

Детекција на знаковен јазик во реално време во чест на “Пет и седумнаесет” од Ацо Шопов

Никола Диневски¹, Петар Атанасовски²

^{1, 2} Студент, Факултет за информатички науки и компјутерско инженерство (ФИНКИ)

УКИМ

Скопје, Р.С. Македонија

nikola.dinevski@students.finki.ukim.mk

petar.atanasovski@students.finki.ukim.mk

Абстракт — Поезијата е дел од нас и нашето културно наследство, преку неа претставена е историјата на нашиот народ. Сепак, убавината на поезијата не треба да остане ограничена во доменот на слухот, таа треба да биде заедничко искуство што ги надминува јазичните и комуникациските бариери. Преку можностите кои ни ги нуди денешната технологија можеме да им го приближиме нашиот секојдневен живот вклучувајќи ја и нашата богата поезија на глувата заедница а овој проект стои како доказ за моќта на оваа визија. Проектот е направен во чест 60 години од Скопскиот земјотрес (1963 година) чии страдања и последици се прикажани во песната „Пет и седумнаесет“ од Ацо Шопов. Креиран е систем кој препознава знаковен јазик во реално време, зборови од наведената песна. Знаковниот јазик се заснова на гестикации како основни елементи на комуникација, со развиена знаковна азбука и има специјален заштитен законски статус. Истиот не е универзален, секоја држава има свој, а овој проект се заснова на Македонскиот знаковен јазик. Користиме база на податоци за идентификување на 7 различни гестови со рацете, која се состои од слики извадени од видео снимка на веб-камера. На влез системот прифаќа снимка во реално време од веб-камера и во секоја рамка го идентификува гестот и го прикажува на екран. Имплементираме претходно трениран Tensorflow SSD Mobile Net V2 модел, кој дополнително го тренираме на нашата сопствена база на податоци. Постојат голем број на алатки и сензори за препознавање на знаковен јазик, но многу од нив се скапи и голем број на луѓе не можат да си ги дозволат. Нашиот модел за препознавање на знаковен јазик работи на 7 гестови, но многу лесно може да биде проширен за повеќе гестови или алфа-нумерички знаци. Без контролирана позадина и светлина успешно ги предвидува гестовите. Тој е ефтин и лесен за користење. Овој модел во реално време им го “преведува” тоа што човекот го покажува. Главната цел на нашиот проект е да се направи интелигентен модел кој им дозволува на глувите и немите лица да уживаат во поезијата на нашите писатели.

Клучни зборови—Детекција на знаковен јазик; OpenCV; Tensorflow; Deep Learning SDD; Python;

I. ВОВЕД

Знаковните јазици се создадени за да им помогнат на глувите и немите. Знаковниот јазичен систем е природен систем на комуникација и е на исто рамниште со вербалната комуникација. Говорот со знаковен јазик е намерно свесно изразување на внатрешната состојба или секој вид на изразување на мисли со цел да се разберат меѓусебно корисниците на знаковниот јазик преку специфично изграден јазичен систем на движења и поставеност на рацете. Овој јазик е систем на договорени движења на делови на телото. Основно средство за комуникација е раката, нејзината поставеност, движење, позиција и ориентација. Сите овие елементи се важни при пренесување на замислената мисла на соговорникот. Покрај рацете, за искажување на мисли, опстојуваат движења на главата, експресија на лицето, движење на устата и движење на телото. Како што кажавме претходно целта на овој проект е да се направи интелигентен модел кој им дозволува на глувите и немите лица да ја доживеат Македонската поезија со сета нејзина големина. При стартување на системот, се активира веб-камерата. Веб-камерата го снима човекот кој сака да комуницира со знаковен јазик. Во реално време сите рамки од снимката на веб-камерата, се обработуваат, па се предвидува во која од седумте класи, односно седумте гестови припаѓа конкретниот гест кој се прикажува на камера. Седумте гестови кои ги опфаќаме во овој проект се: “Да”, “Историја”, “Сила”, “Стемнето”, “Сонце”, “Надевање”, “Пет и седумнаесет”. Моделот кој го користиме е еден од претходно тренирани модели од Tensorflow, SSD Mobile Net V2. Кој ние го дотрениравме на наша база на податоци од слики од гестови. За тренирање користиме по 19 слики од гест (38 од гестот „Пет и седумнаесет“), или вкупно 152 слики. А за тестирање користиме по 1 слики од гест (2 од гестот „Пет и седумнаесет“), или вкупно 8 слики. Тренинг множеството е 95%, а множеството за тестирање изнесува 5%. Сликите за тренирање се направени на иста светлина и позадина бидејќи користиме претходно трениран модел кој знае вешто да се справи со проблемите на осветлување и позадина. Кога моделот ќе ја добие обработената слика за предвидување, ја бира најголемата веројатност за тоа во која класа припаѓа обработената слика. Потоа на излез ја враќа почетната слика заедно со информацијата за предвидената класа. Дополнително на екран во моменталната рамка се исцртува правоаголник околу гестот на човекот, односно околу неговата рака, заедно со веројатноста дека тој гест припаѓа на класата која моделот ја предвидил. Па така, на крај добиваме излез во реално време за тоа што значи, тоа што човекот го покажува на камера. Кодот, моделот и базата на податоци, односно целиот проект се наоѓа на GitHub на следниот линк : https://github.com/ndinevski/AS_RTSLD/.

II. ТЕХНОЛОГИЈА И АЛАТКИ

Во овој проект користиме голем број на технологии. Голем број на библиотеки, функции, алатки, платформи и апликации. Многу важно во изработка на проекти од овој тип претставува соработката на сите овие технологии и алатки меѓусебно. Верзиите на сите користени технологии и алатки мораат да бидат компатибилни една со друга. Доколку не се целиот проект нема да функционира. За таа цел потребно е да се посвети внимание на софпаѓањето на сите верзии. Да се направи истражување за најновите верзии,

компатибилноста на интерпретерот на програмскиот јазик, оперативниот систем, моделот, библиотеките,..

Користени технологии и алатки во овој проект се следните:

1. Интерфејс : PyCharm IDE (за извршување на кодот и менаџирање на сите верзии)
2. Оперативен Систем : Windows 10
3. Софтвер : Python (3.10.0), виртуелна околина креирана од нас, Numpy (1.23.5), Matplotlib (3.7.1) , cv2 (openCV)(4.7.0), Tensorflow (version 2.12.0), Bazel (5.3.0), Github, MSVC 2019, CUDA (11.0) and CuDNN (8.0) (За забрзување на тренирањето со Nvidia GPU), Protoc (3.20),...
4. Главни алатки :
 - Tensorflow - е бесплатна софтверска библиотека со отворен код за машинско учење и вештачка интелигенција. Може да се користи за низа задачи, но има посебен фокус на обука на длабоки невронски мрежи. TensorFlow најчесто се користи за класификација, перцепција, разбирање, откривање, предвидување и создавање. Во овој проект најповеќе од Tensorflow користиме Object Detection API. Како и претходно трениран модел SSD MobNet V2
 - OpenCV - е софтверска библиотека за компјутерска визија и машинско учење со отворен код. OpenCV е изграден за да обезбеди заедничка инфраструктура за апликации за компјутерска визија и да ја забрза употребата на машинската перцепција во комерцијалните производи. Ние го користиме за прибирање на сликите и градење на базата на податоци на тренинг и тест множеството. За обработка на сликите, како и за користењето на веб-камерата, обработката во реално време, имплементацијата со моделот на предвидување и прикажување на резултатите.
 - LabelImg - е графичка алатка за лабелирање на слики. Напишан е во Python и користи Qt за својот графички интерфејс. Лабелите се зачувуваат како XML датотеки. Ние го користиме за лабелирање на тренинг сликите и правење нивна соодветна XML датотека која подоцна ја користиме за создавање на записи за тренирање на моделот.

III. МАШИНСКО УЧЕЊЕ И ОТКРИВАЊЕ НА ОБЈЕКТИ

Машинското учење е гранка на вештачката интелигенција (ВИ) која се фокусира на развој на алгоритми и модели способни да учат и да донесуваат предвидувања или одлуки без да бидат експлицитно програмирани. Овие модели се дизајнирани автоматски да ги анализираат односите во рамките на податоците, извлекувајќи вредни сознанија и генерирајќи точни предвидувања. Основаната идеја, која го прави машинското учење на некој начин магично, е да им се овозможи на компјутерите да учат од искуства од минатото за да извршуваат конкретни задачи. За разлика од традиционалното програмирање, каде е потребно некој да ја осмисли и испише целата логиката на програмата, со што ќе го дефинира однесувањето на апликацијата врз одредени влезни податоци, машинското учење е автоматизиран процес во кој алгоритмот автоматски ги формулира правилата од податоците. Овие правила ја претставуваат логиката на програмата врз која се носат

заклучоци и предвидувања. Врз основа на методите и начинот на учење, машинското учење е поделено главно на четири типа, и тоа:

- 1) **Надгледувано машинско учење:** Тип на машинско учење во кој моделот се обучува со користење на обележани податоци. Обележените податоци се состојат од влез (познат како карактеристики) и излез (познат како ознака). Од овие податоци се генерира функција која го мапира влезот во посакуваниот излез.
- 2) **Ненадгледувано учење:** Тип на машинско учење каде моделот се обучува на неозначени податоци. За разлика од надгледуваното учење, нема претходно дефинирани ознаки или целни излези поврзани со влезните податоци. Наместо тоа, целта на ненадгледуваното учење е да се откријат скриени односи и зависности во податоците и даденото множество да се моделира во кластери (подгрупи или подмножествата) кои понатаму може да се користат.
- 3) **Полунадгледувано учење:** Тип на машинско учење што спаѓа помеѓу надгледувано и ненадгледувано учење. Во полу-надгледуваното учење, моделот се обучува со користење на комбинација од означени и неозначени податоци.
- 4) **Засилено учење:** Тип на машинско учење каде се дава делумна автономност на моделот за тој сам да носи одлуки, но секоја одлука (акција) има некоја последица на околината. Врз оваа последица моделот може да биде награден или не, така што ваквиот модел се стреми постојано да ја максимизира (или минимизира) својата награда.

Откривањето на објекти „Object Detection“ е технологија од компјутерска визија која вклучува идентификување и локализирање на објекти во дигитални слики или видеа. Алгоритмите кои се користат за детекција на објекти имаат цел да детектираат и класифицираат повеќе објекти од интерес во рамките на една слика. Откривањето на објекти се разликува од класификацијата на слики, каде што задачата е да се додели една ознака на цела слика. Наместо тоа, алгоритмите за откривање објекти не само што ги класифицираат објектите, туку и ја одредуваат нивната просторна локација во рамките на сликата. Кај нашиот модел ова може јасно да се воочи, така што тој исцртува рамка околу детектираниот објект или објекти во рамките на една слика. Со развојот на машинското учење, откривањето на објекти го заземе централното место во напредокот на компјутерската визија со многу развиени модели за специфични и генерални примени. Пристапот на овие проблеми преку машинско учење ни овозможува да ги оставиме скриените релации во пикселите целосно на машината, додека ние имаме за задача да подготвиме добро податочно множество врз кое истите ќе се обучуваат. Ваквите модели, при соодветно тренирање даваат одлични резултати за решавање на проблеми од областа на компјутерска визија, резултати кои е невозможно да се постигнат преку традиционален пристап на програмирање колку и труд да вложиме.

IV. ОБУКА НА МОДЕЛ ЗА ДЛАБОКО УЧЕЊЕ ПРЕКУ ТЕХНИКАТА НА ПРЕНОСНО УЧЕЊЕ (TRANSFER LEARNING)

Вреди да се напомене дека проблемите од компјутерска визија често бараат огромна количина означени податоци и значителни пресметковни ресурси за обука на модели за длабоко учење, па не би сакале да го правиме тоа од почеток на нашата машина. За да се надмине овој проблем постои техника наречена преносното учење. Преносното учење е

техника за машинско учење што го користи знаењето стекнато од решавање на една задача и го применува за да ја подобри изведбата на различна, но поврзана задача. Наместо да го започне процесот на учење од нула за новата задача, преносното учење му овозможува на моделот да го користи знаењето стекнато за време на обуката на различен модел. Како почетна точка се користи модел кој е претходно обучен на голема база на податоци и сложена задача. Ваквиот модел е веќе запознаен со околината и ова значително го намалува времето потребно за тренирање на моделот и ги подобрува перформансите. За потребите на овој проект избран е модел кој се заснова на надгледувано машинско учење и ја користи техниката на преносно учење (Transfer Learning). Моделот користен при решавање на проблемот е SSD Mobilenet V2 FPN-Lite 320x320. „Mobile Net SSD“ е невронска мрежа за детекција на повеќе објекти во една слика “Single-Shot Multibox Detection (SSD)”. Во рамки на сликата наоѓа координати на објектите и дава веројатности за припадност на класи на тие објекти. Врз основа на овие веројатности се извршува детекцијата на објекти. Првично, обучен е на базата на податоци COCO 2017 со слики намалени до резолуција 320x320, на овие податоци е обучен со ненадгледувано учење и има добра способност да ги подели множествата слики на кластери, преку кои потоа лесно носи заклучоци. Дополнителното обучување за нашиот проблем е направено преку претходно означените (лабелирани) слики од тренинг множеството. Моделот SSD Mobilenet вообичаено се користи на послаби компјутерски уреди како што е мобилниот (оттука иди и името Mobilenet), но и покрај тоа, дава перформанси со висока прецизност. API-то на Tensorflow го вклучува овој модел.

V. ПРОЦЕС

Во овој дел од документацијата ќе објасниме за тоа како го создадовме целиот проект од почетокот. Главната структура на процесот на креирање на овој проект се состои од неколку чекори. На почетокот, потребна ни е виртуелна околина и сите потребни пакети, библиотеки, алатки и технологии. Потоа собирањето на слики и креирањето на базата на податоци. Нивно лабелирање и поделба на сетовите. Потоа преземање на потребните модели и Object Detection API. Потоа создавање на мапа на лабелите и сликите, па креирање на TF записи. Како и поставување и ажурирање на Tensorflow Object Detection pipeline конфигурацијата. Потоа користење на transfer learning за создавање на deep learning model, и негово тренирање. Потоа анализа и евалуација на резултатите, дискусија и оптимизација на моделот. За на крај да можеме да направиме тестирање прво на слика, како и тестирање со веб-камера во реално време. И на крај зачувување и експортирање на истренираниот модел. Во следните параграфи ќе ги објасниме сите овие чекори подетално.

A. Потребни технологии и виртуелна околина

На почетокот на овој проект, направивме истражување за тоа со кои технологии и алатки би била најдобра реализацијата на нашата идеја и нашиот проект. Одлучивме тоа да го направиме со Tensorflow и Python. Како IDE го користиме PyCharm, бидејќи имаме претходни искуства со истото. За да можеме да ги користиме овие технологии и цел проект да функционира, сите технологии мораат да бидат компатибилни.

Пред да започнеме со било каква инсталација на пакети, библиотеки или алатки, направивме една виртуелна околина. Виртуелната околина е создадена врз постоечката Python инсталација, и може опционално да биде изолирана од пакетите во основната средина, така што се достапни само оние кои се експлицитно инсталирани во виртуелната средина. На кратко кажано се користи за менаџирање на Python пакети за различни проекти. Ни дозволува да избегнеме глобално инсталирање на Python пакети кои би можеле да ги оштетат системските алатки или да влијаат на други проекти. Бидејќи располагаме со голем број на пакети и библиотеки и сакаме да избегнеме што е можно поголем број на конфликти со останати проекти и основната инсталација на Python, имплементацијата на виртуелна околина е соодветна за овој проект.

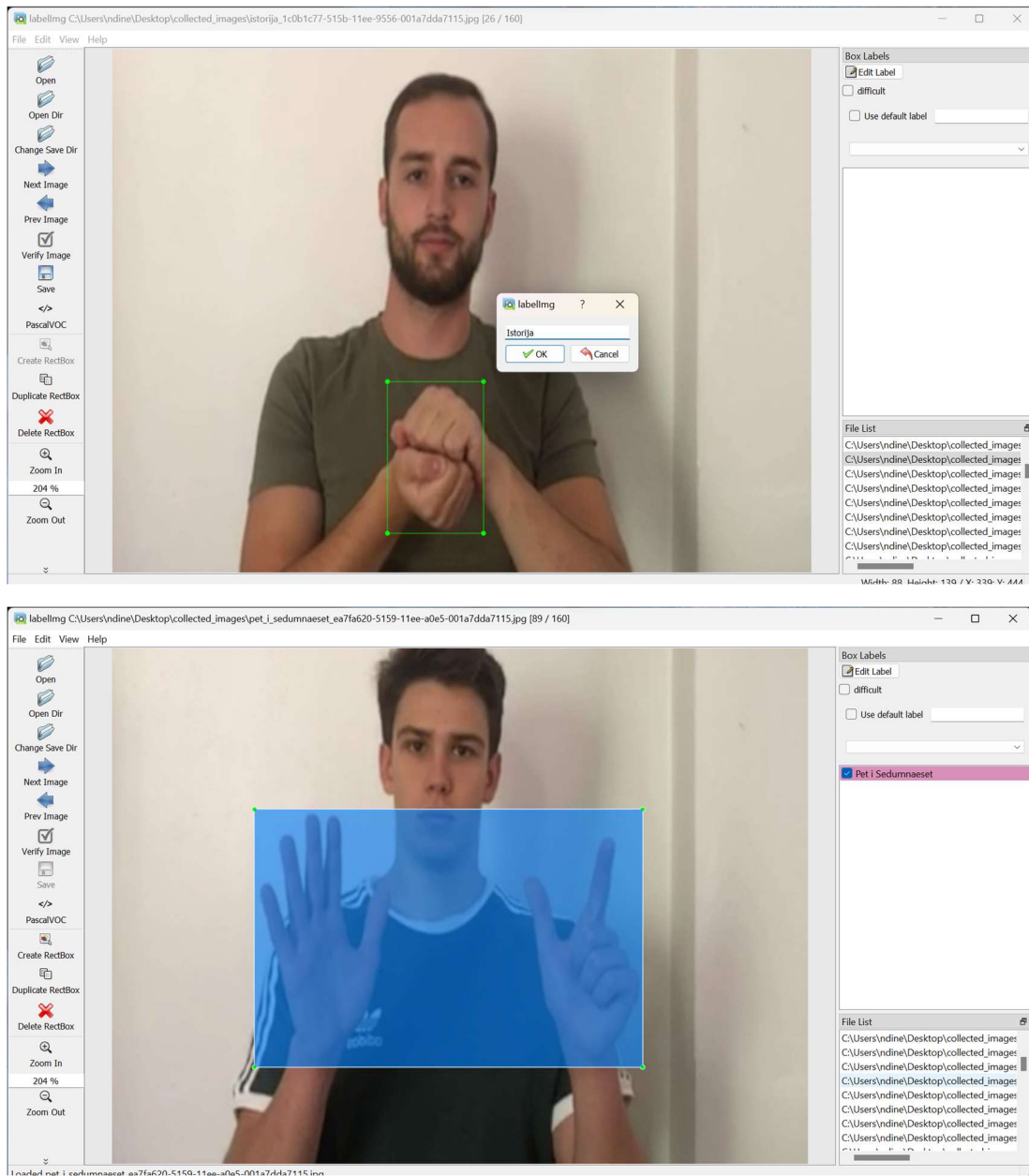
В. Собирање на слики

Следен чекор во изработката на овој проект е собирањето на слики. За да може моделот успешно да предвидува гестови, потребно е да биде трениран на истите тие гестови. Најповолно, би било да биде трениран на што е можно повеќе слики. Слики во различни големини, светлина, раце од различни луѓе,... За таа цел, напишавме кратка Python скрипта. Во оваа скрипта главно се користи OpenCV. На почетокот ја отвораме веб-камерата, потоа одиме низ сите лабели (гестови) и за секоја лабела зачувуваме слика. Сликите се зачувуваат на секои 3 секунди, додека снима камерата. Тоа го правиме за сите гестови. Сите слики се именуваат со единствен uid, така што никои две слики немаат исто име. Овој процес трае неколку минути, бидејќи за секој од гестовите правиме по 10 слики. Но го повторуваме овој процес неколку пати, бидејќи правиме слики од различни луѓе, на различни позадини, и различни светлини. Откако ќе ги добиеме сите слики, следен чекор е да ги лабелираме и поделиме во сетови за тренирање.

С. Лабелирање на слики

Лабелирањето на сликите е многу важен чекор во овој проект. Лабелирањето на слики е вид на означување на податоци што се фокусира на идентификување и означување конкретни детали во сликата. Во компјутерската визија, означувањето на податоците вклучува додавање ознаки на необработени податоци како што се слики и видеа. Секоја ознака претставува класа на објект поврзана со податоците. Надгледуваните модели на машинско учење користат ознаки кога учат да идентификуваат одредена класа на објекти во неklasифицирани податоци. Тоа им помага на овие модели да го поврзат значењето на податоците, што може да помогне во обуката на моделот. Лабелирањето на сликата се користи за креирање на податоци за моделите за компјутерска визија, кои се поделени на множества за тренирање, кои се користат за првично обучување на моделот и комплети за тестирање/валидација што се користат за проценка на перформансите на моделот. Базата на податоци се користи за да се обучи и процени моделот, а потоа моделот може автоматски да доделува ознаки на невидени, односно да класифицира неозначени податоци. Ние ја користиме алатката LabelImg. Тоа е графичка алатка напишана во Python за лабелирање на слики и е доста едноставна за користење. По стартување на алатката, го отвораме фолдерот со претходно сите собрани слики. Потоа една по една, за секоја слика ставаме лабела. Лабелите се ставаат така што селектираме одреден дел од сликата и го именуваме. Така, во

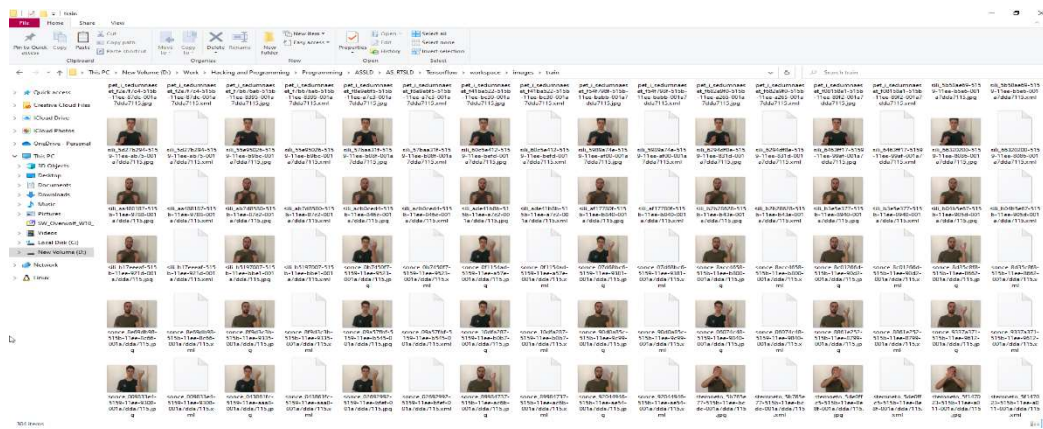
нашиот случај, селектираме правоаголна форма околу гестот кој се покажува на камера, а потоа ставаме име на таа класа. На пример селектираме правоаголник околу стисната тупаница (гестот “Да”), и потоа запишуваме лабела “Да”. Ова го правиме за секоја од сликите. На овој начин ние креираме XML датотеки во кои има информации за тоа кој гест е на сликата и каде се наоѓа. Неколку чекори подоцна овие датотеки ќе ни бидат потребни за да направиме TF записи, за да можеме да го тренираме моделот. На слика 1 и 2 можеме да видиме како изгледа лабелирањето на сликите. На слика 1 е селекцијата на гестот и запишување на класата. На слика 2 е означена класата со соодветната слика, и е создадена XML датотеката.



Сл. 1. и 2. LabelImg

D. Поделба на тренинг и тест множества

После лабелирањето на сликите, ја правиме распределбата на сетовите. Во Машинско учење, поделбата на нашата база на податоци за моделирање на примероци за тренирање и тестирање е веројатно еден од најраните чекори за препроцесирање што треба да ги преземеме. Создавањето различни примероци за обука и тестирање ни помага да ги оцениме перформансите на моделот. Тренинг сетот е подмножество податоци и е одговорно за обука на моделот. Обично моделот за машинско учење учи да предвидува со разбирање на шемите и врските скриени во податоците. Моделот ќе научи од шаблоните и односите помеѓу променливите за тежина во нашиот пример. По обуката, потврдувањето и изборот на моделот, зависи од тестирањето на неговите перформанси за извлечената подгрупа на податоци кои го сочинуваат тест сетот. Бидејќи најчесто располагаме со повеќе модели, со тест сетот донесуваме одлука кој од нив е најдобар и дава најдобри резултати. Вкупно направивме по 20 слики за гест. За секој од гестовите направивме слики од двајцата, што значи имаме слики од различни луѓе. На различни позадини и во различни светлини, со цел да добиеме “пофлексибилна” база на податоци. За моделот да биде прецизен кога ќе добие снимка, односно слики од нови луѓе, нови позадини и различни светлини. Вкупно имаме 160 слики. За тренинг множеството користиме по 19 слики од гест, односно 152 слики, додека за тест множеството користиме по 1 слика од гест, односно 8 слики. На крај добиваме тренинг наспрема тест множество во однос 95% наспрема 5%. Не постои оптимален процент на поделба. Треба да се дојде до сооднос кој одговара на барањата и ги задоволува потребите на моделот. Сепак, постојат две главни грижи при одлучувањето за оптималната поделба. Ако има помалку податоци за обука, моделот за машинско учење ќе покаже голема варијанса во обуката. Со помалку податоци за тестирање, статистиката за евалуација на моделот ќе има поголема варијанса. Во суштина, треба да се дојде до оптимална поделба што одговара на потребата на базата на податоци и моделот. И за да се дојде до оптималната поделба, најчесто треба да се пробува и експериментира со различни соодноси на тренинг и тест множество. Повеќето модели имаат и валидациско множество, кое се користи за модифицирање и оптимизирање на хиперпараметрите на моделот. Најчести соодноси за тренинг и тест множество кои се користат се 60:40, 70:30, 80:20,... На слика 3 може да се види како изгледаат сликите во тренинг множеството, заедно со соодветните XML датотеки. Ние се одлучивме за оваа распределба бидејќи добиваме најдобри резултати при тестирање со видео камерата во реално време.



Сл. 3. Фолдер на тренинг множество

Е. Tensorflow и конфигурација

Следен чекор е да ги подготвиме овие податоци што ги имаме за тренирање. Како технологија за работа, го избравме Tensorflow. Машинското учење е сложена дисциплина, но имплементацијата на моделите за машинско учење е многу полесно отколку што беше порано, благодарение на библиотеки за машинско учење како што е TensorFlow на Google. Tensorflow го олеснува процесот на собирање податоци, модели за обука, правење предвидувања и оптимизација на резултати. Може да обучи и да прави длабоки невронски мрежи за рачно напишана класификација, препознавање слики, зборови, рекурентни невронски мрежи, секвенцни модели, обработка на природен јазик и симулации. TensorFlow исто така има широка библиотека на претходно обучени модели кои можат да се користат во различни проекти. Најголема придобивка што ја нуди TensorFlow за развој на машинско учење е апстракцијата. Наместо да се занимава со големи детали за имплементација на алгоритми, или да пронајди соодветни начини за поврзување на излезот од една функција со влезот на друга, програмерот може да се фокусира на целокупната логика на апликацијата. TensorFlow се грижи за деталите зад сцената. Ние во овој проект, го користиме SSD MobNet V2 моделот. Тој е веќе трениран модел, кој ние го дообучуваме и го имплементираме на нашите податоци, за нашиот проект. За да го искористиме Tensorflow, во нашиот проект ние ја клонираме библиотеката за модели на Tensorflow, од официјалниот GitHub. Така на располагање ги имаме повеќето модели, пакети, функции и алатки за детекција на објекти, тестирање и тренирање, алатки за статистики и евалуација и слично. Тука е важно да се инсталираат и сите потребни библиотеки и пакети кои оваа библиотека ги користи, за се да функционира успешно. За ова да го постигнеме, потребни се низа на неколку команди во терминалот. Тука има и скрипта за верификација на тоа дали се е во ред и успешно инсталирано. Додека оваа скрипта не помине и каже дека се е во ред, не можеме да продолжиме понатаму со тренирањето на моделот. Скриптата помага, бидејќи ни кажува за секоја грешка која ја имаме, грешка во податоците, грешка во верзиите на библиотеките и нивната взаемна компатибилност, непостоечки пакети и слично. Со повеќето проблеми се справуваме така што ги инсталираме пакетите кои ги немаме или ги намалуваме или зголемуваме верзиите на веќе постоечките пакети со цел да бидат компатибилни една со друга. После успешно извршување на скриптата за верификација можеме да продолжиме со креирањето на мапа на лабелите со сликите и самите TF записи.

Ф. TF записи и мапа на лабели

Следно креираме мапа на лабели, која всушност за секоја лабела (гест) кој го предвидуваме се мапира во единствено ID, ова го правиме за да можеме да ги креираме TF записите. Следно со лабелираните тренинг и тест сетовите кои ги добивме претходно и оваа мапа на лабели можеме да ги креираме TF записите. TF записите се компонента на Tensorflow, која всушност служи како бинарен формат за складирање на податоци. Ако работите со голем број на податоци, користењето на бинарен формат за складирање на вашите податоци може да има значително влијание врз перформансите на вашиот pipeline config и како последица на времето за обука на вашиот модел. Бинарните податоци завземаат помалку простор на дискот, потребно е помалку време за копирање и може многу поефикасно да се читаат од дискот. Овие перформанси не се единствената предност на TF

записите. Го олеснува комбинирањето на повеќе бази на податоци и се интегрира со функционалноста за импортирање на податоци и препроцесирање на податоци од библиотеката. Особено за множества на податоци кои се премногу големи за да бидат целосно складирани во меморијата, ова е предност бидејќи само податоците што се потребни во тој момент (batch) се вчитуваат од дискот и потоа се обработуваат. Во нашиот случај ние всушност ги поврзуваме XML датотеките, со означени лабели и позиција на гест, заедно со мапата на лабели и сето тоа го претставуваме во бинарен формат, како еден TF запис. За креирањето на записите, користиме Python скрипта. На крај добиваме два TF записи, еден тренинг TF запис и еден тест TF запис.

G. Pipeline конфигурација

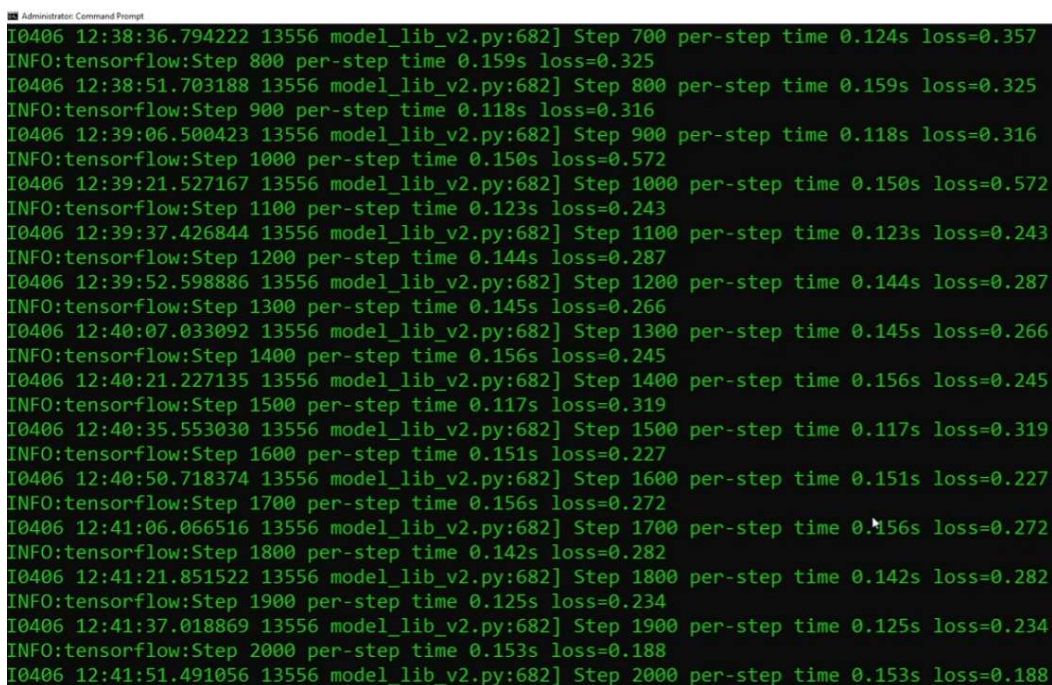
Последен чекор пред тренирањето на моделот, е да се постави pipeline конфигурацијата за детекција на објекти на Tensorflow. За таа цел правиме Python скрипта. Оваа скрипта го зема веќе тренираниот модел, и ја користи pipeline конфигурацијата на SSD MobNet V2. Tensorflow API-то за детекција на објекти користи protobuf датотеки за да го конфигурира тренинг процесор, односно pipeline конфигурацијата. Тоа е датотека која се состои од параметри и метрики како број на класи, препроцесирање на слики и нивни карактеристики, извлекува карактеристики од сликите, активациски функции на невроните, калкулатор за софпаѓања, постпроцесирање, и голем број на други хиперпараметри. Откако ќе го земе скриптата веќе тренираниот модел, ги менуваме потребните параметри и карактеристики. И правиме врска со мапата на лабели од нашите гестови. Како и TF записите на тренинг множеството и на тест множеството. На тој начин, на крај добиваме различен модел. Надграден модел од веќе тренираниот модел на Tensorflow. Овој нов модел е моделот кој ќе го тренираме. Тој сега има врска со нашата база на податоци, со нашите гестови и е спремен да биде трениран. Pipeline конфигурацијата која ја добивме по менување на веќе постоечката од Tensorflow моделот, е нова конфигурација специфична за нашиот проект, и може да биде споделувана и користена за слични проекти како нашиот.

H. Тренирање

Процесот на тренирање кој може да се види на слика 4 започнува со повикување на скрипта од Tensorflow библиотеката и се специфицира локацијата на сликите од множеството за тренирање и моделот. Тренирање е делот во кој моделот учи од дадените влезни слики. Алгоритамот за учење итерира низ тренирачкото множество и правејќи грешки ги ажурира параметрите на својата невронска мрежа. Една итерација низ целото множество е позната како епоха при учење. Пред да започнеме со тренирање, важно е да го наведеме бројот на епохи кои ќе ги измине моделот. Бројот на епохи е хиперпараметар чиј избор е клучен за да се постигнат добри перформанси на моделот. При изградба на еден модел сакаме тој добро да генерализира, односно да се справува со генерални податоци, онакви какви што до сега немал видено. За да го постигнеме тоа, од една страна, потребно е да го истренираме моделот доволно долго, на повеќе епохи за да ги научи зависностите меѓу податоците и да носи добри заклучоци. Но, од друга страна, тука може да се јави и проблем на претренирање. Доколку го оставиме моделот да тренира на голем број на епохи, тој ќе започне да бара ситници во податоците од тренирачкото множество со цел да ги погодува

перфектно нивните клси. На тој начин ќе ги научи сите слики за тренирање „на памет“. Ова е непосакувано затоа што кога моделот ќе се соочи со досега сеуште не видена слика, ќе ги бара тие ситници кои не секогаш ќе ги пронајде и поради тоа ќе носи грешни одлуки. Во ваков случај моделот нема добро да генерализира и станува збор за претренирање, односно overfitting. Проблемот на overfitting е еден од најчестите проблеми при изградба на било каков модел од областа на машинско учење. Со цел да се надмине овој проблем и да се одреди оптималниот број на епохи, вообичаена практика е да се следат перформансите на моделот на посебна база на податоци за валидација, која во нашиот случај се сликите од тест множеството. Податокот за валидација не се користи за обука, туку се користи за евалуација на перформансите на моделот за време на процесот на обука, како и по завршување на процесот на обука. Податоците од ова множество се непознати за моделот и тој не може да ги научи и да ги ажурира своите параметри врз основа на нив. Со следење на метриките за валидација, како што се точноста или загубата, во различни епохи, може да се набљудува како еволуираат перформансите на моделот. Прецизноста, точноста и чувствителноста се вообичаено користени метрики за оценка на перформансите на моделите за машинско учење, особено во задачите на класификација. Секоја од овие метрики дава увид од различни аспекти на перформансите на моделот во однос на исправноста и комплетноста. За подобро разбирање на овие метрики воведени се некои универзални променливи како вистински позитиви (TP), вистински негативни (TN), лажни позитивни (FP), лажни негативни (FN). Најкористени метрики се:

- 1) **Точност** : Ја мери целокупната исправност на предвидувањата земајќи ги предвид и вистинските позитивни и вистинските негативни. $(TP + TN) / (TP + TN + FP + FN)$
- 2) **Прецизност** : Го мери уделот на точно предвидените позитивни примероци (вистински позитивни) од вкупните примероци предвидени како позитивни $(TP) / (TP + FP)$
- 3) **Одзив** : Го мери процентот на правилно предвидените позитивни примери (вистински позитивни) од вистинските позитивни примероци. $(TP) / (TP + FN)$



```

Administrator: Command Prompt
I0406 12:38:36.794222 13556 model_lib_v2.py:682] Step 700 per-step time 0.124s loss=0.357
INFO:tensorflow:Step 800 per-step time 0.159s loss=0.325
I0406 12:38:51.703188 13556 model_lib_v2.py:682] Step 800 per-step time 0.159s loss=0.325
INFO:tensorflow:Step 900 per-step time 0.118s loss=0.316
I0406 12:39:06.500423 13556 model_lib_v2.py:682] Step 900 per-step time 0.118s loss=0.316
INFO:tensorflow:Step 1000 per-step time 0.150s loss=0.572
I0406 12:39:21.527167 13556 model_lib_v2.py:682] Step 1000 per-step time 0.150s loss=0.572
INFO:tensorflow:Step 1100 per-step time 0.123s loss=0.243
I0406 12:39:37.426844 13556 model_lib_v2.py:682] Step 1100 per-step time 0.123s loss=0.243
INFO:tensorflow:Step 1200 per-step time 0.144s loss=0.287
I0406 12:39:52.598886 13556 model_lib_v2.py:682] Step 1200 per-step time 0.144s loss=0.287
INFO:tensorflow:Step 1300 per-step time 0.145s loss=0.266
I0406 12:40:07.033092 13556 model_lib_v2.py:682] Step 1300 per-step time 0.145s loss=0.266
INFO:tensorflow:Step 1400 per-step time 0.156s loss=0.245
I0406 12:40:21.227135 13556 model_lib_v2.py:682] Step 1400 per-step time 0.156s loss=0.245
INFO:tensorflow:Step 1500 per-step time 0.117s loss=0.319
I0406 12:40:35.553030 13556 model_lib_v2.py:682] Step 1500 per-step time 0.117s loss=0.319
INFO:tensorflow:Step 1600 per-step time 0.151s loss=0.227
I0406 12:40:50.718374 13556 model_lib_v2.py:682] Step 1600 per-step time 0.151s loss=0.227
INFO:tensorflow:Step 1700 per-step time 0.156s loss=0.272
I0406 12:41:06.066516 13556 model_lib_v2.py:682] Step 1700 per-step time 0.156s loss=0.272
INFO:tensorflow:Step 1800 per-step time 0.142s loss=0.282
I0406 12:41:21.851522 13556 model_lib_v2.py:682] Step 1800 per-step time 0.142s loss=0.282
INFO:tensorflow:Step 1900 per-step time 0.125s loss=0.234
I0406 12:41:37.018869 13556 model_lib_v2.py:682] Step 1900 per-step time 0.125s loss=0.234
INFO:tensorflow:Step 2000 per-step time 0.153s loss=0.188
I0406 12:41:51.491056 13556 model_lib_v2.py:682] Step 2000 per-step time 0.153s loss=0.188

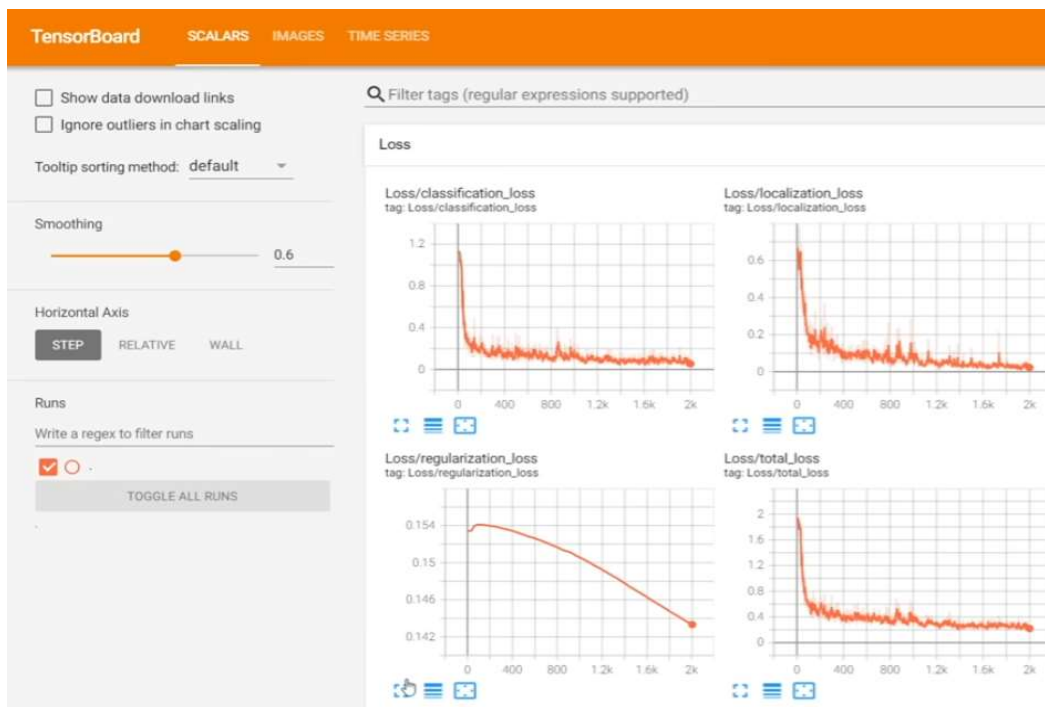
```

Сл. 4. Процес на тренирање

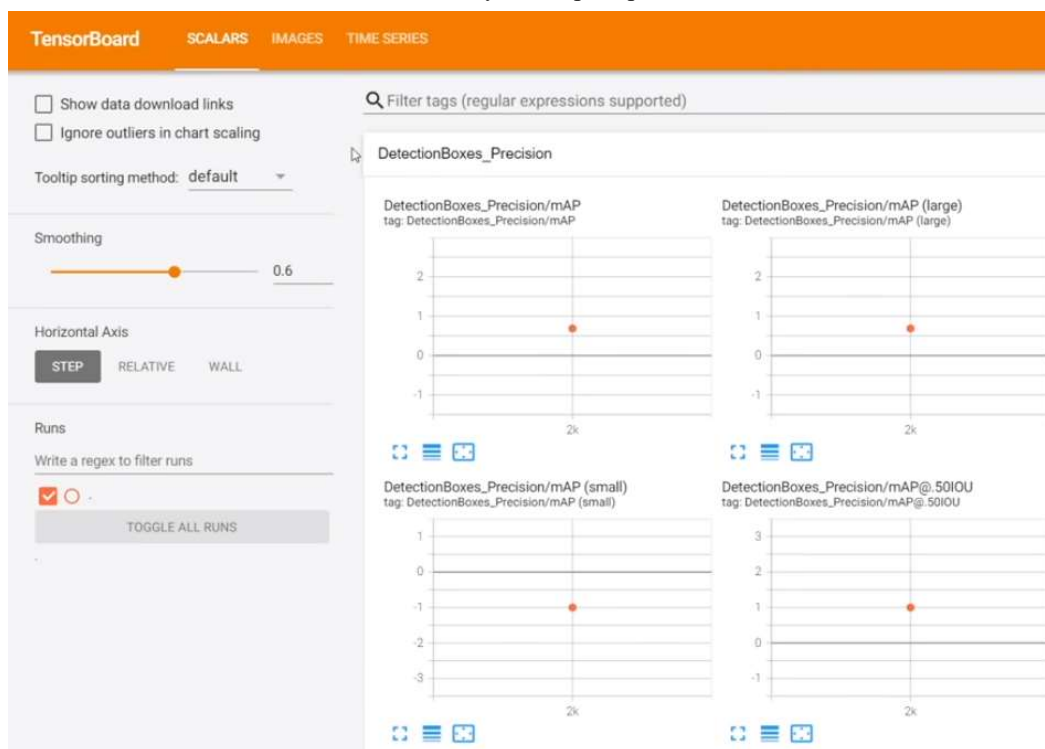
Важно е да се напомене дека процесот на тренирање на модел за машинско учење може да се одвива преку централната процесорска единица (CPU) или преку графичката картичка (GPU). Користењето на графичката картичка при тренирање може да обезбеди доста предности и вреди да се разгледа таа опција. Основна функција на графичката картичка е да ги рендерира сликите на екран, односно да извршува графички интензивни задачи, затоа таа нуди голема пресметувачка моќност, брза меморија и ефикасност што се од голема значајност при тренирање на модел, особено од областа на компјутерска графика. CUDA (Compute Unified Device Architecture) е паралелна компјутерска платформа и програмски модел развиен од NVIDIA. Им овозможува на програмерите да ја искористат пресметковната моќ на NVIDIA GPU (графички процесорски единици) за компјутерски задачи за општа намена, вклучувајќи машинско учење, научни симулации, обработка на слики и многу повеќе. При тренирање на модел препорачливо е да се тренира моделот преку CUDA на графичката картичка, посебно ако го тренираме моделот од почеток (слика 109). Во нашиот случај тренирањето целосно се одвиваше на процесорот бидејќи моделот е претходно трениран и целокупниот процес траеше околу 2 часа, па немаше потреба од искористување на ресурсите на графичката картичка.

1. Евалуација и checkpoints

За максимално искористување на можностите на моделот при даденото множество за тренирање, ние се одлучивме нашиот модел да го оставиме да се тренира на 10000 епохи. Притоа, на секој 1000 епохи овозможивме верзија од моделот автоматски да се зачувува на нова локација со што би добиле 10 различни модели тренирани на различен број на епохи. Со понатамошна споредба на различните модели, можевме јасно да видиме кој модел дава најдобри перформанси на множеството за тестирање, за тој модел понатаму да го зачуваме и искористиме како финален. Првиот модел кој го испитавме беше оној тестиран на 10000 чекори и веднаш заклучивме дека не сме ги постигнале посакуваните резултати. Но со понатамошно тестирање на моделите тренирани на помал број на епохи, стигнавме до заклучок дека моделот кои ги има перформансите кои ги посакувавме беше оној трениран на 7000 епохи. Она што се случи при процесот на тренирање на нашиот модел беше претренирање, односно моделот до 7000-та епоха учеше се подобро да генерализира. Но, во понатамошните епохи почна да ги учи податоците од тренинг множеството на памет. Моделот обучен на 10000 епохи кога ќе се соочи со проблем кој веќе го видел при тренирање го класифицира со точност од скоро 100%, но проблемот е тоа што кога ќе се соочи со нов проблем оваа точност драстично се намалува и не ги задоволува потребите на нашиот проект. Во споредба со него, моделот трениран на 7000 епохи лесно и прецизно ги препознаваше симболите од знаковниот јазик на тест множеството. На слика 5 можеме да ги видиме статистиките за перформансите на моделот за време на тренингот, додека на слика 6 можеме да ги видиме статистиките од евалуацијата на моделот по тренирањето.



Сл. 5. Статистики за загуба на тренирачко множество

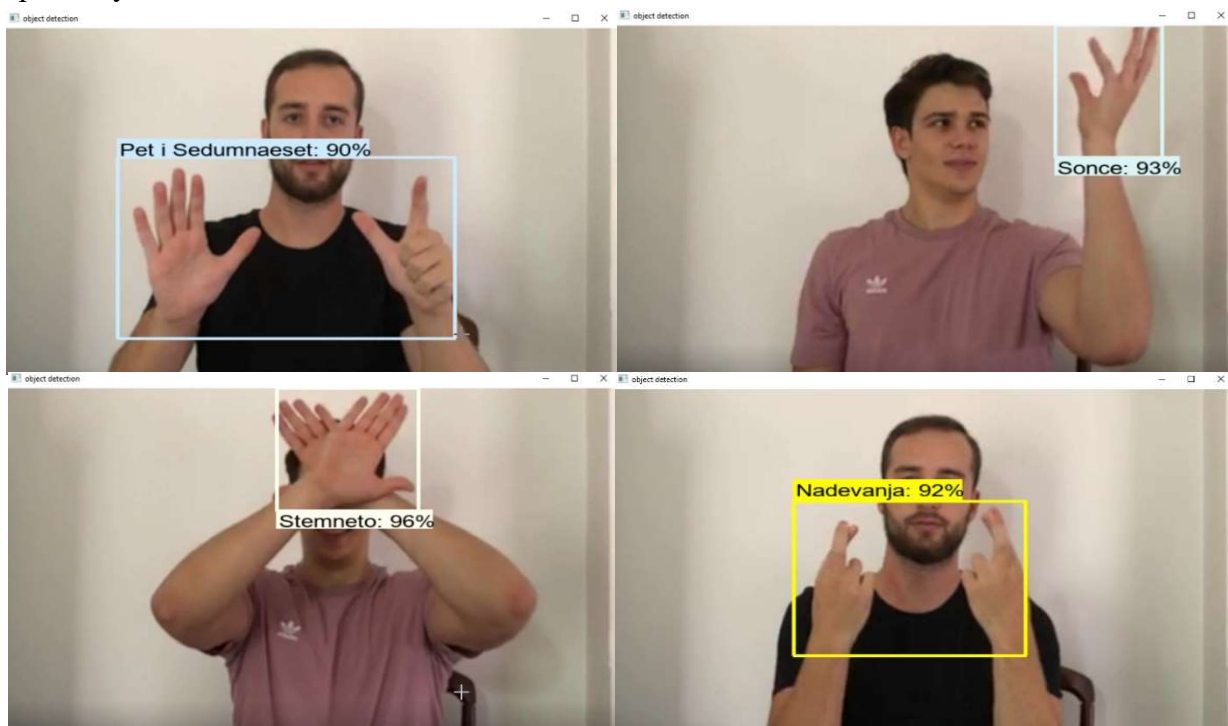


Сл. 6. Статистики за прецизност на тренирачко множество

1. Тестирање на слика и веб-камера во реално време

За спроведување на евалуацијата на моделот, прво ги следиме неговите перформанси на конкретна слика од тест множеството. За таа цел, напишавме python скрипта преку која го наведуваме моделот кој сакаме да го отвориме (моделот од специфичниот checkpoint).

Потоа, избираме конкретна слика од тест множеството која ја пуштаме до моделот за детекција. Моделот ги применува алгоритмите за детекција при што ги детектира деловите од сликата кои ни се од интерес и им придружува одредена веројатност за припадност на некоја од класите. Преку библиотеката `matplotlib` оваа финална слика се исцртува на екран со што можеме да го визуелизираме резултатот. Исто така напишавме нова `python` скрипта преку која слично како и при креирањето на податочните множества, ја отвараме камерата на нашиот уред и при нејзина активност правиме слика на секоја рамка (`frame`). Слика ја даваме на моделот кој ги применува алгоритмите за детекција при што ги детектира деловите од сликата и им придружува одредена веројатност. На овој начин можеме да го видиме однесувањето на моделот кое нам ни е најважно и сме целосно сигурни дека нашиот модел е соодветен и ги задоволува барањата на проблемот. Со тоа сме спремни за понатамошна обработка и зачувување на моделот. На слика 7 може да се виде како моделот успешно ги предвидува гестовите.



Сл. 7. Тест во реално време

К. Замрзнување на графот и зачувување на моделот

Откако ќе се осигуриме за однесувањето на нашиот модел и ќе се задоволиме со неговите перформанси, потребно е да го замрзнеме графот. Замрзнувањето на графот се однесува на конвертирање и зачувување на параметрите на обучениот модел во датотека или формат што може да се користи за заклучување без дополнителна обука. Откако графикот ќе се замрзне, параметрите на моделот стануваат фиксни и повеќе не се ажурираат за време на предвидување, што резултира со неколку придобивки како: ефикасност на заклучоците, лесна преносливост, како и конзистентност на резултатите. Штом графикот се замрзне, дополнителна обука не е можна освен ако моделот не се одмрзне и тренира. Затоа, клучно е да се сме сигурни дека моделот е соодветно обучен и оценет пред да се го преземеме овој

чекор. Зачувувањето на моделот го направивме во „TensorFlow.js (TF.js)“ формат. Тако што ја инсталиравме библиотеката „tensorflowjs“ во виртуелната околина и преку командна линија повикавме скрипка која го зачува моделот на локацијата `Tensorflow\workspace\models\my_ssd\export`. Следејќи ги овие чекори добивме модел во формат `TensorFlow.js`, што во иднина би ни овозможило да го распоредиме и извршуваме моделот во веб-прелистувачи или Node.js средини користејќи ја библиотеката `TensorFlow.js`. Дополнително, преку повикување на скрипта од командна линија, верзија на моделот зачувавме и во „TensorFlow Lite (TFLite)“ формат на локација `Tensorflow\workspace\models\my_ssd\tfliteexport`. Така добивме и модел во формат `TensorFlow Lite`, кој е оптимизиран за распоредување на уреди со ограничени ресурси и обезбедува ефикасни способности за заклучување.

VI. МОЖНОСТИ НА МОДЕЛОТ ВО ИДНИНА

По успешно извршување на сите досега наведени чекори добиваме функционална апликација која може да има широка примена, како и однапред трениран модел чија функционалност може лесно да се надополни. Значително важно е што моделот не е неопходно да работи во рамките на апликацијата, туку преку замрзнатите верзии, неговата примена може да биде на било која друга апликација. Вака готовиот шаблон на модел може да најде огромна примена во наши идни проекти, поврзани со детекција на делови од слика како на пример: детекција на лице, детекција на луѓе, детекција на израз на лице. Дополнително може да се прошири да работи и со поголем број на гестови односно зборови од песната „Пет и седумнаесет“ на Ацо Шопов. За повеќето зборови во таа песна, гестовите не се еден гест, туку движење со рацете и менување на гестот за покажување на еден единствен збор. За таа цел, за да се постигне препознавање на таков тип на зборови, моделот треба да биде трениран дополнително и со различна логика. Односно, треба да препознава и да зачувува претходни гестови, за понатаму да одлучи кој бил зборот кој е покажан.

VII. ЗАКЛУЧОК

Проблемот кој што ние го решаваме со креираната апликација е добро познат проблем со кој општеството се соочувало од секогаш и сеуште се соочува. За лицата со посебни потреби е тешко да ги доживеат стиховите на Македонските поети и да уживаат во нивната поезија. Користејќи го моделот кој го креиравме, овој проблем делумно се надминува. Еден секојдневен пример каде би нашла примена оваа апликација се собирите на кои се чита поезија, многу лесно може да се задолжи човек кој ќе го преведува читаното на знаковен јазик а моделот ќе испишува значењето во форма на текст. Наша цел како развивачи на софтвер е да ги воочиме потребите во општеството и да се потрудиме да ги решиме што е можно поефикасно. Достапни се различни информации, модели и апликации кои доколку знаеме вешто да ги пронајдеме, имплементираме и модифицираме за нашите потреби, значително ја олеснуваат работата и даваат одлични крајни резултати. На крај го добивме тоа што го замисливме. Проектот може дополнително да се прошири на сите зборови од песната „Пет и седумнаесет“ како и други песни, исто така да биде трениран на поголем број слики, од повеќе луѓе и на различни услови, со што би се зголемиле неговите перформанси.

VIII. РЕФЕРЕНЦИ

- [1] Nicholas Renotte, Tensorflow Object Detection Course, [Онлајн], Достапно на линкот : <https://github.com/nicknochnack/TFODCourse>
- [2] Tensorflow, [Онлајн], Достапно на линкот : <https://www.tensorflow.org/>
- [3] Object Detection, [Онлајн], Достапно на линкот : <https://medium.com/ml-research-lab/what-is-object-detection-51f9d872ece7>
- [4] Machine Learning, [Онлајн], Достапно на линкот : <https://www.ibm.com/topics/machine-learning>
- [5] About Train, Validation and Test Sets in Machine Learning, [Онлајн], Достапно на линкот : <https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7>
- [6] Labelling for Image Annotation [Онлајн], Достапно на линкот : <https://viso.ai/computer-vision/labeling-for-image-annotation/>
- [7] Capture video from camera/file with OpenCV in Python, [Онлајн], Достапно на линкот : <https://note.nkmk.me/en/python-opencv-videocapture-file-camera/>
- [8] Build from source on Windows, [Онлајн], Достапно на линкот : https://www.tensorflow.org/install/source_windows
- [9] What Is Transfer Learning? Exploring the Popular Deep Learning Approach., [Онлајн], Достапно на линкот : <https://builtin.com/data-science/transfer-learning>
- [10] Pipeline configuration files, [Онлајн], Достапно на линкот : <https://www.ibm.com/docs/en/cics-ts/5.6?topic=infrastructure-pipeline-configuration-files>
- [11] Tensorflow Records? What they are and how to use them, [Онлајн], Достапно на линкот : <https://medium.com/mostly-ai/tensorflow-records-what-they-are-and-how-to-use-them-c46bc4bbb564>