov, ako napr. TensorFlow Lite Model Maker [11], využitým v rámci tejto práce, dotrénovať na vlastnej špecifickej dátovej sade. Základne informácie o zverejnených checkpointoch sú uvedené v tabuľke 5.1.

Tabuľka 5.1: Veľkosť, odozva a presnosť zverejnených checkpointov [15]. Doba odozvy bola meraná na mobilných zariadeniach Google Pixel 4, s využitím štyroch CPU vlákien a mAP je metrika *mean Average Precision*, spočítaná na validačnej dátovej sade COCO 2017 [30].

Model	Veľkosť [MB]	Odozva [ms]	mAP
EfficientDet-Lite0	4.4	37	25.69%
EfficientDet-Lite1	5.8	49	30.55%
EfficientDet-Lite2	7.2	69	33.97%
EfficientDet-Lite3	11.4	116	37.70%
EfficientDet-Lite4	19.9	260	41.96%

5.4 Sledovanie objektov

V prípade, že nás nezaujíma len detekcia objektu z jedného snímku, ale jeho pohyb v čase, je potrebné využiť tracker, ktorý je schopný správne priradiť detekované objekty k jednotlivým trajektóriam, poprípade predikovať ich nasledujúcu pozíciu.

Poznáme trackere zvané *Single Object Tracker* (SOT) sledujúce len pohyb jediného objektu, ktoré sú väčšinou schopné po prvotnej inicializácii sledovať daný objekt v následujúcich snímkoch. Na rozdiel od nich, *Multiple Object Tracker* (MOT) sú schopné sledovať pohyb niekoľkých detekovaných objektov, rôznych kategórií, zároveň.

SORT

Algoritmus s názvom *Simple Online and Realtime Tracking* (SORT) [4] predstavuje minimálny tracker, umožňujúci sledovanie niekoľkých objektov so zameraním na rýchlosť inferencie a sledovanie v reálnom čase, vďaka čomu je tento algoritmus vhodným kandidátom pre využitie vo výslednej aplikácii. Princíp algoritmu je následovný:

1. V prvom kroku je vykonaná predikcia nasledujúcich pozícií bounding boxov pomocou Kalmanovho filtra, pre každú, z už existujúcich trajektórií.

2. Následne je spočítaná matica cien pre všetky možnosti priradenia nových detekcií k jednotlivým existujúcim trajektóriam, kde ceny sú vyjadrené ako *Intersection over Union* (IoU) detekovaného a predikovaného bounding boxu. Pre minimalizáciu celkovej ceny a získanie výsledného priradenia detekcií k trackerom je využitý algoritmus Kuhn–Munkres, tzv. *Hungarian algorithm*.

3. V poslednom kroku sú aktualizované Kalmanove filtre pre jednotlivé trajektórie ich priradenými detekciami. Pre detekcie, ktoré neboli priradené sú vytvorené nové trajektórie a taktiež trajektórie, ktoré neboli aktualizované určitý počet snímkov, sú vymazané.

Kapitola 6

Návrh

Následujúca kapitola je venovaná návrhu aplikácie, čo sa týka užívateľského rozhrania a štruktúry databázy. Popísané sú jednotlivé obrazovky aplikácie, navigácia medzi nimi a princíp analýzy tréningu, z pohľadu užívateľa. Taktiež sa časť kapitoly venuje výberu technológií použitých pre implementáciu užívateľského rozhrania aplikácie, či databázy, spolu s uvedením ER diagramu a vysvetlenia jednotlivých entít.

6.1 Užívateľské rozhranie aplikácie

Pred samotnou realizáciou aplikácie bol v prvom rade vytvorený návrh užívateľského rozhrania tak, aby boli splnené základné požiadavky na aplikáciu, definované v sekcii 3.4. Zároveň bol kladený dôraz na to, aby vzhľad aplikácie bol prehľadný, jej ovládanie intuitívne a efektívne, keďže ju užívatelia budú využívať počas cvičenia a nebolo by vhodné ich naviac zbytočne zaťažovať. Výsledný návrh, vytvorený v nástroji Figma[1], sa skladá z troch hlavných obrazoviek a to obrazovka *Workout*, *History* a *Settings*.

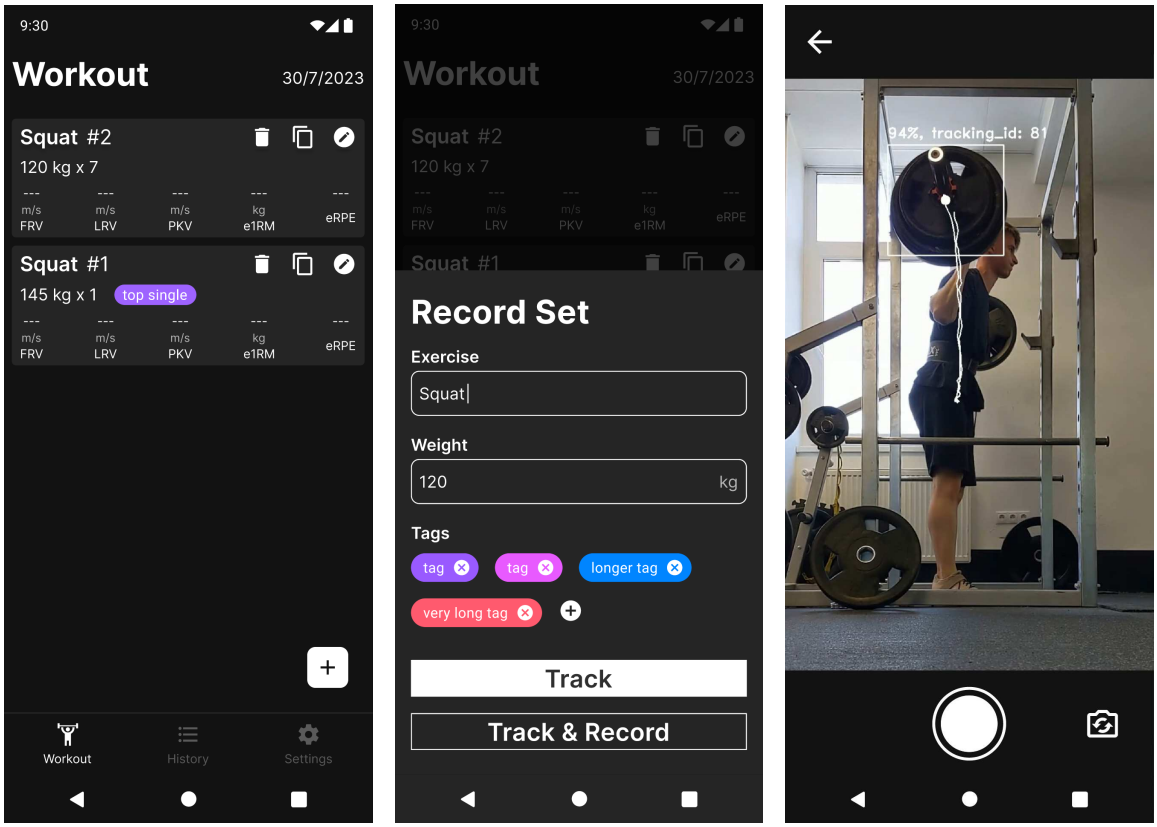
Obrazovka *Workout*

Cieľom obrazovky *Workout* je zobraziť užívateľovi prehľad sérií, ktoré boli zaznamenané v rámci aktuálneho tréningu a otvorením modalu *Record Set*, mu umožniť jednoducho špecifikovať údaje o následujúcej sérii a začať jej analýzu prechodom do obrazovky *Record*.

Náhľady vykonaných sérii zobrazujú základné údaje, ako napr. názov cviku, počet vykonaných opakovaní, či rôzne štítky so stručnými informáciami. Po kliknutí na kartu náhľadu bude užívateľ presmerovaný do obrazovky *Set Details*, ktorá už obsahuje zaznamenané rýchlostné metriky jednotlivých opakovaní v rámci série. V rámci náhľadu série sú taktiež zobrazené tlačítka pre úpravu zadaných údajov, vymazanie série, či jej skopírovanie, po ktorého stlačení bude otvorený modal *Record Set*, predvyplnený údajmi z danej série, čo urýchli ovládanie aplikácie v prípade, že užívateľ chce analyzovať niekoľko sérií rovnakého cviku.

Modal *Record Set* umožňuje užívateľovi definovať základné údaje o nahranej sérii a je ďalej možné ho rozšíriť napr. o nastavenie poskytnutia spätnej väzby počas cvičenia. Tlačidlá *Track* a *Track & Record* presmerujú užívateľa na obrazovku *Record*, pričom tlačidlo *Track & Record* zabezpečí, že okrem analýzy série bude taktiež nahraný aj video záznam.

[1] návrh aplikácie vo Figme



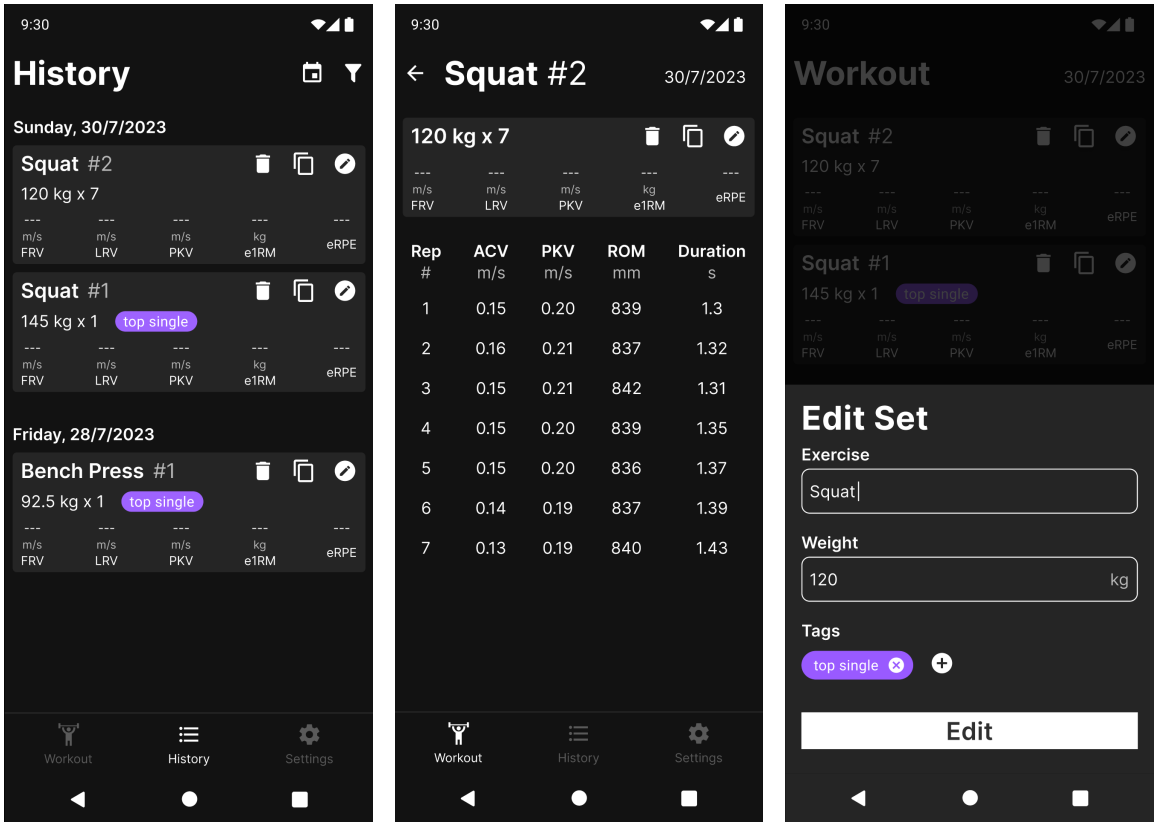
(a) Obrazovka *Workout*. (b) Modal *Record Set*. (c) Obrazovka *Record*.

Obr. 6.1: Ilustrácia obrazovky *Workout*, poskytujúcej prehľad o sériach v rámci aktuálneho tréningu. Po stlačení tlačidla pre zaznamenanie ďalšej série, alebo priamo okopírovaním, už existujúcej série sa otvorí modal *Record Set*, z ktorého sa po doplnení údajov môže užívateľ presunúť do obrazovky *Record*.

Obrazovku *Record* tvorí obraz z kamery mobilného zariadenia, spolu s vykreslenými bounding boxmi detekovaných kotúčov. V prípade, že detektor nájde v obraze viacero váhových kotúčov, môže užívateľ kliknutím na bounding box označiť, ktorý z nich sa bude sledovať v rámci vykonanej série cviku. Následne kliknutím na tlačidlo nahrávania, spustí sledovanie a analýzu jednotlivých opakovaní. Podporované je využitie ako prednej, tak aj zadnej kamery telefónu. Počas zaznamenávania aplikácia detekuje jednotlivé fázy cviku a počíta potrebné metriky, pričom po ukončení bude užívateľ presmerovaný na obrazovku *Set Details*, zobrazujúcu detailnejšie údaje o práve analyzovanej sérii. Pre nameranie presnejších dát bude potrebné, aby užívatelia nasmerovali kameru mobilného zariadenia tak, aby snímala činku s naloženými závažiami z boku tak, aby dochádzalo k čo najmenšiemu skresleniu závaží a detekovaných bounding boxov.

Obrazovka *History*

Prehľad o sériach z minulých tréningov poskytuje obrazovka *History*. Náhľady sérii sú totožné s obrazovkou *Workout*. Okrem toho však poskytuje obrazovka *History* možnosť filtrovať historické údaje napr. podľa cviku. V budúcnosti by bolo možné rozšíriť obrazovku o pokročilejšiu analýzu trendov vo výkone, či export dát.

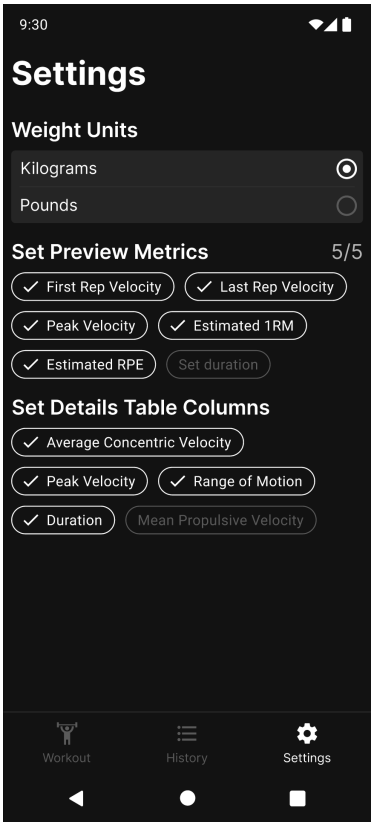


(a) Obrazovka *History*. (b) Obrazovka *Set Details*. (c) Modal *Edit Set*.

Obr. 6.2: Ilustrácie obrazovky *History*, ktorá informuje užívateľa o všetkých jeho analyzovaných sériach. Po kliknutí na náhľad série je užívateľ presmerovaný na obrazovku *Set Details* zobrazujúcu metriky pre jednotlivé opakovania v rámci série. Modal *Edit Set* je využitý pre úpravu zadaných informácií o analyzovanej sérii.

Obrazovka *Settings*

Obrazovka *Settings* bude užívateľom ponúkať možnosť prispôsobiť si aplikáciu podľa vlastných potrieb. Môže sa jednať o voľbu jednotiek pre zadanie váhy, voľbu metrík zobrazených v obrazovke *Set Details*, nastavenie rôznych parametrov detektoru ako napr. nastavenie detekčného prahu, či minimálny rozsah pohybu pre detekciu opakovaní.



Obr. 6.3: Ilustrácia obrazovky *Settings* poskytujúca užívateľom možnosť prispôsobiť si aplikáciu vlastným potrebám.

Voľba aplikačného frameworku

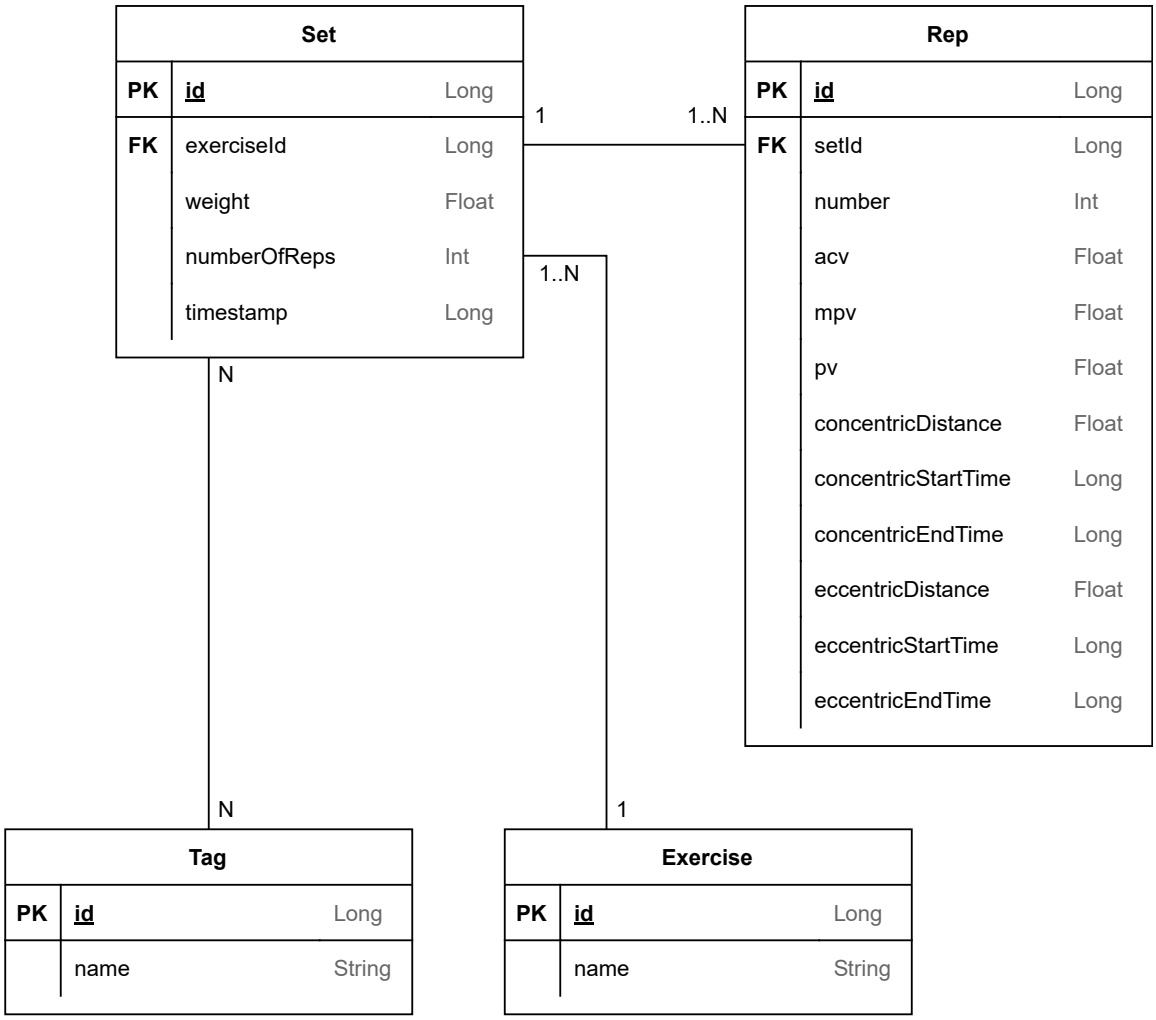
Vzhľadom na to, že navrhovaná aplikácia je aktuálne zameraná len na mobilné zariadenia s operačným systémom Android, bol pre jej tvorbu zvolený najnovšie odporúčaný framework Jetpack Compose, popísaný v kapitole 4.4. Framework Jetpack Compose podporuje využitie *Material Design 3*[2], vyvinutého spoločnosťou Google. Jedná sa o dizajnový jazyk, doporučený pre tvorbu Android aplikácií, ktorého cieľom je zjednotenie užívateľskej skúsenosti naprieč rôznymi zariadeniami. Do budúcna by bolo taktiež možné rozšíriť aplikáciu na mobilné zariadenia s operačným systémom iOS, využitím frameworku *JetBrains - Compose Multiplatform*[3].

6.2 Návrh databáze

Perzistentné uloženie dát bude zabezpečovať off-line lokálna databáza Room, popísaná v kapitole 4.3. Bude však potrebné štruktúrovať kód aplikácie tak, aby bolo v budúcnosti možné prípadné rozšírenie a využitie cloudových riešení. Spočiatku však vďaka využitiu lokálnej databázy, v prípade zverejnenia aplikácie, nebude potrebné riešiť správu a škálovanie vlastných serverov, či tarify v prípade poskytovateľov cloudových služieb a lokálna databáza by nemala užívateľov príliš ovplyvňovať pri analýze ich tréningu.

[2] Material Design 3
[3] Compose Multiplatform

Mimo relačné dáta bude potrebné ukladať aj jednoduché údaje typu kľúč-hodnota v prípade užívateľských preferencií, čo zabezpečíme využitím *DataStore*, taktiež popísaného v kapitole 4.3.



Obr. 6.4: ER diagram zobrazujúci navrhnutú štruktúru databáze v rámci aplikácie.

Entita *Set*

Entity `Set` reprezentuje jednu analyzovanú sériu určitého cviku. Obsahuje cudzí kľúč `exerciseId`, ktorým sa odkazuje na entitu `Exercise` definujúcu, o aký cvik ide. Ďalším atribútom je `weight`, ktorý predstavuje váhu zdvíhanú v rámci danej série. Atribút `numberOfReps` hovorí o počte opakovaní vykonaných v rámci série, a síce sa jedná o redundantný údaj, uloženie v entite `Set` uľahčí dotazovanie pri zobrazovaní náhľadov vykonaných sérii. Posledný atribút `timestamp` obsahuje časové údaje o tom, kedy bola séria vykonaná. K sérii môže byť priradených niekoľko štítkov, entita `Tag`, umožňujúcich užívateľom pridať rozširujúce informácie.

Entita *Rep*

Entita `Rep` predstavuje jedno opakovanie vykonané v rámci série, na ktorú sa odkazuje pomocou cudzieho kľúča `setId`. Atribút `number` špecifikuje poradie opakovania v rámci série, t.j. o koľké opakovanie sa jedná.

Atribúty `acv`, `mpv` a `pv` predstavujú rýchlostné metriky ACV, MPV a PV, popísané v sekcii 2.2. `concentricDistance` je dĺžka trajektórie, prejdenej počas koncentrickej fázy opakovania, v metroch, `concentricStartTime` a `concentricEndTime` sú časové razítka v milisekundách, počiatku a konca koncentrickej fázy. Obdobné informácie sú zaznamenané aj v prípade excentrickej fázy opakovania.

Kapitola 7

Realizácia

Kapitola sa zaoberá samotnou realizáciou aplikácie. V prvom rade sa venuje procesu trénovania vlastného modelu pre detekciu váhových kotúčov, kde sú spomenuté využité knižnice, dátová sada spolu s názornou ukážkou ale hlavne vyhodnotenie natrénovaných modelov a výber výsledného modelu, ktorý bude použitý v rámci mobilnej aplikácie.

Nasleduje popis zdrojového kódu samotnej aplikácie, ktorý je rozdelený do troch častí, podľa jednotlivých vrstiev architektúry aplikácie, pričom dôraz je kladený prevažne na implementáciu algoritmu pre analýzu série opakovaní v reálnom čase počas cvičenia a na implementáciu jednotlivých obrazoviek užívateľského rozhrania.

7.1 Tréning detektoru

Vzhľadom na to, že detekcia váhových kotúčov v posilovni nie je úplne bežná úloha, bolo potrebné natrénovať vlastný detektor, vhodný pre inferenciu v reálnom čase na mobilných zariadeniach. Dnes už existuje pomerne veľké množstvo riešení a frameworkov, zaoberajúcich sa integráciou modelov strojového učenia do mobilných aplikácií. V rámci implentácie bol využitý framework *TensorFlow Lite* [9], vyvíjaný spoločnosťou Google a to hlavne vďaka jeho popularite, jednoduchému nasadeniu, dobrej dokumentácií a kvôli ponúkaným predtrénovaným modelom a nástrojom na ich dodatočnú úpravu podľa vlastných potrieb.

Ako bolo spomínané v sekcii 5.3 medzi ponúkané modely patria modely architektúry EfficientDet-Lite, ktoré bolo potrebné upraviť na detekciu váhových kotúčov za pomoci nástroja TensorFlow Lite Model Maker [11].

Dátová sada

Na dotrénovanie vyššie spomenutých modelov postačia aj menšie dátové sady, pričom sa podarilo nájsť niekoľko takýchto dátových sád zameraných na detekciu činiek, či váhových kotúčov. Častým problémom však boli relatívne nepresné anotácie, ktoré presahovali výrazne za požadované objekty, či ďalšie nekonzistencie.

Nakoniec bola vybratá dátová sada [1], obsahujúca pomerne kvalitne anotované obrázky s rôznymi typmi závaží z rôznych prostredí. Celkovo táto sada obsahuje 444 obrázkov rozdelených do trénovacej množiny s 301 obrázkami, validačnej množiny s 84 obrázkami a testovacej sady s 59 obrázkami. Obrázky z trénovacej sady boli následne augmentované zmenou jasu, kontrastu, či zakrytím rôznych častí.

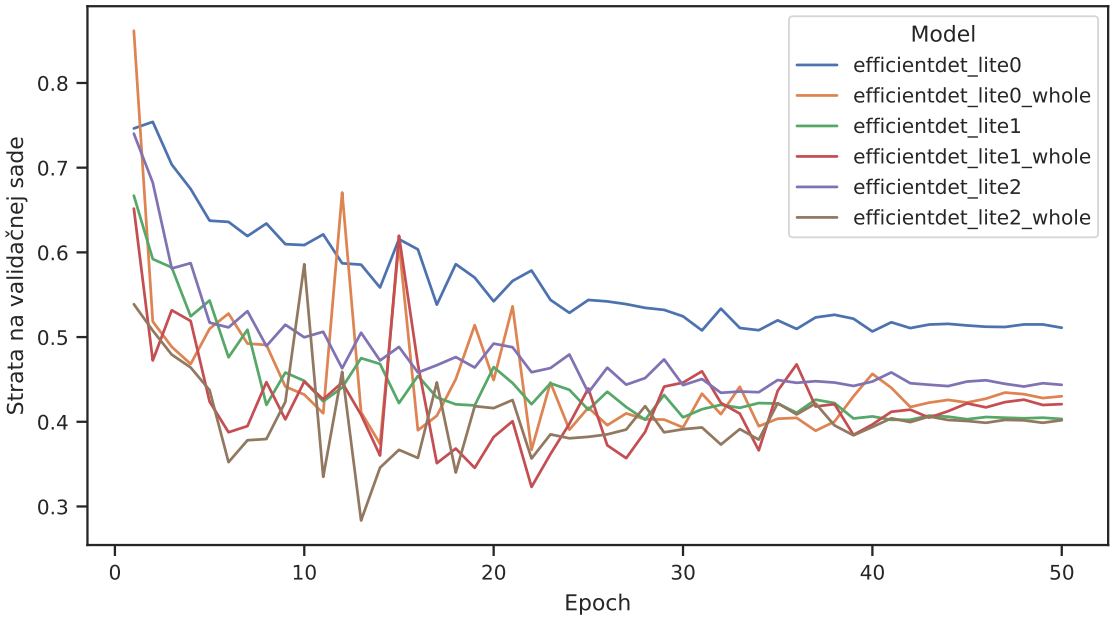
Spomínaná dátová sada bola doplnená niekoľkými vlastnými anotovanými obrázkami, celkovo spolu s augmentáciou, obsahuje trénovacia sada 936 snímkov, validačná sada obsahuje 88 obrázkov a testovacia sada 61.

Obr. 7.1: Ukážka niekoľkých obrázkov z dátovej sady [1] spolu s anotáciami.

Výsledné modely

Na vyššie uvedenej dátovej sade bolo natrénovaných niekoľko modelov architektúry EfficientDet-Lite0 až EfficientDet-Lite2. Väčšia architektúry už neboli uvažované, keďže výsledky dosiahnuté na danej dátovej sade s využitím väčších modelov boli skoro rovnaké zatiaľ, čo počty aritmetických operácii a doba inferencie sa zväčšovala násobne, viď namerané časy inferencie v tab. 5.1, či odhadované počty operácií v tab. 7.1. Priebeh trénovania jednotlivých modelov je ukázaný na obr. 7.2.

Skripty využité na tréning a vyhodnotenie modelov, spolu s dátovou sadou a všetkým potrebným k ich spusteniu sú dostupné v prílohe, či na verejne dostupnom GitHub repozitáry[1].



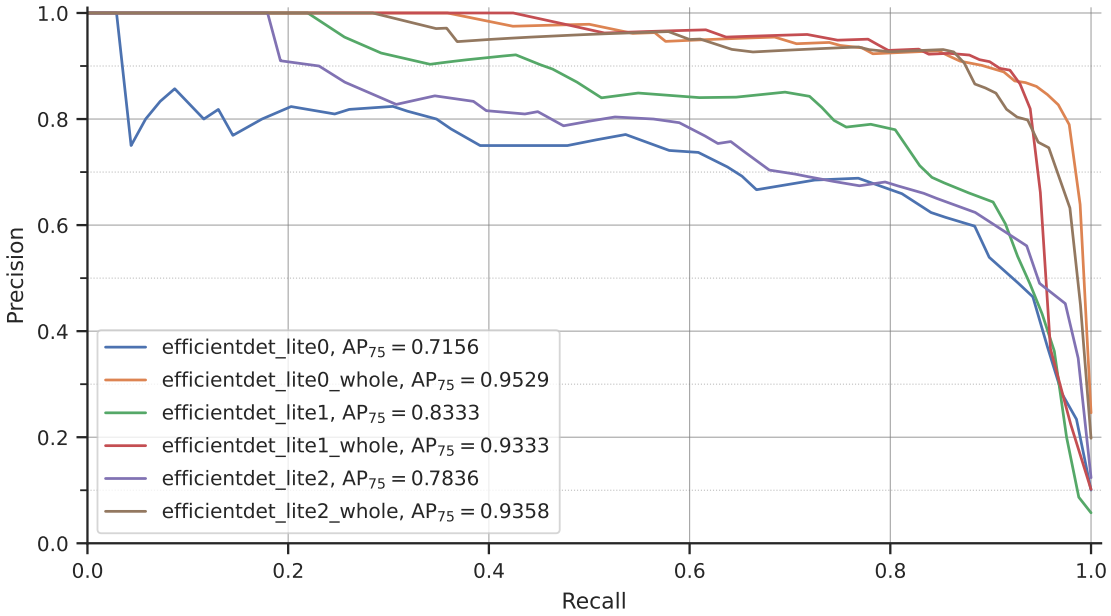
Obr. 7.2: Vývin hodnoty stratovej funkcie, zahŕňajúcej lokalizačnú aj klasifikačnú chybu, v čase pre jednotlivé modely. Modely s názvami končiacimi príponou `_whole` boli dotrénované celé, zatiaľ čo u ostatných modelov boli dotrénované len koncové vrstvy.

Pre porovnanie modelov boli na testovacej dátovej sade vyhodnotené *Precision-Recall* krivky, viď obr. 7.3, a ROC (*Receiver Operating Characteristic*) krivky, viď obr. 7.4, s *Intersection over Union* prahom, pre porovnanie detekcií s anotáciami nastaveným na hodnotu 0.75.

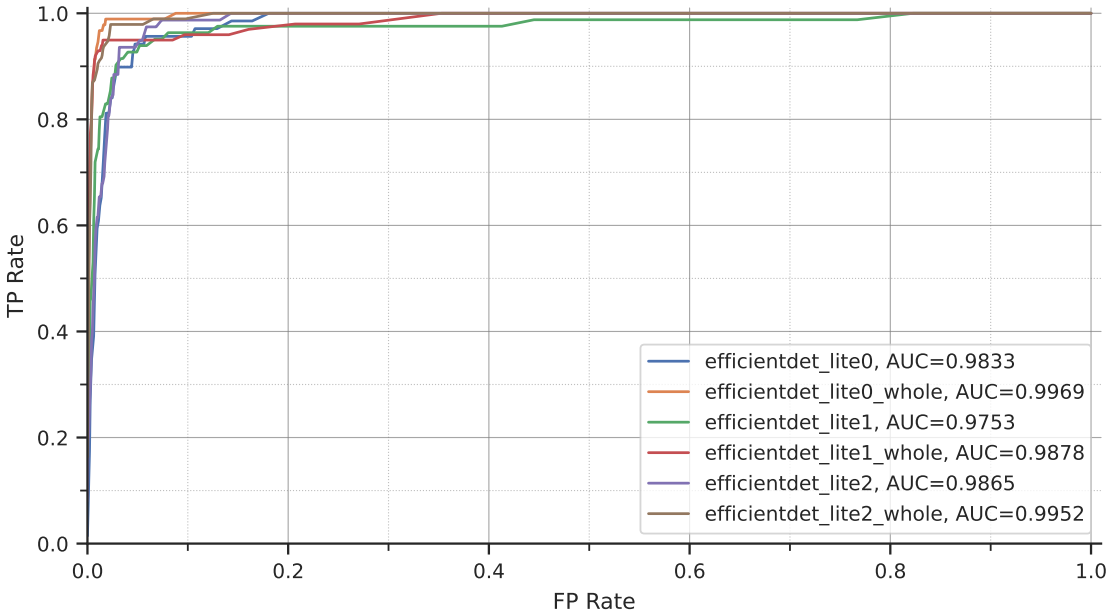
Z grafov je vidno, že trénovanie celých modelov výrazne napomohlo ich presnosti, oproti trénovaniu len posledných vrstiev. Takto natrénované modely EfficientDetLite-0 až EfficientDetLite-2 dosahujú na testovacej sade veľmi podobných výsledkov, dokonca najmenší model EfficientDetLite-0 sa javí byť o čosi lepší. Taktiež ak porovnávame odhadnuté počty operácií nutných pre inferenciu u jednotlivých modelov v tab. 7.1, vidíme, že najmenší model EfficientDet-Lite0 potrebuje na inferenciu zhruba dvakrát menej operácií ako model EfficientDet-Lite1, či štyrikrát menej ako model EfficientDet-Lite2.

Na základe vyššie uvedených dôvodov bol teda vybraný a využitý v rámci výslednej aplikácie práve model EfficientDet-Lite0.

[1] GitHub repozitár



Obr. 7.3: Krivka *Precision-Recall* s IoU prahom nastaveným na 0.75, porovnávajúca jednotlivé modely. Ku každému modelu je uvedená metrika *Average Precision*, ktorej hodnoty chceme maximalizovať. Krivka je tým lepšia, čím je vyššie a posunúta doprava.



Obr. 7.4: ROC krivky pre jednotlivé modely, spolu s metrikou AUC (Area Under the Curve), vyjadrujúcej obsah pod jednotlivými krivkami, pričom väčšie hodnoty metriky AUC a krivky posunuté hore a doľava sú lepšie. IoU prah bol opäť nastavený na hodnotu 0.75.

Tabuľka 7.1: Odhadované počty aritmetických operácií, či operácií MAC (*Multiply–Accumulate*), potrebných na inferenciu v prípade jednotlivých kvantizovaných modelov a ich veľkosti, čo sa týka priestoru zabraného na disku.

Model	Aritmetické op. $\times 10^9$	MAC op. $\times 10^9$	Veľkosť na disku [MB]
EfficientDet-Lite0	1.752	0.876	4.3
EfficientDet-Lite1	3.547	1.773	5.7
EfficientDet-Lite2	6.066	3.033	7.1

7.2 Členenie zdrojového kódu aplikácie

Architektúra zdrojového kódu aplikácie je primárne členená do troch vrstiev, spomínaných v kap. 4.4. Balíček `com.xkosin09.velocityTracker.ui` odpovedá prezentačnej vrstve, `com.xkosin09.velocityTracker.domain` odpovedá doménovej vrstve a balíček `com.xkosin09.velocityTracker.data` obsahuje triedy implementujúce dátovú vrstvu, pričom detaily implementácie jednotlivých vrstiev sú uvedené v následujúcich kapitolách.

Mimo spomínané vrstvy obsahuje zdrojový kód taktiež balíček `res`, obsahujúci rôzne dáta akými sú napr. reťazové konštanty zobrazované v rámci aplikácie v súbore `strings.xml`, či iné konštanty vyjadrujúce veľkosti jednotlivých prvkov užívateľského rozhrania v súbore `dimens.xml` a pod. Ďalej sú tu obsiahnuté potrebné ikony, fonty, či konštanty vyjadrujúce niektoré konfiguračné parametre TensorFlow Lite modelu.

Využívaný natrénovaný model EfficientDet-Lite0 je dodávaný spolu s aplikáciou v balíčku `assets`.

7.3 Implementácia dátovej vrstvy

Dátová vrstva je implementovaná pomerne štandardným spôsobom pomocou knižnice *Room*. V balíčku `com.xkosin09.velocityTracker.data` nájdeme implementáciu jednotlivých entít a vzťahov medzi nimi, definovaných v kap. 4.3, pričom využívané *SQLite* dotazy implementujú odpovedajúce *data access* objekty (DAO).

Balíček taktiež obsahuje implementáciu *DataStore* repozitáru pre ukladanie užívateľských preferencií.

7.4 Implementácia doménovej vrstvy

Doménová vrstva v našom prípade obsahuje hlavne triedy implementujúce detekciu objektov, sledovanie detekcií a algoritmus analyzujúci danú sériu opakovaní určitého cviku. Mimo ne tu však nájdeme aj rôzne pomocné triedy, či metódy rozširujúce funkcionalitu tried, implementovaných externými knižnicami, využívanými v rámci aplikácie.

Detekcia objektov z kamery

Na prácu s kamerou mobilného zariadenia v rámci aplikácie bola využitá knižnica *CameraX*[2], poskytujúca API umožňujúce zobrazenie náhľadu kamery, tvorbu fotiek a videí, či vykonanie analýzy jednotlivých snímkov. Väčšinu funkcionality potom sprístupňuje `LifecycleCameraController`[3]. Inštancia tejto triedy je využitá v rámci obrazovky *Record* z prezentačnej vrstvy, viď výpis 7.1.

```
val analyzer = remember {
    WeightImageAnalyzer(
        // Inicializácia detekteru objektov
        detector = TFLiteWeightDetector(
            // ...
        ),
        // Instancia triedy SortTracker pre sledovanie objektov
        sortTracker = recordState.sortTracker,
        onObjectsDetected = { objects, timestamp ->
            // Aktuálizácia detekcií v SORT trackeri
            recordViewModel.updateDetections(
                // ...
            )
        },
        // ...
    )
}

val controller = remember {
    LifecycleCameraController(context).apply {
        // povolenie analýzy snímkov a nahrávania videa
        setEnabledUseCases(
            CameraController.IMAGE_ANALYSIS or
            CameraController.VIDEO_CAPTURE
        )
        // nastavenie vlastnej analýzy
        setImageAnalysisAnalyzer(
            ContextCompat.getMainExecutor(context),
            analyzer
        )
        // ...
    }
}
```

Výpis 7.1: Využitie objektu triedy `LifecycleCameraController` pre ovládanie kamery a analýzu snímkov v obrazovke *Record*.

[2] Android dokumentácia – CameraX
[3] Android dokumentácia – LifecycleCameraController

V rámci konfigurácie je povolené nahrávanie videa a analýza jednotlivých snímkov, pričom pre analýzu je využitý objekt triedy `WeightImageAnalyzer`, ktorá implementuje rozhranie `ImageAnalysis.Analyzer`, poskytované knižnicou *CameraX*. Vstupom konštruktoru triedy `WeightImageAnalyzer` je detektor typu `TFLiteWeightDetector`, objekt triedy `SortTracker` a funkcia `onObjectsDetected` volaná po úspešnej detekcii objektov.

Trieda `WeightImageAnalyzer` implementuje metódu `analyze`, volanú pre jednotlivé snímky z kamery. V rámci nej sa najprv vykoná krok predikcie následujúcej polohy objektov volaním metódy `predict` objektu triedy `SortTracker`. Analyzovaný snímok je potom škálovaný a konvertovaný do požadovaného formátu. Následne je volaná metóda `detect` objektu triedy `TFLiteWeightDetector`, ktorej výsledkom sú detekcie jednotlivých závaží v snímku a tie sú spolu s časovým razítkom snímku predané funkcii `onObjectsDetected`. Funkcia `onObjectsDetected`, mimo iné, volá metódu `update` na objekte triedy `SortTracker` a predáva jej zoznam nových detekcií, čím sa vykoná aktualizácia trackeru.

Samotný proces detekcie objektov v danom snímku implementuje trieda `TFLiteWeightDetector`, viď výpis 7.2, obsahujúca atribút `objectDetector` typu `ObjectDetector`[4], poskytovaného knižnicou TensorFlow lite. V rámci metódy `setupObjectDetector` je tento atribút inicializovaný a nakonfigurovaný tak, aby využíval vlastný natrénovaný model EfficientDet-Lite0. Počas konfigurácie je taktiež možné nastaviť akcelerátor, na ktorom bude inferencia prebiehať, či počet CPU vlákien využitých pre inferenciu. Po zbežnom otestovaní bola zvolená inferencia na CPU s využitím 4 jadier, ktorá ponúkala uspokojivý výkon v podobe 25-28 snímkov za sekundu naprieč rôznymi modernejšími mobilnými zariadeniami a v prípade menej výkonných mobilných zariadení sa javilo, že v rámci možností ponúka uspokojivé výsledky. Pred vydaním aplikácie by však bolo vhodné výkon otestovať formálnejšie, alebo umožniť užívateľom zvoliť si vhodný akcelerátor, či počet vlákien CPU.

Argumentami metódy `detect` je snímok správnej veľkosti a jeho rotácia. V rámci metódy je snímok ďalej pred-spracovaný a následne je vykonaná samotná inferencia detektoru, ktorej výsledkom je zoznam detekcií typu `Detection`[5], obsahujúcich pozíciu a veľkosť bounding boxu a detekčné skóre, pričom detekcie, ktorých skóre je menšie ako nastavený prah sú automaticky odstránené.

[4] TensorFlow Lite – ObjectDetector
[5] TensorFLow Lite – Detection

```
class TFLiteWeightDetector(
    private val threshold: Float = 0.5f,
    // ...
): WeightDetector {
    private var objectDetector: ObjectDetector? = null

    private fun setupObjectDetector() {
        // Inicializácia atribútu objectDetector s využitím
        // modelu EfficientDetLite-0 a detekčným prahom treshold
        // ...
    }

    // Detekcia objektov v snímku
    override fun detect(bitmap: Bitmap, rotation: Int): List<Detection> {
        if (objectDetector == null) {
            setupObjectDetector()
        }

        // Nastavenie správnej rotácie
        val imageProcessor = ImageProcessor.Builder()
            .add(Rot90Op(-rotation / 90))
            .build()

        val tensorImage = imageProcessor
            .process(TensorImage.fromBitmap(bitmap))

        val results = objectDetector?.detect(tensorImage)

        // Návrat detekcií, či prázdneho zoznamu
        return results?.toList() ?: emptyList()
    }
    //...
}
```

Výpis 7.2: Trieda `TFLiteWeightDetector` implementujúca detektor váhových kotúčov v snímku.

Sledovanie detekcií

Počas vývoja nebola nájdená, pre aplikáciu vhodná, implementácia trackeru v jazyku Kotlin, či Java a preto bolo potrebné previesť pôvodnú implementáciu[6] SORT [4] trackeru z jazyku Python do jazyka Kotlin. Výsledok je možné nájsť v balíčku `com.xkosin09.velocityTracker.domain.sortTracker`, pričom podstatná je primárne trieda `SortTracker` využívaná pre sledovanie detekcií.

Táto trieda implementuje metódy `predict` a `update` vykonávajúce kroky predikcie a aktualizácie trackeru, popísané v kap. 5.4. Výsledný zoznam sledovaných detekcií je možné zís-

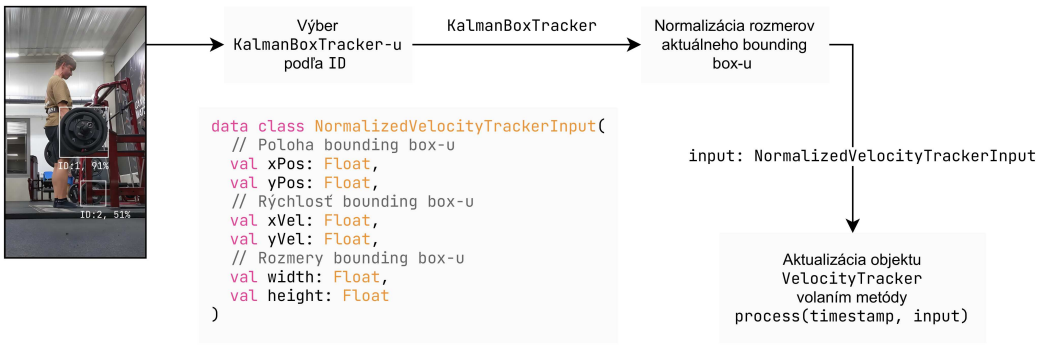
[6] GitHub repozitár – SORT

kať volaním metódy `getDetections`, ktorá vráti zoznam objektov typu `TrackedDetection`, ktoré okrem detekčného skóre a bounding boxu, obsahujú aj identifikáciu sledovanej trajektórie.

Spolu s trackerom boli implementované aj pomocné triedy `KalmanFilter`, či `KalmanBoxFilter`. Ako náhrada za knižnicu *NumPy*[7], umožňujúcu efektívne maticové počítanie v jazyku Python, bola zvolená knižnica *multik*[8] v jazyku Kotlin. Pre minimalizáciu ceny priradenia detekcií, viď druhý krok algoritmu v kap. 5.4, bola prebratá voľne dostupná implementácia algoritmu Kuhn–Munkres [25].

Analýza série cviku a výpočet metrík

Analýzu jednotlivých opakovaní v rámci cviku implementuje trieda `VelocityTracker` pomocou metódy `process`, ktorá na vstup dostáva informácie o novej polohe, rýchlosti a veľkosti vybraného bounding boxu.



Obr. 7.5: Analýza série opakovaní na základe detekcií váhových kotúčov vo videu. V prípade niekoľkých detekcií sa využije užívateľom zvolený bounding box. Dáta sú normalizované a nasleduje spracovanie triedou `VelocityTracker`.

Metóda `process`, viď výpis 7.3, na základe smeru aktuálnej rýchlosti a historických údajov určuje, v akej fáze cviku sa momentálne nachádzame. V prípade, že prebieha určitá fáza, zaznamenáva údaje o trajektórii, rozmeroch váhového kotúča a času detekcie. Tie sú potrebné pre výpočet rýchlostných metrík, či filtrovanie menších záchvevov činky, ktoré nie nutne odpovedajú opakovaniam.

[7] NumPy
[8] multik

```
fun process(time: Long, input: NormalizedVelocityTrackerInput) {
    // Rozmery bounding boxu sú filtrované pomocou kĺzavého priemeru
    val width = widthRunningAverage.update(input.width)
    val height = heightRunningAverage.update(input.height)

    // Uloženie trajektórie v prípade prebiehajúcej fáze opakovania
    if (currentPhase != null) {
        appendToBarPath(input.yPos, input.xPos,
            width, height, time)
    }

    // Kontrola ukončenia concentrickej fáze
    if (currentPhase == RepetitionPhase.CONCENTRIC) {
        if (input.yVel > 0) {
            positiveVelStreak++
            negativeVelStreak = 0

            if (positiveVelStreak >= velocityStreakThreshold) {
                endPhase()
            }
        } else {
            positiveVelStreak = 0
        }
    }

    // Kontrola začiatku koncentrickej fáze
    if (input.yVel < 0 && currentPhase == null) {
        negativeVelStreak++
        positiveVelStreak = 0

        // Obnovenie trajektórie
        if (negativeVelStreak == 1) {
            resetBarPath()
        } else {
            // Uloženie trajektórie aj pred samotným započatím fáze
            appendToBarPath(input.yPos,input.xPos,
                width, height, time)
        }

        if (negativeVelStreak >= velocityStreakThreshold) {
            startPhase(RepetitionPhase.CONCENTRIC)
        }
    }

    // obdobne kontrola začiatku a konca excentrickej fáze...
}
```

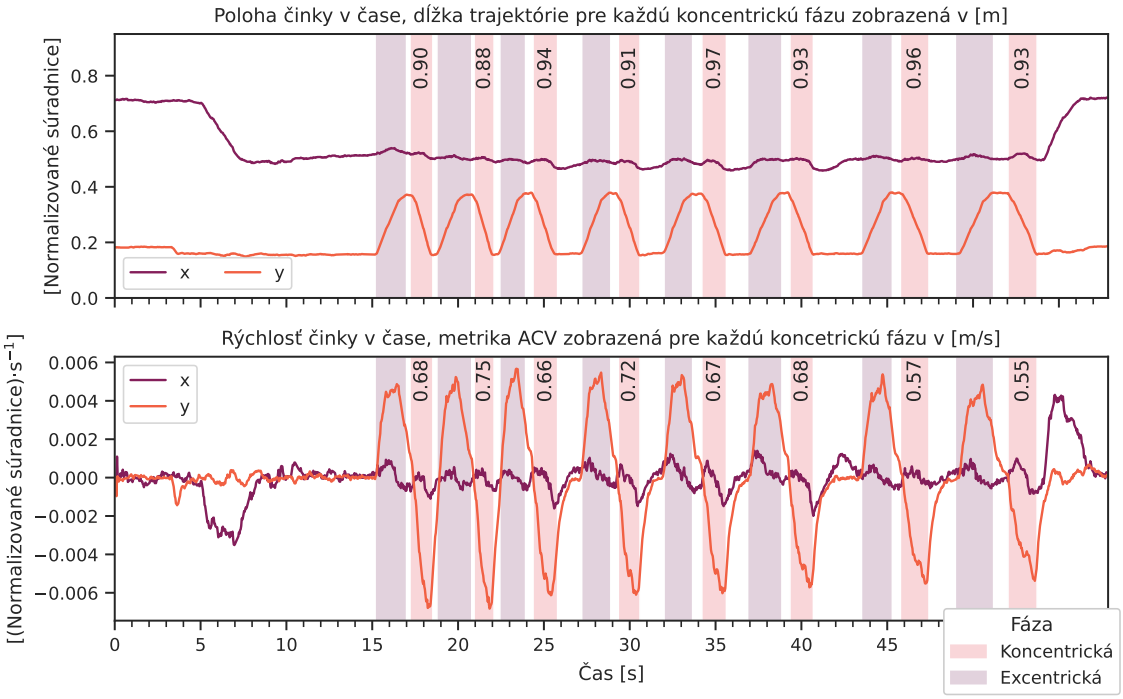
Výpis 7.3: Náčrt metódy pre detekciu a analýzu jednotlivých fáz opakovaní v rámci série, implementovú triedou `VelocityTracker`.

Výpočet a filtrovanie prebieha v implementovanej metóde endPhase. Táto metóda v prvom rade spresní počiatok a koniec danej koncentrickej alebo excentrickej, nájdením maxima, či minima trajektórie v ose Y. Na základe veľkosti absolútnej hodnoty rozdielu týchto hodnôt, v porovnaní s historicky maximálnym rozdielom sa určí, či sa jedná o skutočnú fázu alebo len šum v hodnotách. V prípade nájdenia nového maximálneho rozdielu sa spätne filtrujú už získané výsledky. Takýto spôsob filtrovania šumu väčšinou funguje potom, čo bolo vykonané a detekované prvé opakovanie v rámci série. V prípade, že užívateľ chce, aby mu bolo počas cvičenia signalizované dosiahnutie určitých rýchlostných metrík, je však potrebné taktiež vyfiltrovať šum pred vykonaním prvého opakovania, aby užívateľ nebol zbytočne vyrušovaný. Filtrovanie preto prebieha taktiež na základe užívateľom určeného koeficientu `minDistance`, čo napomáha odstrániť napr. počiatočné vybratie činky zo stojanu.

Nakoniec je v rámci metódy `endPhase` vytvorený objekt typu `VelocityTrackerResult`, viď výpis 7.4, a ten je pridaný do zoznamu detekovaných fáz. Dĺžka prejdenej trajektórie je kalibrovaná pomocou známej skutočnej veľkosti váhového kotúča, štandardne 45 cm, a detekovaných rozmerov kotúča, teda výšky a šírky bounding boxu, filtrovaných pomocou kĺzavého priemeru. V prípade, že sa skutočná veľkosť závažia líši od štandardnej, užívateľ ju môže upraviť.

```
data class VelocityTrackerResult(
    val timeStart: Long, // časové razítko začiatku fáze
    val timeEnd: Long, // časové razítko konca fáze
    val phase: RepetitionPhase, // typ fáze
    val distance: Float, // dĺžka prejdenej trajektórie
    val acv: Float? = null, // metrika Average Concentric Velocity
    val pv: Float? = null, // metrika Peak Velocity
    val mpv: Float? = null, // metrika Mean Propulsive Velocity
    internal val yDiff: Float, // abs. hodnota rozdielu medzi min. a max.
                               // hodnotou polohy v ose Y
)
```

Výpis 7.4: Reprezentácia jednej excentrickej, či koncentrickej fáze opakovania v rámci série. Atribúty `acv`, `pv` a `mpv`, predstávajúce rýchlostné metriky, má z definície zmysel počítať, len v prípade koncentrickej fáze. Atribút `yDiff` je použitý pre filtrovanie výsledkov popísané vyššie.



Obr. 7.6: Koncentrické a excentrické fázy opakovaní detekované vyššie uvedeným algoritmom spolu so spočítanými rýchlostnými metrikami. Vyhodnotenie pre viacero videí je možné nájsť v prílohe alebo na zverejnenom GitHub repozitári[9].

7.5 Implementácia prezentačnej vrstvy

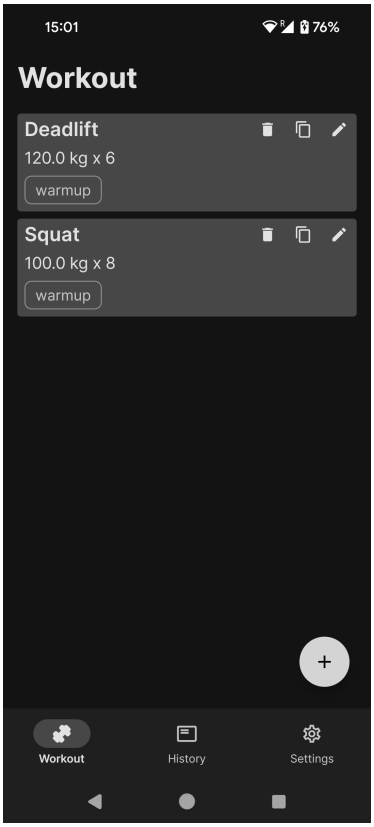
Implementáciu jednotlivých obrazoviek aplikácie s využitím knižnice *Jetpack Compose* nájdeme v balíčku `com.xkosin09.velocityTracker.ui`, spolu s rôznymi pomocnými funkciami, či komponentami využívanými v rámci niekoľkých obrazoviek, ako napr. spodný navigačný panel, či vrchný panel, náhľady pre zaznamenané série, modal pre úpravu série alebo modal pre nahranie novej série.

Na najvyššej úrovni sa nachádza komponenta `AppScreen`, ktorá zabezpečuje zobrazenie vrchného, či spodného navigačného panelu v rámci obrazoviek a následne vykresľuje komponentu `AppNavHost`, definujúcu navigačný graf medzi jednotlivými obrazovkami aplikácie a vykresľujúcu komponenty, ktoré ich reprezentujú.

Obrazovka *Workout*

Obrazovku *Workout* implementuje komponenta `WorkoutScreen`. Náhľady jednotlivých sérii vykresľuje komponenta `LazySetList` s využitím dostupnej komponenty `LazyColumn`[10], ktorá postupne vykresľuje len viditeľné položky zoznamu, čo v prípade veľkého množstva dát napomáha k lepšiemu výkonu aplikácie. Samotné náhľady jednotlivých sérii reprezentuje komponenta `SetPreviewCard`.

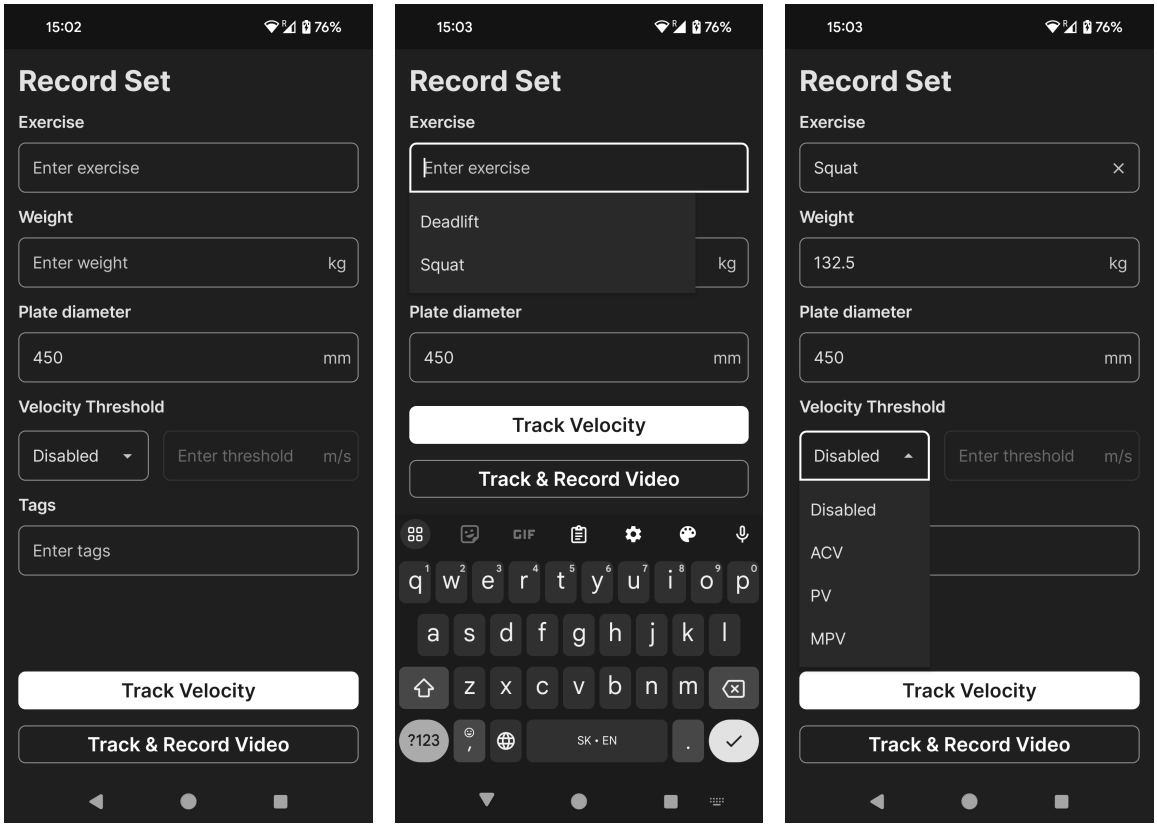
[9] GitHub repozitár
[10] Jetpack Compose – LazyColumn



Obr. 7.7: Obrazovka *Workout* vo výslednej aplikácii.

Vykreslenie modalov pre analýzu novej série, či upravenie existujúcej, zabezpečujú komponenty `RecordSetModal` a `EditSetModal` resp. V rámci nich je pri zadávaní cviku, či tagov, užívateľovi automaticky ponúknutý zoznam s nápovedou, čo umožňujú komponenty `ExerciseSearchBar` a `TagSearchBar`, využívajúce spoločnú komponentu `SearchBarWithHints`.

Pri nahrávaní novej série má užívateľ možnosť zmeniť štandardnú veľkosť kotúča, či zvoliť si metriku a hodnotu prahu. Ak hodnota vybranej metriky počas vykonávania série klesne pod daný prah, vyššie spomínaná trieda `VelocityTracker` upozorní užívateľa pomocou zvukového signálu. Po potvrdení zadaných údajov je užívateľ presmerovaný na obrazovku *Record*, popísanú v kap. 7.5.



Obr. 7.8: Ukážka modalu *Record Set* spolu s nápovedou pri doplňovaný údajov, čí výberom metriky pre signalizáciu počas cvičenia.

Obrazovka *Record*

Obrazovka *Record* je implementovaná komponentou `RecordScreen`. V rámci nej sú využité poskytované komponenty `LifecycleCameraController`[11], pre ovládanie kamery, viď kap. 7.4, a komponenta `AndroidView`[12], ktorá ponúka možnosť zobraziť objekty *View*[13], ešte neimplementované v novom framework-u *Jetpack Compose*, čo je práve prípad knižnice *CameraX*.

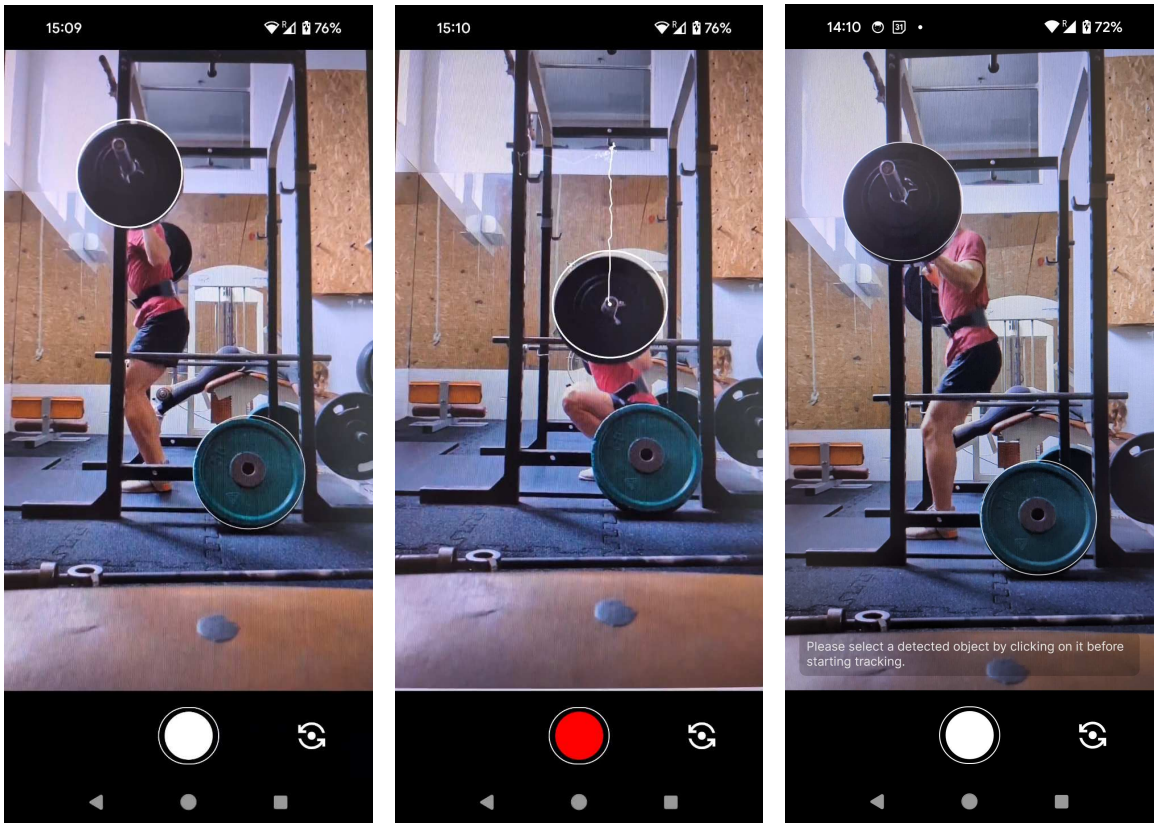
Vykreslenie trajektórie, detekcií a prípadnú interakciu s nimi implementuje komponenta `DrawDetections`. Ovládací panel kamery vykresľuje komponenta `CameraButtons`, ktorá taktiež zabezpečuje správne umiestnenie panela, či poradia tlačidiel, v prípade rotácie mobilného zariadenia.

Môže nastať prípad, kedy detekcia závaží nebude, pre zvolený detekčný prah úspešná, alebo bude detekovaných až niekoľko závaží a užívateľ bude musieť pred započatím analýzy vybrať vhodnú detekciu. V takýchto prípadoch je pomocou komponenty `InfoBox` užívateľovi zobrazená nápoveda.

Väčšina logiky je potom implementovaná v triede `RecordViewModel`, ktorá sa stará napr. o spustenie analýzy série so všetkými potrebnými parametrami a jej následné zastavanie,

[11] Android dokumentácia – LifecycleCameraController
[12] Android dokumentácia – AndroidView
[13] Android dokumentácia – View

vloženie výsledných dát do perzistentnej databázy, či uloženie nahrávaného video záznamu série.



Obr. 7.9: Ukážka obrazovky *Record*. Pred začiatkom analýzy série sú užívateľovi zobrazené všetky detekcie bez trajektórií, ďalej je zobrazovaný už len vybraný bounding box spolu s jeho trajektóriou. Posledný obrázok ukazuje zobrazenie nápovedy v prípade, že v snímku bolo nájdených viacero detekcií a užívateľ sa pokúsi zahájiť analýzu pred výberom jednej z nich.

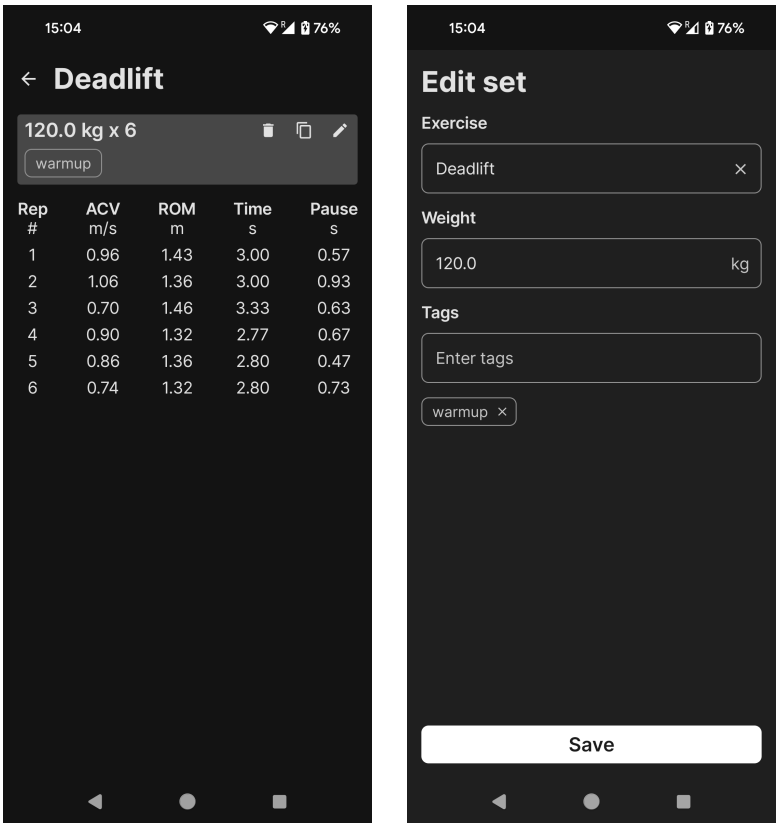
Obrazovka *Set Details*

Zobrazovanie nameraných dát pre danú sériu má na starosti obrazovka *Set Details*, implementovaná komponentou `SetDetailsScreen`. Náhľad série obsahujúci váhu, počet opakovaní a štítky tentokrát poskytuje komponenta `SetDetailsCard`. Obrazovka opäť podporuje otvorenie jednotlivých modalov pre upravenie zadaných údajov, či ich skopírovanie a nahratie novej série, poprípade vymazanie série.

Jednotlivé existujúce metriky sú reprezentované enumeračnou triedou `Metric`, ktorá definuje taktiež ich názov zobrazený v záhlaví tabuľky, či jednotky, v ktorých sú metriky uvádzané. Na triede `Rep` reprezentujúcej rovnomennú entitu v databáze, je neskôr definovaná metóda `getMetricValue(metric: Metric)`, ktorá na základe danej metriky a uložených údajov o opakovaní vráti naformátovaný reťazec obsahujúci hodnotu metriky zobrazenú v tabuľke.

Niektoré metriky, ako napr. ACV, MPV, či PV, sa získajú priamo z uloženého objektu `Rep`, iné, ako napr. metrika ROM *Range of Motion*, reprezentujúca prejdenú trajektóriu, či

dĺžka trvania jednotlivých fáz opakovania, alebo celého opakovania, je potrebné z uložených údajov odvodiť.



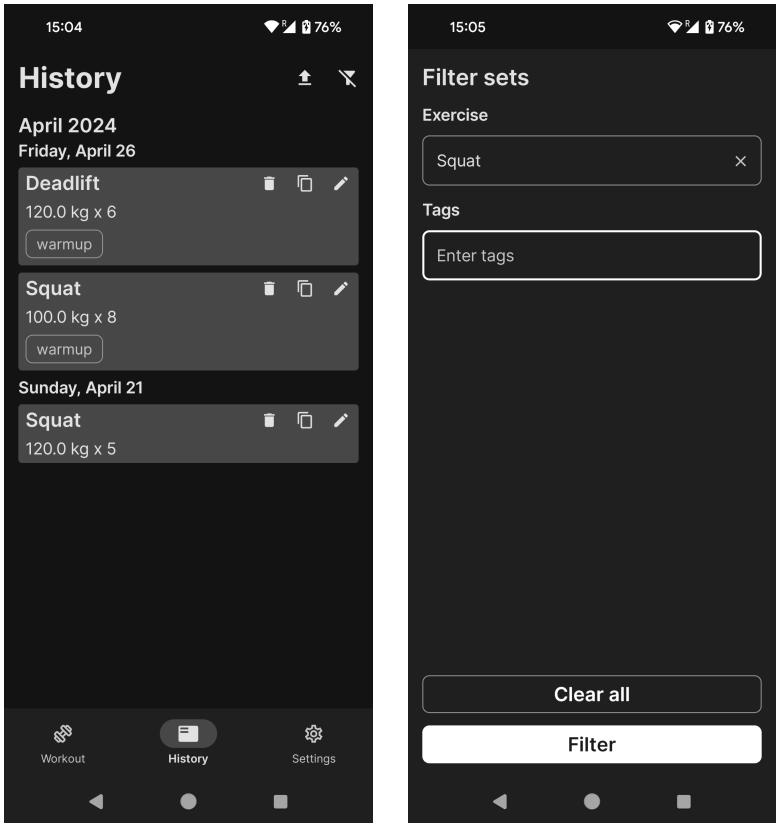
Obr. 7.10: Obrazovka *Set Details* vo výslednej aplikácii, spolu s otvoreným modalom *Edit Set*, slúžiacim pre úpravu zadaných údajov o sérii.

Modal *Edit Set* je implementovaný komponentou `EditSetModal`, ktorá zdieľa časť logiky s komponentou `RecordSetModal`, no zobrazuje len uložené údaje o sérii a neponúka možnosti nastavenia parametrov, akými sú veľkosť závažia, či prah vybranej metriky, keďže tieto údaje sú využité len počas analýzy série a ďalej nie sú ukladané.

Obrazovka *History*

Obrazovka *History* je implementovaná komponentou `HistoryScreen`. Z obr. 7.11 je vidieť, že obrazovka *History* je pomerne podobná obrazovke *Workout* a teda zdieľajú časti zdrojového kódu a to napr. komponentu `SetDetailsPreview` pre náhľady jednotlivých sérií, komponenty pre jednotlivé modaly, atď.

Jednotlivé náhľady sérií sú oproti obrazovke *Workout* zoskupené po mesiacoch a dňoch. Obrazovka *History* naviac podporuje filtrovanie medzi jednotlivými sériami na základe cviku, či štítkov. Nastavenie filtrov umožňuje komponenta `FilterModal`, samotnú logiku vyhľadávania potom implementuje trieda `HistoryViewModel` pomocou dynamického zostavenia *SQLite* dotazu v metóde `buildFilterQuey`.



Obr. 7.11: Ukážka implementovanej obrazovky *History* a modalu *Filter Set*.

Trieda `HistoryViewModel` taktiež implementuje metódu `createSpreadsheet` a `saveSpreadsheet` slúžiace na export dát vo formáte `.xls`. Metóda `createSpreadsheet` vytvorí v rámci súboru zvlášť hárok pre každý cvik a naplní ho údajmi z databázy, metóda `saveSpreadsheet` dočasne uloží vytvorený súbor v pamäti.

Komponenta `HistoryScreen` následne vytvorí `Intent` s akciou `ACTION_SEND`[14] a typom `application/vnd.ms-excel`, čo už umožní užívateľovi uložiť a zobraziť exportované dáta pomocou preferovanej aplikácie nainštalovanej v jeho mobilnom zariadení, ktorá daný typ súboru podporuje.

Obrazovka *Settings*

Obrazovku *Settings* implementuje komponenta `SettingsScreen`, ktorá pracuje prevažne s objektom triedy `UserPreferenceViewmodel`, ktorý poskytuje aktuálne hodnoty užívateľských preferencií a taktiež metódy pre ich zmenu. Objekt danej triedy je zdieľaný naprieč ostatnými obrazovkami, aby mali možnosť meniť vykreslený obsah, práve na základe týchto preferencií. Každé z nastavení je potom implementované zvlášť komponentou v balíčku `ui.settings`.

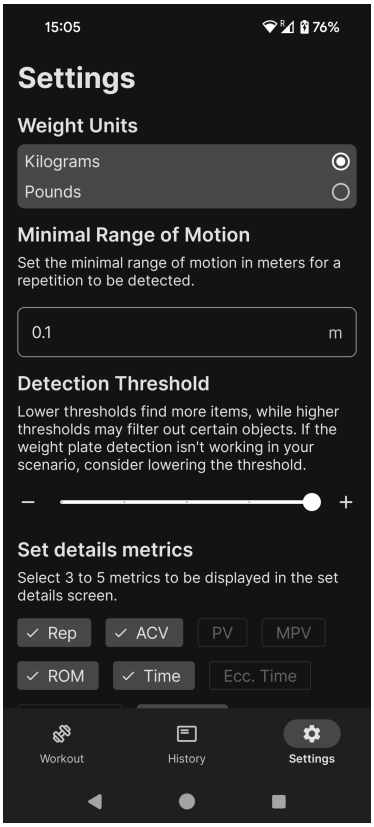
Výber váhových jednotiek ovplyvňuje vykreslenie obrazoviek *Workout*, *History*, *Set Details*, či jednotlivé modaly, ktoré musia pri zadávaní váhy užívateľovi zobraziť správnu jednotku.

[14] Android dokumentácia – Intent

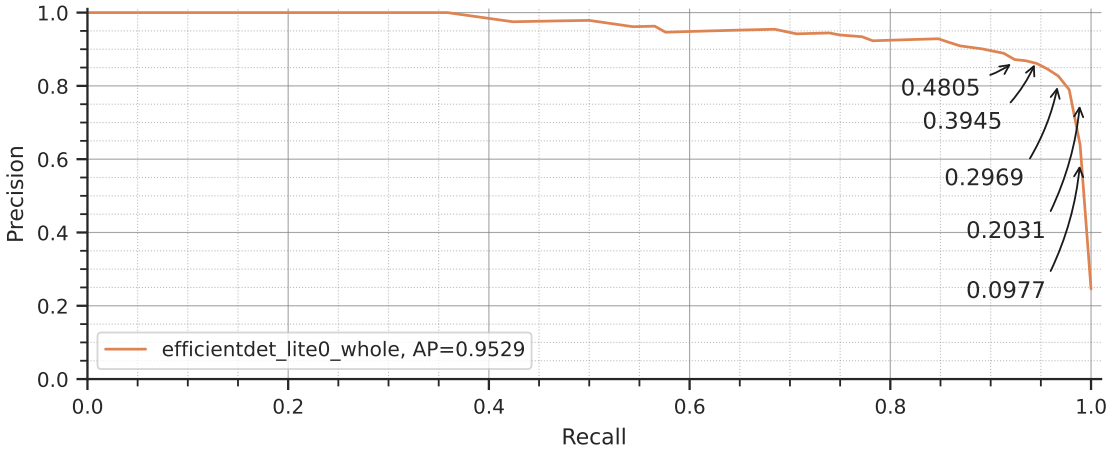
Hodnota nastavenia minimálnej prejdenej vzdialenosti je využitá ako argument `minDistance`, pri vytváraní objektu triedy `VelocityTracker`, slúžiaci na počiatočné filtrovanie šumu v zaznamenaných fázach, viď. kap. 7.4.

Prah detekcie je využitý pri vytváraní objektu triedy `TFLiteWeightDetector`, viď kap. 7.4. Užívateľ nemôže priamo zadať konkrétnu hodnotu, no je mu ukázaná škála prahov, pričom najnižší prah odpovedá hodnote 0.1, najvyšší hodnote 0.5 a kroky majú rozostup 0.1, viď obr. 7.13.

Užívateľ má taktiež možnosť zvoliť si až päť rôznych metrík, zobrazovaných v obrazovke *Set Details*.



Obr. 7.12: Obrazovka *Settings* vo výslednej aplikácii.



Obr. 7.13: Body na *Precision-Recall* krivke využívaného modelu EfficientDet-Lite0, približne odpovedajúce detekčným prahom $0.1, 0.2, 0.3, 0.4$ a 0.5, ktoré si užívateľ môže zvoliť v aplikácii.

Kapitola 8

Testovanie

Následujúca kapitola sa zaoberá vyhodnotením implementovaného riešenia, čo sa týka presnosti sledovania trajektórie činky. Najprv porovnáme získané výsledky so softvérom Kinovea, slúžiacim na anotáciu videí a následnú analýzu pohybov v čase. V tomto prípade vyhodnotenie prebieha na sade obsahujúcej 34 vlastných videí, natočených v rôznych posilňovniach, s rôznymi typmi váhových kotúčov.

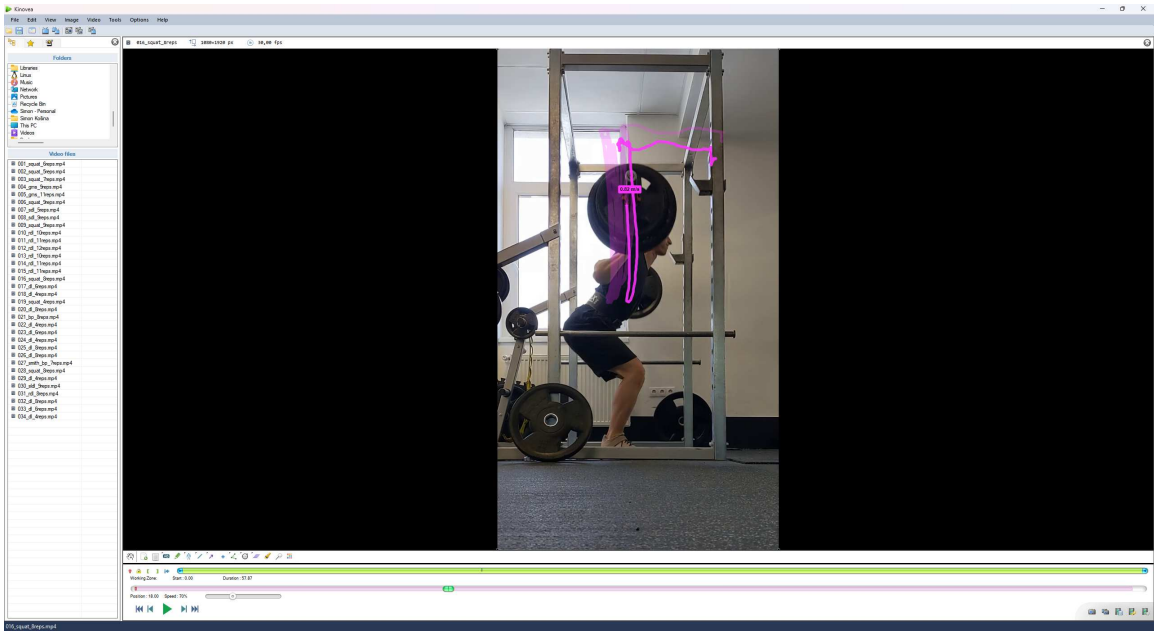
Ďalej sa nám vďaka spolupráci s Centrom športových aktivít VUT podarilo získať prístup k profesionálnemu systému pre sledovanie pohybu Qualysis a bolo tak natočených zopár videí, spolu s referenčnými trajektóriami nameranými práve týmto systémom, na ktorých bol následne taktiež vyhodnotené implementované riešenie.

8.1 Porovnanie so softvérom Kinovea

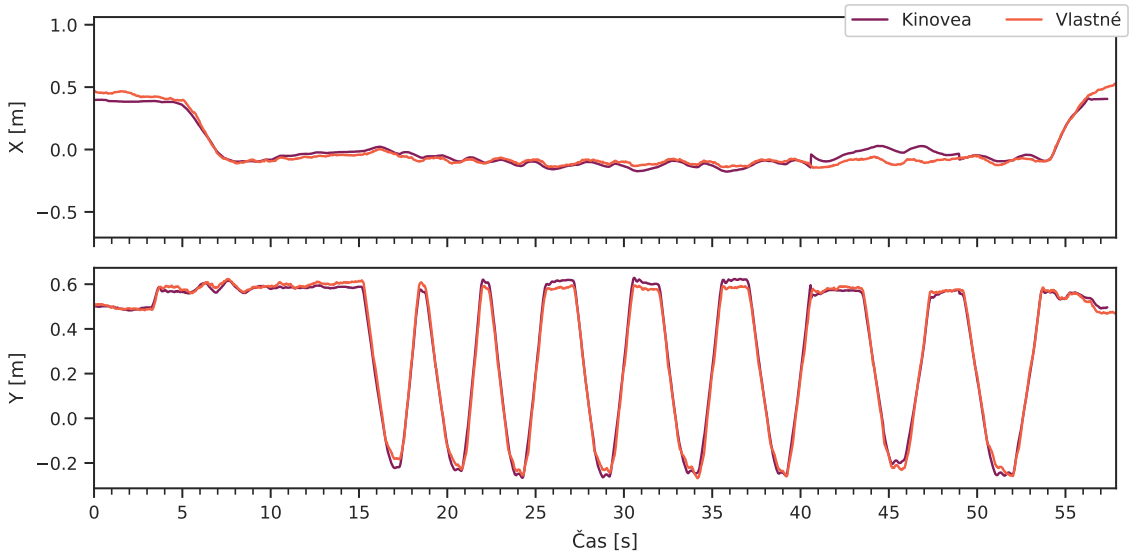
Kinovea [5] je nástroj, slúžiaci na anotáciu videí a následnú analýzu pohybov. V rámci kontextu tejto práce, je zaujímavá hlavne možnosť sledovania trajektórie manuálne zvoleného bodu vo videu. Softvér bol teda využitý na sledovanie trajektórie olympijskej tyče v jednotlivých videách z testovacej sady, pričom vzdialenosti boli kalibrované pomocou známej veľkosti váhových kotúčov v jednotlivých videách (45 cm). Sadu testovacích videí je možné opäť nájsť v prílohe, či v repozitári[1].

Zo softvéru je možné exportovať trajektóriu do CSV súboru pričom získame tri stĺpce s údajmi o čase [s], posune v smere osi X [cm] a posune v smere osi Y [cm]. Vyexportované trajektórie je potom možné porovnať s výstupom vyvinutého algoritmu. Pred samotným porovnaním bolo však potrebné zarovnať súradnicové systémy, keďže výstupom implementovaného riešenia sú súradnice polohy v rámci snímku, s počiatkom v ľavom hornom rohu, a výstupom softvéru Kinovea bol posun voči počiatku súradnicového systému definovaného pri kalibrácii. Vizuálne porovnanie trajektórií je možné vidieť napr. na obr. 8.2.

[1] GitHub repozitár



Obr. 8.1: Anotácia jednotlivých videí a sledovanie trajektórie činky v softvéri Kinovea.



Obr. 8.2: Porovnanie trajektórie činky z videa `016_squat_8reps`, teda polohy v ose X a v ose Y v závislosti na čase, získanej pomocou softvéru Kinovea a pomocou vlastného riešenia.

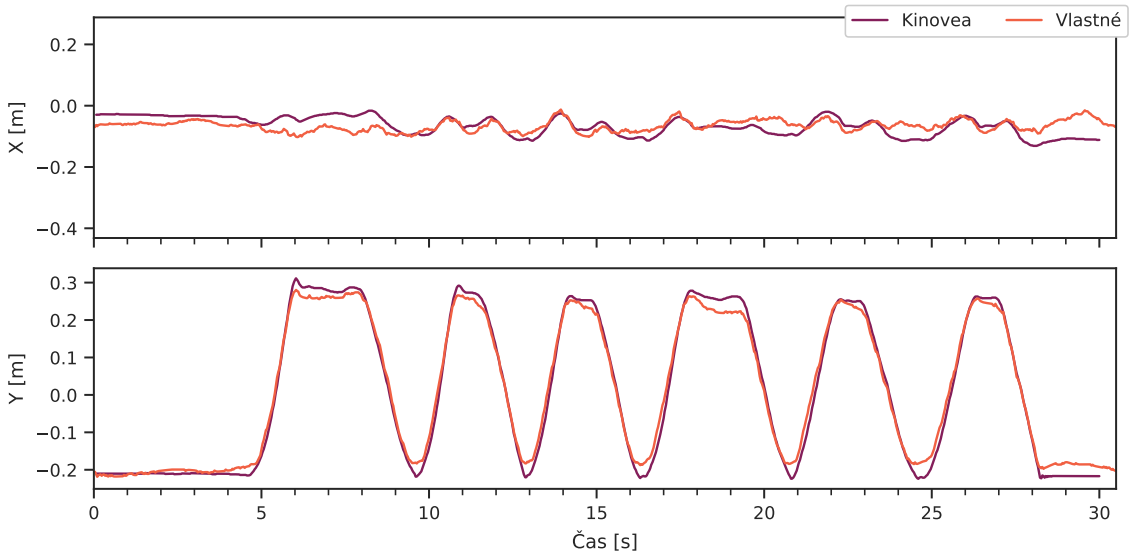
Okrem vykreslenia grafov porovnávajúcich trajektórie činiek v jednotlivých videách, boli taktiež spočítané štatistiky porovnávajúce výstupy z Kinovea a z vlastného riešenie, ktoré nájdeme v tab. 8.1.

Tabuľka 8.1: Porovnanie výstupu zo softvéru Kinovea a z vlastného riešenia. V stĺpcoch je uvedená stredná kvadratická chyba (*Mean Squared Error*, MSE) pre trajektórie v smere osi X a osi Y, následne je uvedený Pearsonov korelačný koeficient r, opäť spočítaný zvlášť pre jednotlivé smery.

Video	MSE_X	MSE_Y	r_X	r_X
001_squat_6reps	0.0021	0.0012	0.9767	0.9979
002_squat_5reps	0.0011	0.0008	0.9781	0.9973
003_squat_7reps	0.0046	0.0023	0.9160	0.9991
004_gms_9reps	0.0005	0.0003	0.9910	0.9958
005_gms_11reps	0.0004	0.0003	0.9697	0.9968
006_squat_9reps	0.0007	0.0014	0.9810	0.9984
007_sdl_5reps	0.0008	0.0019	0.6312	0.9935
008_sdl_9reps	0.0007	0.0007	0.5296	0.9948
010_rdl_10reps	0.0011	0.0011	0.9971	0.9918
011_rdl_11reps	*0.0112*	0.0012	0.9624	0.9894
012_rdl_12reps	*0.0133*	0.0005	0.9621	0.9976
013_rdl_10reps	0.0003	0.0022	0.9978	0.9790
014_rdl_11reps	0.0078	0.0010	0.9926	0.9960
015_rdl_11reps	*0.0259*	0.0022	0.9922	0.9615
016_squat_8reps	0.0016	0.0005	0.9766	0.9976
018_dl_4reps	0.0003	0.0004	0.8409	0.9957
019_squat_4reps	0.0039	0.0074	0.7054	0.9957
020_dl_8reps	0.0002	0.0017	0.8583	0.9937
021_bp_8reps	0.0013	0.0009	0.9875	0.9980
022_dl_4reps	0.0008	0.0005	0.3028	0.9938
023_dl_6reps	0.0010	0.0005	**0.2000**	0.9969
024_dl_4reps	0.0019	0.0018	0.6264	0.9857
025_dl_8reps	0.0001	0.0023	0.9517	0.9918
026_dl_8reps	0.0005	0.0062	0.8890	0.9282
027_smith_bp_7reps	0.0004	0.0019	0.8324	0.9967
028_squat_8reps	0.0012	0.0011	0.9905	0.9949
029_dl_4reps	0.0004	0.0002	0.8300	0.9982
030_sldl_9reps	0.0006	0.0016	0.9624	0.9884
031_rdl_8reps	0.0006	0.0021	0.9818	0.9866
032_dl_8reps	*0.0101*	0.0035	0.5008	0.9458
033_dl_6reps	0.0015	0.0031	0.3893	0.9648
034_dl_4reps	0.0004	0.0009	0.6751	0.9961

Pre jednotlivé videá bol taktiež vykonaný test nulovej hypotézy tvrdiacej, že dáta sú nekorelované a pochádzajú z normálneho rozdelenia pravdepodobnosti, pričom p-hodnoty trajektórie v smere osi Y boli takmer nulové v prípade všetkých videí a teda nulovú hypotézu môžeme zamietnuť. V prípade trajektórií v smere osi X už boli p-hodnoty o čosi väčšie, no stále výrazne pod hladinou významnosti 0.05. Najmenšia hodnota korelačného koeficientu r_X bola dosiahnutá pre video `023_dl_6reps`. Stredná kvadratická chyba však v tomto prípade nebola až tak vysoká, ako u iných videí a z grafu vidíme, že odchýlky sú len v

rádoch jednotiek centimetrov. Porovnanie trajektórií pre dané video je vykreslené na obr. 8.3 a výsledky v tab. 8.1 sú hrubo zvýraznené. V prípade pohybu v smere osi X však celkovo vidíme menšie hodnoty koeficientu r aj pre iné videá a je možné, že štatistická významnosť korelácie bola ovplyvnená pomerne veľkým počtom vzorkov.



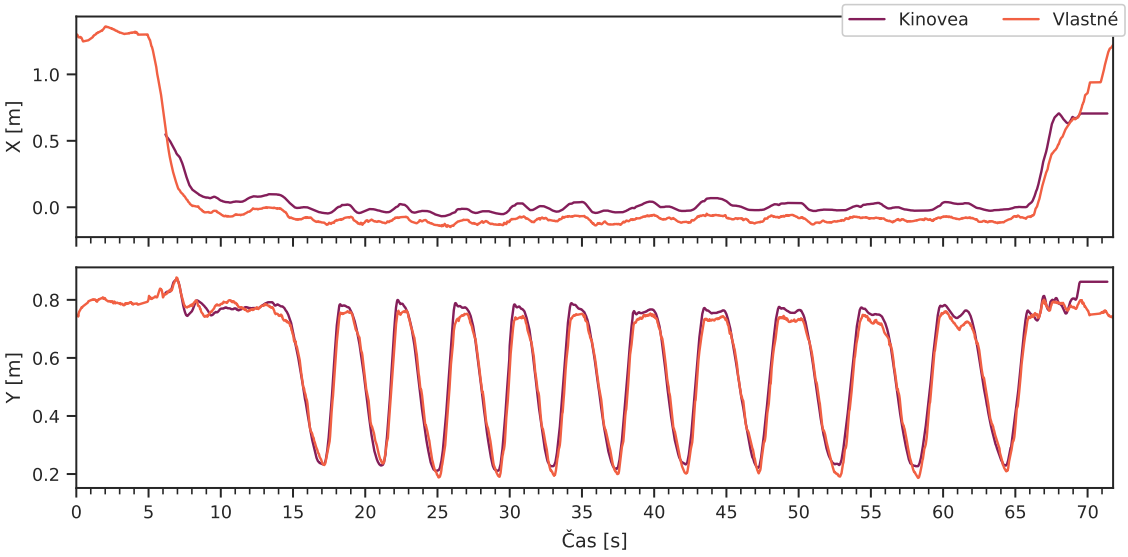
Obr. 8.3: Porovnanie výstupov pre video `023_dl_6reps` s najnižšou hodnotou $r_X = 0.2000$.

Ak sa pozrieme na stredné kvadratické chyby, vidíme, že v prípade osi Y sú tieto hodnoty pomerne nízke. O čosi horšie sú na tom hodnoty stredných kvadratických chýb v prípade porovnania trajektórií v smere osi X. Najhorších výsledkov dosiahli videá `011_rdl_11reps`, `012_rdl_12reps`, `015_rdl_11reps` a `032_dl_8reps`, pričom vizuálne porovnanie nájdeme na obr. 8.4, 8.5, 8.6 a 8.7 resp. Výsledky spomenutých videí v tab. 8.1 sú zvýraznené kurzívou.

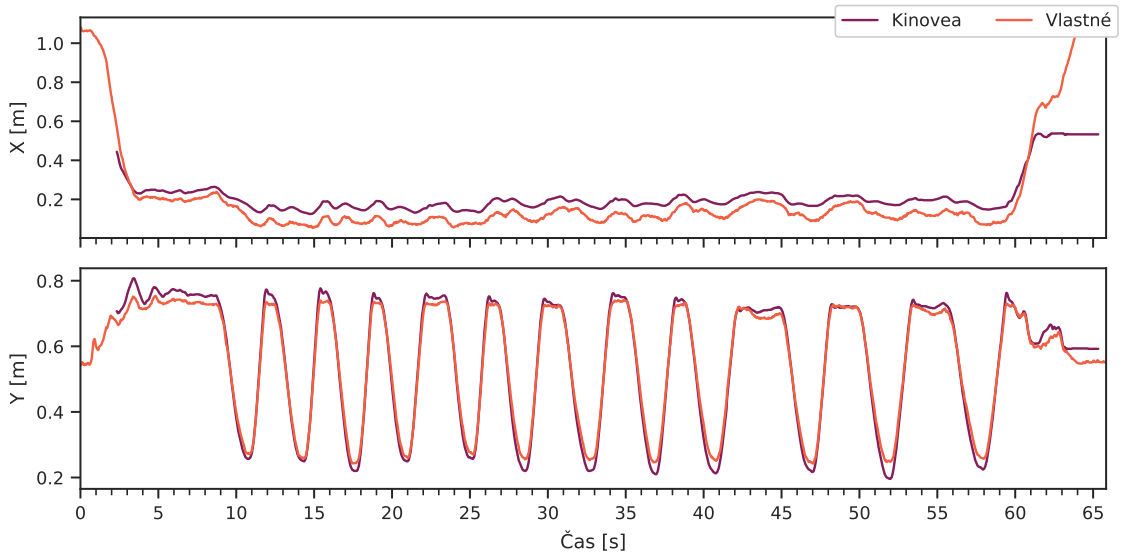
V prípade videí `011_rdl_11reps`, `012_rdl_12reps` vidíme, že na začiatku a konci videa je posun v smere osi X až prehnane veľký. V danú dobu však nie je vo videách vidno celý kotúč ale len jeho časť, čo znamená, že detekovaný bounding box je menší, a teda po kalibrácií pomocou veľkosti kotúča si algoritmus myslí, že prejdená trajektória je dlhšia ako v skutočnosti. Akonáhle je v snímkoch vidno celý kotúč, algoritmus sa správa podľa očakávaní.

Video `015_rdl_11reps` nie je natočené z ideálneho uhlu, teda kolmo na tyč, a na jeho začiatku a konci je kotúč trochu skreslený a teda detekovaný bounding box je opäť užší.

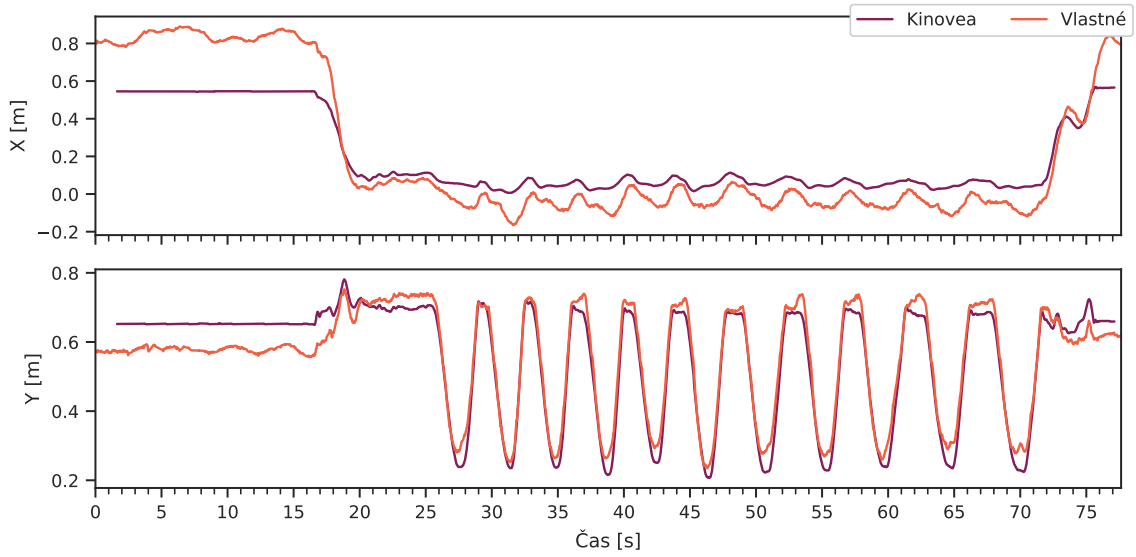
Jediné naozaj problematické je video `032_dl_8reps`, ktoré taktiež nie je natočené z ideálneho uhlu, no aj napriek tomu v ňom dochádza k príliš veľkému skresleniu posunu v smere osi X.



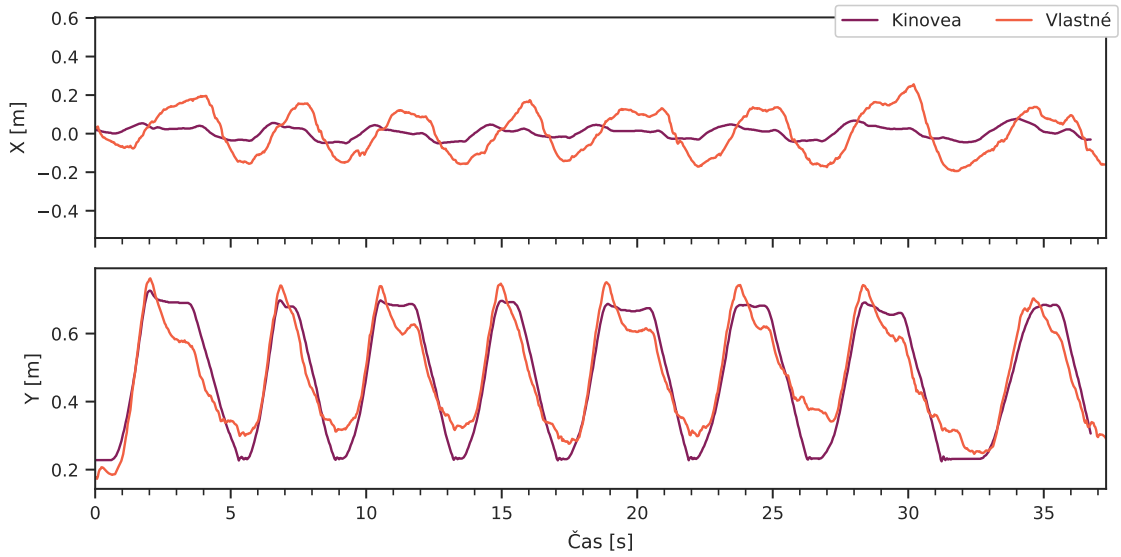
Obr. 8.4: Porovnanie výstupov pre video `011_rdl_11reps` s hodnotou $\mathrm{MSE}_X = 0.0112$.



Obr. 8.5: Porovnanie výstupov pre video `012_rdl_12reps` s hodnotou $\mathrm{MSE}_X = 0.0133$.



Obr. 8.6: Porovnanie výstupov pre video `015_rdl_11reps` s hodnotou $\mathrm{MSE}_X = 0.0259$.



Obr. 8.7: Porovnanie výstupov pre video `032_dl_8reps` s hodnotou $\mathrm{MSE}_X = 0.0101$.

Celkovo môžeme zhodnotiť, že implementované riešenie dospieva k podobným výsledkom ako softvér Kinovea, ak sú dodržané podmienky a kamera je natočená kolmo na tyč tak, aby nedochádzalo k skresleniu. Do budúcna by však bolo dobré vyladiť algoritmus tak, aby bol robustnejší a dokázal sa lepšie s takýmito situáciami vysporiadať, podobne, ako tomu je v prípade softvéru Kinovea. Taktiež by bolo vhodné viacej vyhladiť signál a parametre trackeru, keďže v porovnaní s Kinovea, sú získané výsledky o čosi viac zašumené. Môžeme si však všimnúť, že v tab. 8.1 chýbajú videá `009_squat_9reps`, `017_dl_6reps` a to z dôvodu, že v nich na chvíľu došlo k zakrytiu sledovaných objektov. Implementovaný vlastný algoritmus sa s týmto zakrytím pomerne dobre vysporiadal, no softvér Kinovea, s pôvodným nastavením, mal problém a sledovaný objekt bol stratený.

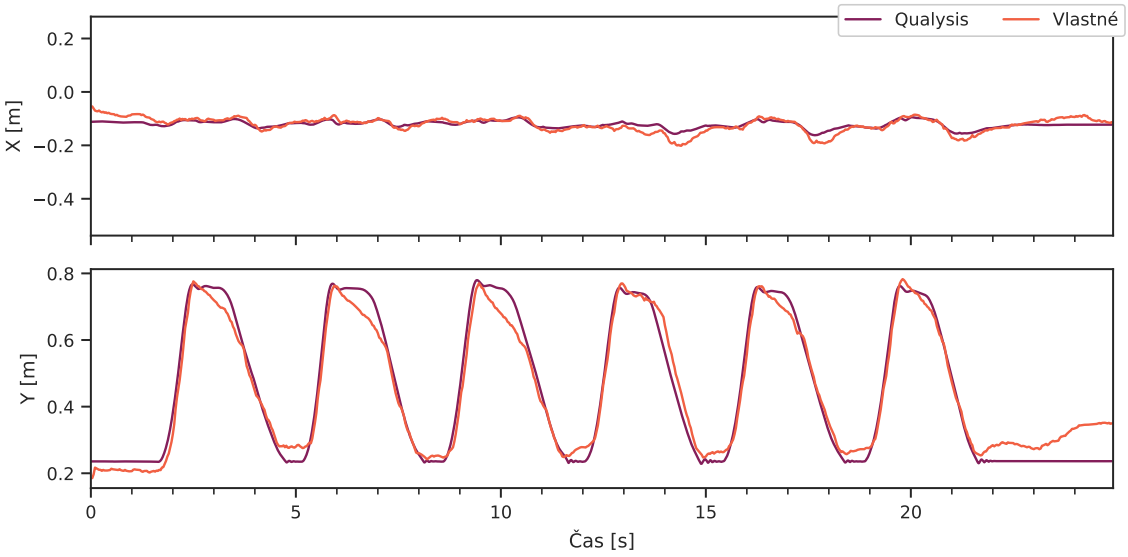
8.2 Porovnanie so systémom Qualysis

V spolupráci s Centrom športových aktivít VUT, bolo možné zhotoviť niekoľko kamerových záznamov, kde trajektória tyče bola zaznamenávaná využitím profesionálneho systému pre sledovanie pohybu Qualysis[2]. Zhotovené videá, spolu so zaznamenanými trajektóriami je opäť možné nájsť v prílohe.

Na dátach bola vykonaná rovnaká analýza ako pri porovnaní so softvérom Kinovea, výsledky je možné nájsť v tab. 8.2, či v nižšie vykreslených grafoch.

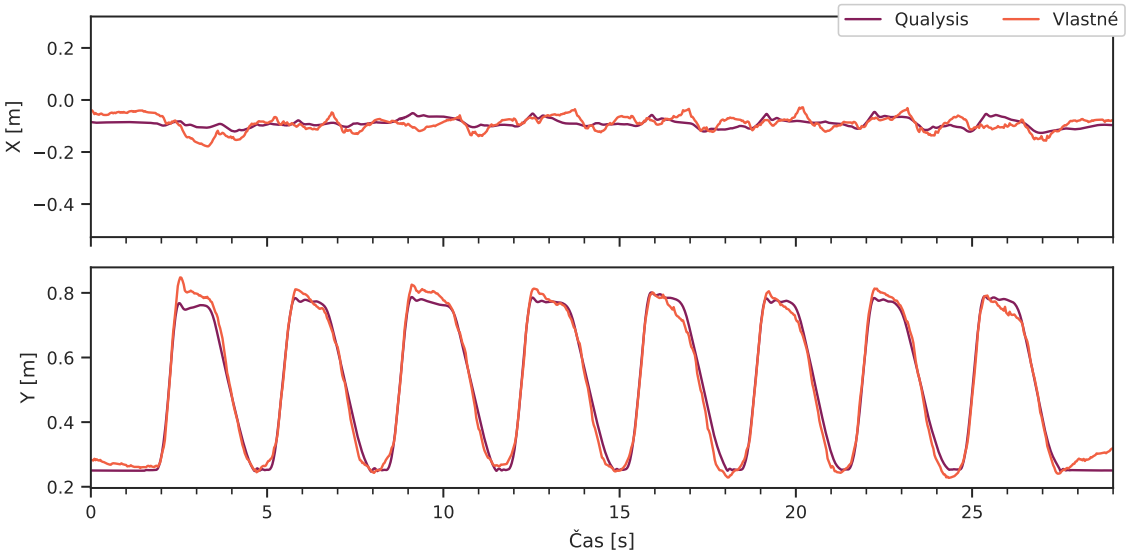
Tabuľka 8.2: Porovnanie dát nameraných systémom Qualysis a výstupu vlastného riešenia. V stĺpcoch je uvedená stredná kvadratická chyba (*Mean Squared Error*, MSE) pre trajektórie v smere osi X a osi Y, následne je uvedený Pearsonov korelačný koeficient r, opäť spočítaný zvlášť pre jednotlivé smery.

Video	MSE_x	MSE_y	r_x	r_y
`deadlift1_mobile_side_6reps`	0.0003	0.0018	0.8530	0.9832
`deadlift2_mobile_side_8reps`	0.0007	0.0007	0.4032	0.9930
`squat1_mobile_side_6reps`	0.0008	0.0018	0.5646	0.9903
`squat2_mobile_side_8reps`	0.0013	0.0015	0.0775	0.9928
`squat3_mobile_side_8reps`	0.0005	0.0019	0.5208	0.9907

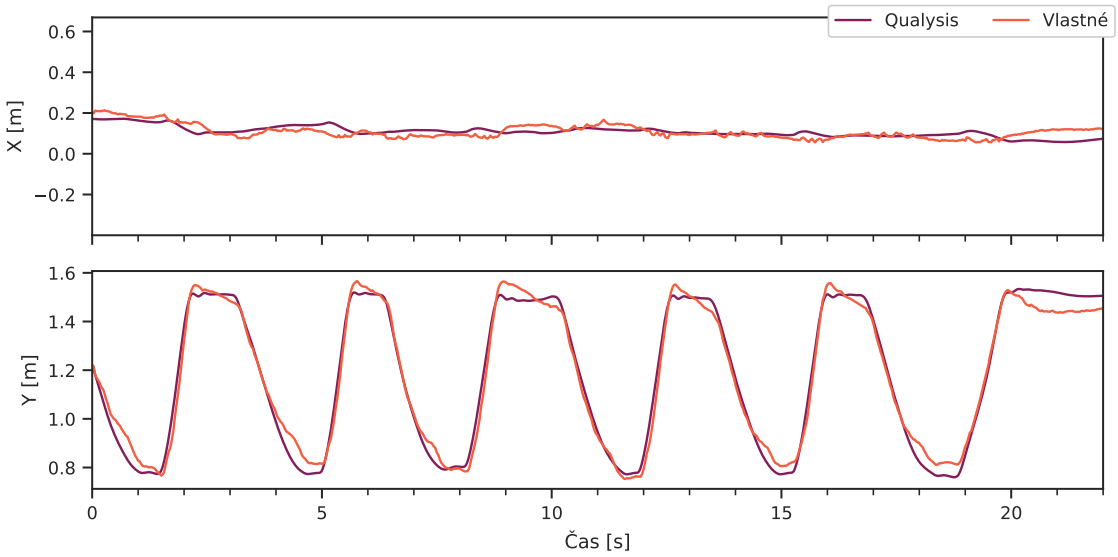


Obr. 8.8: Porovnanie výstupov pre video `deadlift1_mobile_side_6reps`.

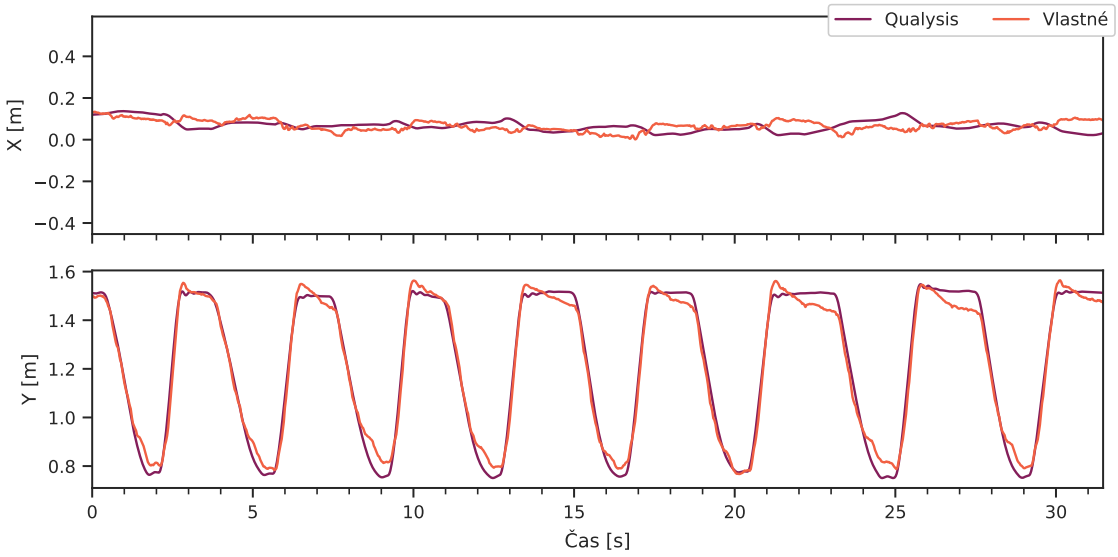
[2]Systém Qualysis



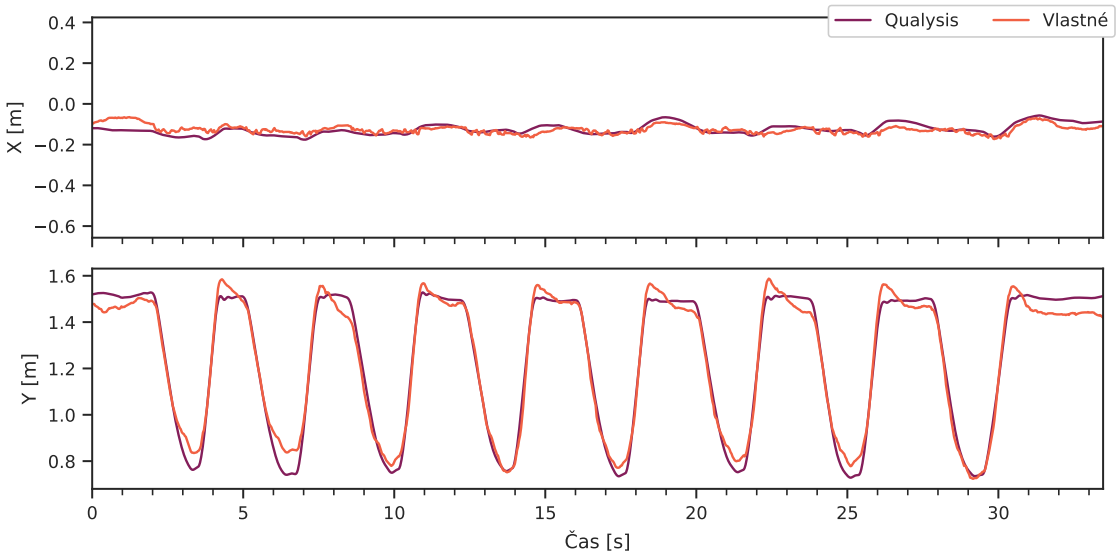
Obr. 8.9: Porovnanie výstupov pre video `deadlift2_mobile_side_8reps`.



Obr. 8.10: Porovnanie výstupov pre video `squat1_mobile_side_6reps`.



Obr. 8.11: Porovnanie výstupov pre video `squat2_mobile_side_8reps`.



Obr. 8.12: Porovnanie výstupov pre video `squat3_mobile_side_8reps`.

Z časových dôvodov bohužiaľ nebolo možné vytvoriť sadu obsahujúcu viacej videí, no výsledky porovnania sú podobné ako v prípade softvéru Kinovea. Môžeme zhodnotiť, že trajektórie namerané implementovanou aplikáciou sú podobné s referenčnými, no opäť by bolo vhodné ďalej experimentovať s nastaveniami, či využívanými metódami sledovania závaží a vyhladiť výstupný signál.

Kapitola 9

Záver

Cieľom práce bolo vytvoriť aplikáciu, schopnú poskytnúť užívateľom spätnú väzbu počas cvičenia pomocou nameraných rýchlostných metrík, ktorá by v budúcnosti mohla byť porovnateľná s komerčne dostupnými riešeniami, poskytovanými za mnohonásobne vyššie ceny.

V rámci textu práce boli najprv vysvetlené jednotlivé rýchlostné metriky a ich použitie pre plánovanie silového tréningu. Následne bol vykonaný prieskum existujúcich riešení, kde boli uvedené jednak riešenia využívajúce lineárne prevodníky, či iné bezdrôtové senzory, ale aj už existujúce aplikácie na podobnom princípe, ktorých kvalita, aspoň v prípade aplikácií dostupných pre zariadenia so systémom Android, nebola príliš vysoká. Ďalšia kapitola sa venovala základným princípom vývoja aplikácií pre mobilné zariadenia s operačným systémom Android a neskôr boli stručne opísané spôsoby detekcie a sledovania objektov, spolu s často využívanými architektúrami neurónových sietí.

Pred započatím realizácie bol vytvorený návrh aplikácie, čo sa týka približnej podoby užívateľského rozhrania a taktiež štruktúry databázy. Ďalej bola opísaná samotná realizácia aplikácie, spolu s trénovaním a vyhodnotením niekoľkých modelov využitých pre detekciu závaží. Zo špecifikácie aplikácie, uvedenej v kap. 3.4, spĺňa výsledná aplikácia základné požiadavky a taktiež bola naviac implementovaná možnosť exportu dát. Implementovaný algoritmus analýzy sérií sa vo väčšine prípadov dokáže vysporiadať s pohybom činky pred započatím samotnej série opakovaní a taktiež poskytuje užívateľovi spätnú väzbu v prípade prekročenia nastaveného prahu pre vybranú rýchlostnú metriku. Do budúcna je možné implementovať spomínané rozšírenia týkajúce sa predikcie počtu opakovaní v rezerve, či 1RM na základe historických dát, alebo analýzu rôznych trendov, pričom aktuálne si môžu užívatelia vykonať analýzu sami, na základe exportovaných dát, alebo pridať možnosť spätnej analýzy už nahratých videí.

Ďalej je v pláne zverejniť aplikáciu v službe Google Play. Síce bola aplikácia priebežne testovaná na niekoľkých mobilných zariadeniach, pred jej zverejnením je potrebné vykonať dôraznejšie testovanie, čo sa týka funkcionality na rôznych verziách systému Android a mobilných zariadeniach a taktiež testovanie výkonu aplikácie počas analýzy jednotlivých sérií. Okrem kvantitatívneho testovania, pripadá v úvahu získať spätnú väzbu na aplikáciu od užívateľov, ohľadom ich skúsenosti s jej používaním počas cvičenia a tú následne zapracovať do aplikácie.

Na základe porovnania so softvérom Kinovea a systémom Qualysis, môžeme zhodnotiť, že by bolo vhodné ďalej vyladiť parametre využitého trackeru a získať tak hladší priebeh signálu polohy, či experimentovať s využitím iných SOT, či MOT riešení namiesto algoritmu SORT a pokúsiť sa o robustnejšie sledovanie trajektórie činky, ktoré sa vysporiada aj s jej reidentifikáciou, v prípade, že sa ju po určitú dobu nepodarí detekovať. V prípade de-

tekcie niekoľkých bounding boxov, by bolo taktiež možné zautomatizovať výber vhodného bounding boxu, napr. s využitím dĺžky prejdenej trajektórie.

Literatúra

[1] *Barbell Detector Dataset*. Roboflow, sep 2022 [cit. 2024-3-9]. Dostupné z: https://universe.roboflow.com/personal-yvwij/barbell-detector-svmfh.

[2] Banyard, H. G., Nosaka, K., Vernon, A. D. a Haff, G. G. The reliability of individualized load-velocity profiles. *Int. J. Sports Physiol. Perform.* Júl 2018, zv. 13, č. 6, s. 763–769.

[3] Beck, B. R., Daly, R. M., Singh, M. A. F. a Taaffe, D. R. Exercise and Sports Science Australia (ESSA) position statement on exercise prescription for the prevention and management of osteoporosis. *J. Sci. Med. Sport*. Elsevier BV. Máj 2017, zv. 20, č. 5, s. 438–445.

[4] Bewley, A., Ge, Z., Ott, L., Ramos, F. a Upcroft, B. Simple online and realtime tracking. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, s. 3464–3468. DOI: 10.1109/ICIP.2016.7533003.

[5] Charmant, J. *Kinovea*. 2023 [cit. 2024-4-14]. Dostupné z: https://www.kinovea.org.

[6] García Ramos, A., Pestaña Melero, F. L., Pérez Castilla, A., Rojas, F. J. a Gregory Haff, G. Mean velocity vs. Mean propulsive velocity vs. Peak velocity: Which variable determines bench press relative load with higher reliability? *J. Strength Cond. Res.* Máj 2018, zv. 32, č. 5, s. 1273–1279.

[7] Girshick, R. Fast R-CNN. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, s. 1440–1448. DOI: 10.1109/ICCV.2015.169.

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

[9] Google. *TensorFlow: TensorFlow Lite*. [cit. 2024-4-11]. Dostupné z: https://www.tensorflow.org/lite.

[10] Google. Jetpack Compose for Android Developers. *Android Developers Website* [online]. August 2022 [cit. 2023-12-31]. Dostupné z: https://developer.android.com/courses/jetpack-compose/course.

[11] Google. *TensorFlow: TensorFlow Lite Model Maker*. 26. mája 2022 [cit. 2024-4-11]. Dostupné z: https://www.tensorflow.org/lite/models/modify/model_maker.

[12] Google. Developer guides. *Android Developers Website* [online]. 5. júna 2023 [cit. 2024-1-16]. Dostupné z: https://developer.android.com/docs.

[13] Google. Compose Documentation. *Android Developers Website* [online]. 11. októbra 2023 [cit. 2023-12-31]. Dostupné z: https://developer.android.com/jetpack/compose/documentation.

[14] Google. Guide to app architecture. *Android Developers Website* [online]. 12. decembra 2023 [cit. 2023-12-31]. Dostupné z: https://developer.android.com/topic/architecture.

[15] Google. *TensorFlow: Object Detection with TensorFlow Lite Model Maker*. 15. mája 2023 [cit. 2024-4-11]. Dostupné z: https://www.tensorflow.org/lite/models/modify/model_maker/object_detection.

[16] Google. *Brain AutoML*. GitHub, apríl 2024 [cit. 2024-4-11]. Dostupné z: https://github.com/google/automl. Path: automl; efficientdet.

[17] Gordon, B. R., McDowell, C. P., Hallgren, M., Meyer, J. D., Lyons, M. et al. Association of efficacy of resistance exercise training with depressive symptoms: Meta-analysis and meta-regression analysis of randomized clinical trials. *JAMA Psychiatry*. Jún 2018, zv. 75, č. 6, s. 566–576.

[18] Gordon, B. R., McDowell, C. P., Lyons, M. a Herring, M. P. The effects of resistance exercise training on anxiety: A meta-analysis and meta-regression analysis of randomized controlled trials. *Sports Med.* Springer Nature. December 2017, zv. 47, č. 12, s. 2521–2532.

[19] He, K., Zhang, X., Ren, S. a Sun, J. Deep Residual Learning for Image Recognition. 2015. Dostupné z: https://doi.org/10.48550/arXiv.1512.03385.

[20] Helms, E. R., Storey, A., Cross, M. R., Brown, S. R., Lenetsky, S. et al. RPE and velocity relationships for the back squat, bench press, and deadlift in powerlifters. *J. Strength Cond. Res.* Ovid Technologies (Wolters Kluwer Health). Február 2017, zv. 31, č. 2, s. 292–297.

[21] Hickmott, L. *Systematically individualizing load prescription* [online]. 30. júna 2020 [cit. 2024-1-19]. Dostupné z: https://vitruve.fit/blog/load-autoregulation-vbt/.

[22] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W. et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. Dostupné z: https://doi.org/10.48550/arXiv.1704.04861.

[23] Hu, J., Shen, L. a Sun, G. Squeeze-and-Excitation Networks. *CoRR*. 2017, abs/1709.01507. Dostupné z: http://arxiv.org/abs/1709.01507.

[24] Karim, R. A compiled visualisation of the common convolutional neural networks. *Illustrated: 10 CNN Architectures*. 29. júla 2019 [cit. 2024-4-6]. Dostupné z: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d.

[25] Kevin L. Stern. *Software and Algorithms*. GitHub, 22. mája 2021 [cit. 2024-4-25]. Dostupné z: https://github.com/KevinStern/software-and-algorithms. Path: src; main; java; blogspot; software_and_algorithms; stern_library; optimization; HungarianAlgorithm.java.

[26] Krizhevsky, A., Sutskever, I. a Hinton, G. E. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*. 2012, zv. 25.

[27] Lauersen, J. B., Andersen, T. E. a Andersen, L. B. Strength training as superior, dose-dependent and safe prevention of acute and overuse sports injuries: a systematic review, qualitative analysis and meta-analysis. *Br. J. Sports Med.* BMJ. December 2018, zv. 52, č. 24, s. 1557–1563.

[28] Leite, T. B., Costa, P. B., Leite, R. D., Novaes, J. S., Fleck, S. J. et al. Effects of different number of sets of resistance training on flexibility. *Int. J. Exerc. Sci.* September 2017, zv. 10, č. 3, s. 354–364.

[29] Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B. et al. Feature pyramid networks for object detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, s. 2117–2125.

[30] Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P. et al. Microsoft coco: Common objects in context. In: Springer. *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*. 2014, s. 740–755.

[31] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E. et al. SSD: Single Shot MultiBox Detector. *CoRR*. 2015, abs/1512.02325. Dostupné z: http://arxiv.org/abs/1512.02325.

[32] Morán Navarro, R., Martínez Cava, A., Sánchez Medina, L., Mora Rodríguez, R., González Badillo, J. J. et al. Movement velocity as a measure of level of effort during resistance exercise. *J. Strength Cond. Res.* Ovid Technologies (Wolters Kluwer Health). Jún 2019, zv. 33, č. 6, s. 1496–1504.

[33] O'Shea, K. a Nash, R. *An Introduction to Convolutional Neural Networks*. 2015. Dostupné z: https://doi.org/10.48550/arXiv.1511.08458.

[34] Pal, S. K., Pramanik, A., Maiti, J. a Mitra, P. Deep learning in multi-object detection and tracking: state of the art. *Applied Intelligence*. Sep 2021, zv. 51, č. 9, s. 6400–6429. DOI: 10.1007/s10489-021-02293-7. ISSN 1573-7497. Dostupné z: https://doi.org/10.1007/s10489-021-02293-7.

[35] Pareja Blanco, F., Rodríguez Rosell, D., Sánchez Medina, L., Sanchis Moysi, J., Dorado, C. et al. Effects of velocity loss during resistance training on athletic performance, strength gains and muscle adaptations. *Scand. J. Med. Sci. Sports*. Júl 2017, zv. 27, č. 7, s. 724–735.

[36] Redmon, J., Divvala, S., Girshick, R. a Farhadi, A. You Only Look Once: Unified, Real-Time Object Detection. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, s. 779–788. DOI: 10.1109/CVPR.2016.91.

[37] Ren, S., He, K., Girshick, R. a Sun, J. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*. 2015, zv. 28.

[38] RepOne Strength. *RepOne* [online]. 2024 [cit. 2024-1-20]. Dostupné z: https://www.reponestrength.com/.

[39] Rodríguez Rosell, D., Yáñez García, J. M., Sánchez Medina, L., Mora Custodio, R. a González Badillo, J. J. Relationship between velocity loss and repetitions in reserve in the bench press and back squat exercises. *J. Strength Cond. Res.* Ovid Technologies (Wolters Kluwer Health). September 2020, zv. 34, č. 9, s. 2537–2547.

[40] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. a Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, s. 4510–4520.

[41] Sherrington, C., Fairhall, N. J., Wallbank, G. K., Tiedemann, A., Michaleff, Z. A. et al. Exercise for preventing falls in older people living in the community. *Cochrane Database Syst. Rev.* Wiley. Január 2019, zv. 1, č. 1, s. CD012424.

[42] Simonyan, K. a Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. Dostupné z: https://doi.org/10.48550/arXiv.1409.1556.

[43] Smyth, N. *Jetpack Compose 1.3 Essentials: Developing Android Apps with Jetpack Compose 1.3, Android Studio, and Kotlin*. Payload Media, apríl 2023. ISBN 9781951442644.

[44] Steele, J., Endres, A., Fisher, J., Gentil, P. a Giessing, J. Ability to predict repetitions to momentary failure is not perfectly accurate, though improves with resistance training experience. *PeerJ*. November 2017, zv. 5, s. e4105.

[45] Suchomel, T. J., Nimphius, S., Bellon, C. R. a Stone, M. H. The importance of muscular strength: Training considerations. *Sports Med.* Apríl 2018, zv. 48, č. 4, s. 765–785.

[46] Taber, C., Patterson, E., Shah, J., Francis, P. a Wager, J. Validity and Reliability of a Computer Vision System to Determine Bar Displacement and Velocity. *International Journal of Strength and Conditioning*. Nov. 2023, zv. 3, č. 1. DOI: 10.47206/ijsc.v3i1.263. Dostupné z: https://journal.iusca.org/index.php/Journal/article/view/263.

[47] Tan, M. a Le, Q. V. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020. Dostupné z: https://doi.org/10.48550/arXiv.1905.11946.

[48] Tan, M., Pang, R. a Le, Q. V. *EfficientDet: Scalable and Efficient Object Detection*. 2020. Dostupné z: https://doi.org/10.48550/arXiv.1911.09070.

[49] Valle, C. A Rapid Review of the FLEX Barbell Tracker. *SimpliFaster* [online]. 5. januára 2020 [cit. 2024-1-19]. Dostupné z: https://simplifaster.com/articles/flex-barbell-tracker-review/.

[50] Wang, C.-F. *A Basic Introduction to Separable Convolutions*. 14. augusta 2018 [cit. 2024-4-7]. Dostupné z: https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728.

[51] Xiao, Y., Tian, Z., Yu, J., Zhang, Y., Liu, S. et al. A review of object detection based on deep learning. *Multimedia Tools and Applications*. Sep 2020, zv. 79, č. 33, s. 23729–23791. DOI: 10.1007/s11042-020-08976-6. ISSN 1573-7721. Dostupné z: https://doi.org/10.1007/s11042-020-08976-6.

[52] Yoon, D. H., Lee, J.-Y. a Song, W. Effects of resistance exercise training on cognitive function and physical performance in cognitive frailty: A randomized controlled trial. *J. Nutr. Health Aging.* 2018, zv. 22, č. 8, s. 944–951.

[53] Zourdos, M. C., Klemp, A., Dolan, C., Quiles, J. M., Schau, K. A. et al. Novel resistance training-specific rating of perceived exertion scale measuring repetitions in reserve. *J. Strength Cond. Res.* Ovid Technologies (Wolters Kluwer Health). Január 2016, zv. 30, č. 1, s. 267–275.

Príloha A

Obsah pamäťového média

- `app.zip` – Zdrojový kód aplikácie pre vývojové prostredie *Android Studio*.
- `cesa.zip` – Dáta nazbierané systémom Qualysis v spolupráci s CESA VUT.
- `plagat.png` – Prezentačný plagát.
- `scripts.zip` – Zdrojový kód rôznych skriptov v jazyku Python, používaných na trénovanie modelov, ich vyhodnotenie a taktiež porovnanie so softvérom Kinovea a systémom Qualysis. V zložke `sets` nájdeme nazbierané videá, na ktorých bolo samotné porovnanie vykonané. Zložka `data` obsahuje dátovú sadu využitú na tréning detektoru.
- `thesis.zip` – Zdrojové súbory písomnej správy.
- `vbt.apk` – Balíček pre systém Android s implementovanou aplikáciou.
- `xkosin09_thesis.pdf` – Písomná správa.

Príloha B

Plagát

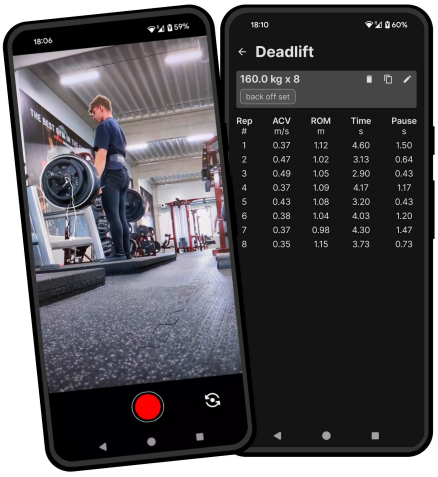
MOBILNÁ APLIKÁCIA PRE PODPORU TRÉNOVANIA SILOVÝCH ŠPORTOV

Silový tréning na základe rýchlosti pohybu

Mobilná aplikácia pre zariadenia s operačným systémom Android pre podporu silového tréningu na základe rýchlosti pohybu (*Velocity Based Training*).

Rýchlostné metriky pre odcvičené opakovania, ako napr. *Average Concetric Velocity*, poskytujú objektívnu spätnú väzbu o intenzite vykonaných sérií.

Aplikácia umožňuje pomocou metód počítačového videa sledovať trajektóriu činky, detekovať fázy jednotlivých opakovaní v rámci série a kalibráciou, pomocou známej veľkosti závažia, merať prejdenú vzdialenosť, či iné rýchlostné metriky.



Okamžitá spätná väzba počas cvičenia

Podpora nastavenia prahu vybranej rýchlostnej metriky pre spätnú väzbu v reálnom čase v podobe zvukového signálu, umožňujucu dodržanie požadovanej intenzity cvičenia.

Možnosť natáčania videozáznamov počas sledovania trajektórie činky pre spätnú analýzu techniky vykonávania cviku.

Implementovaných je hneď niekoľko rýchlostných, či iných metrík, poskytujúcich informácie o odcvičenej sérii. Užívateľom je poskytnutá možnosť výberu zobrazených metrík, podľa ich preferencií.

DIPLOMOVÁ PRÁCA

Autor práce: Bc. Simon Košina

Vedúci práce: Ing. Roman Juránek, Ph.D.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ