|  |
| --- |
| Tehnička škola Ruđera Boškovića |
| Machine learning aplikacija |
| Mentor: Zlatko Nadarević |

|  |
| --- |
| Dorian Granoša 4.H  Zagreb, svibanj 2019. |

Sadržaj

[1. Uvod 2](#_Toc7428651)

[2. Strojno učenje 2](#_Toc7428652)

[3. Zadaci strojnog učenja 2](#_Toc7428653)

[3.1. Učenja pod nadzorom 2](#_Toc7428654)

[3.2. Učenje bez nadzora 3](#_Toc7428655)

[3.3. Učenje bez nadzora 3](#_Toc7428656)

[4. Duboko učenje 3](#_Toc7428657)

[5. Umjetne neuronske mreže 3](#_Toc7428658)

[6. Konvolucijske neuronske mreže 4](#_Toc7428659)

[7. Postupci izrade projekta 7](#_Toc7428660)

[7.1. Odabir dataset-a 8](#_Toc7428661)

[7.2. Odabir modela 9](#_Toc7428662)

[7.3. Treniranje modela 10](#_Toc7428663)

[7.4. Pretvorba modela pogodna za mobilnu upotrebu 12](#_Toc7428664)

[7.5. Izrada mobilne aplikacije 12](#_Toc7428665)

[8. Kod 14](#_Toc7428666)

[8.1. Manifest 14](#_Toc7428667)

[8.2. Grafičko sučelje 15](#_Toc7428668)

[8.3. CameraActivity 16](#_Toc7428669)

[8.4. Camera2BasicFrament 20](#_Toc7428670)

[8.5. PoseEstimation 31](#_Toc7428671)

[8.6. DrawView 32](#_Toc7428672)

[9. Zaključak 35](#_Toc7428673)

[10. Literatura 36](#_Toc7428674)

# Uvod

Machine learning aplikacije je mobilna aplikacija koja se služi strojnim učenjem (eng. Machine learning) kako bi dobili čovjekovu poziciju na slici. U današnje vrijeme kada većina ljudi ide u teretane i tamo vježba, tijekom vježbanja ljudi u mislima odlutaju te prestanu brojati ponavljanja što rezultira u tome da ne znaju gdje su stali. Ovaj rad rješava taj problem tako da umjesto korisnika aplikacija broji ponavljanja. Kada se uključi aplikacija uključi se i kamera koju korisnik usmjeri prema sebi. Dok korisnik vježbanja aplikacija sama broji ponavljanja te se korisnik ne treba brinuti o tome. Aplikacija pamti maksimalan broj ponavljanja do tada izveden što potiče korisnika na više vježbe. Aplikacija radi tako da s kamere uzima jednu sličicu i provodi je kroz neuronsku mrežu koja na kraju daje poziciju čovjeka te pomoću toga broji ponavljanja.

# Strojno učenje

Strojno učenje (eng. Machine learning) je znanstveno istraživanje algoritama i statističkih modela koje računalni sustavi koriste za učinkovito obavljanje određenog zadatka bez korištenja eksplicitnih uputa, umjesto da se oslanjaju na obrasce i zaključke. Strojno učenje smatra se kao podskup umjetne inteligencije (eng. Artificial intelligence). Algoritmi strojnog učenja grade matematički model koji se temelji na uzorku podataka, poznatim kao "skup podataka za vježbu", kako bi se predvidjela ili donijela odluka bez da su izričito programirana za izvođenje zadatka. Algoritmi strojnog učenja koriste se u raznim aplikacijama, kao što je filtriranje e-pošte i računalni vid, gdje je nemoguće razviti algoritam specifičnih uputa za obavljanje zadatka. Strojno učenje usko je povezano s računalnom statistikom, koja se usredotočuje na predviđanja pomoću računala. Proučavanje matematičke optimizacije donosi metode, teoriju i područja primjene u polje strojnog učenja. Rudarstvo podataka (eng. data mining) je područje proučavanja u strojnom učenju i usredotočuje se na analizu istraživačkih podataka kroz učenje bez nadzora. U svojoj primjeni u svim poslovnim problemima, strojno učenje se također naziva prediktivna analitika.

# Zadaci strojnog učenja

Zadaci strojnog učenja klasificiraju se u nekoliko širokih kategorija. U učenju pod nadzorom (eng. Supervised learning), algoritam gradi matematički model iz skupa podataka koji sadrži i ulaze i željene izlaze. Na primjer, ako je zadatak utvrditi da li neka slika sadržavala određeni objekt, podaci o treningu za algoritam učenja pod nadzorom će uključivati slike sa i bez tog objekta, a svaka će slika imati oznaku (izlaz) koja označava da li sadržavala je predmet. U posebnim slučajevima, unos može biti samo djelomično dostupan ili ograničen na posebne povratne informacije. Algoritmi učenje pod djelomičnim nadzorom (eng. Semi-supervised learning) razvijaju matematičke modele iz nepotpunih podataka o vježbanju, gdje dio ulaza uzorka nema oznake.

## Učenja pod nadzorom

Algoritmi klasifikacije (eng. Classification) i regresijski algoritmi su vrste učenja pod nadzorom. Algoritmi klasifikacije koriste se kada su izlazi ograničeni na ograničeni skup vrijednosti. Za algoritam klasifikacije koji filtrira e-poštu, ulaz bi bio dolazna e-pošta, a izlaz bi bio naziv mape u koju će se poslati e-pošta. Za algoritam koji identificira neželjenu e-poštu, izlaz bi bio predviđanje bilo "spam" ili "nije spam", što je predstavljeno logičkim vrijednostima istina i laž. Regresijski algoritmi nazvani su za njihove kontinuirane izlaze, što znači da mogu imati bilo koju vrijednost unutar raspona. Primjeri kontinuirane vrijednosti su temperatura, duljina ili cijena objekta.

## Učenje bez nadzora

Kos učenja bez nadzora (eng. Unsupervised learning) algoritam gradi matematički model iz skupa podataka koji sadrži samo ulaze i nema željenih izlaznih oznaka. Algoritam učenja bez nadzora koristi se za pronalaženje strukture u podacima, kao što je grupiranje ili grupiranje podatkovnih točaka.

## Učenje bez nadzora

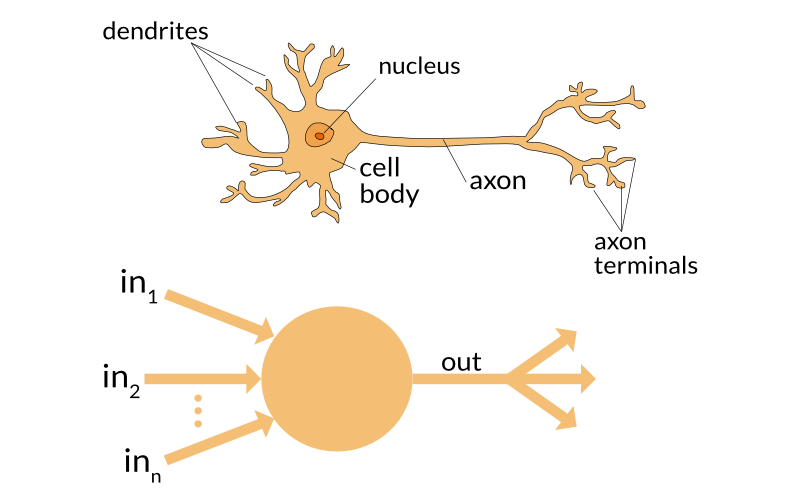
Učenje bez nadzora može otkriti obrasce u podacima i može grupirati ulaze u kategorije, kao u učenju značajki. Smanjenje dimenzionalnosti je proces smanjenja broja "osobina" ili ulaznih podataka u skupu podataka.

# Duboko učenje

Duboko učenje (eng. Deep learning), poznato i kao duboko strukturirano učenje ili hijerarhijsko učenje, dio je šire obitelji metoda neuronskih mreža koje se temelje na umjetnim neuronskim mrežama (eng. Neural network). Učenje se može nadzirati, polu-nadzirati ili bez nadzora.

Arhitekture za dubinsko učenje kao što su duboke neuronske mreže (eng. Deep neural network), duboke mreže vjerovanja (eng. Deep belief network), rekurentne neuronske mreže (eng. Recurrent neural network) i konvolucijske neuronske mreže (eng. Convolutional neural network) primijenjene su na područjima, uključujući računalni vid, prepoznavanje govora, obradu teksta, prepoznavanje zvuka, filtriranje društvene mreže, strojno prevođenje, bioinformatiku, dizajn lijekova, analiza medicinskih slika, inspekcija materijala i društvenih igara, u kojima su su proizveli rezultate usporedive i u nekim slučajevima bolje od ljudskih stručnjaka.

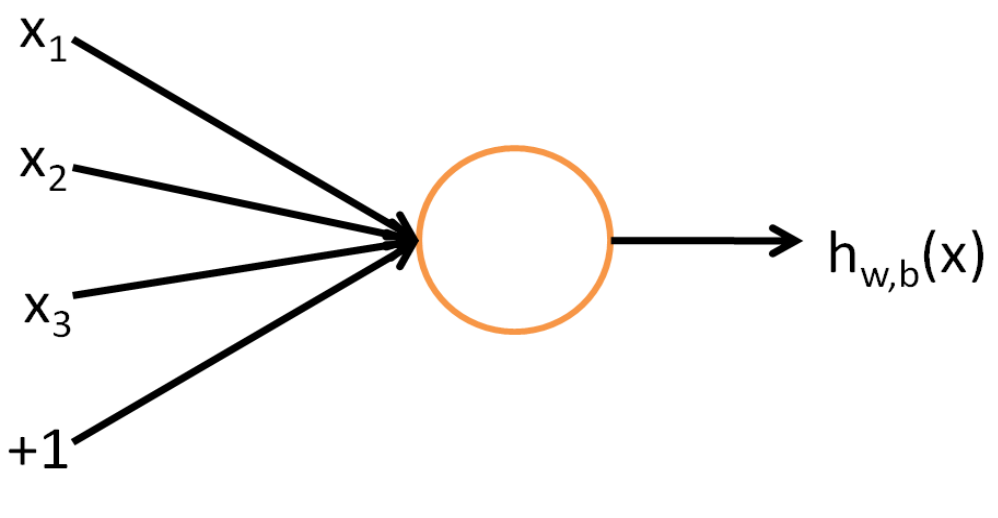
# Umjetne neuronske mreže

Umjetna neuronska mreža (eng. Artificial Neural Network) je algoritam strojnog učenja inspiriran računalnim modelom mozga i bioloških neuronskih mreža. Ukratko, umjetna neuronska mreža je računalna reprezentacija ljudske neuronske mreže (Slika 1) koja regulira ljudsku inteligenciju, rasuđivanje i memoriju

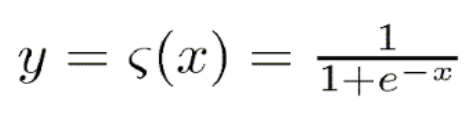
Slika 1 Neurosnke mreže

Glavni razlog za korištenje neuronskih mreža je da su neuronske mreže učinkovite u kompleksu računanja i hijerarhijskog predstavljanje znanja. Neuroni povezani aksonima i dendritima u složene neuronske mreže mogu proći i razmijeniti informacije, pohraniti rezultate računanja, proizvesti apstrakcije i provesti proces učenja u više koraka. Računski model takvog sustava može vrlo učinkovito simulirati proces učenja slično kao i u biološkim sustavu.

Najjednostavnija moguća neuronska mreža sastoji se od jednog neurona (Slika 2). Koristeći biološku analogiju, neuron je računska jedinica koja uzima ulaze preko dendritija kao električne ulaze i prenosi ih preko aksona na sljedeći sloj ili izlaz mreže.

U terminima strojnog učenja, gore prikazana mreža (Slika 2) ima jedan ulazni sloj, jedan skriveni sloj (taj se sastoji od jednog neurona) i jedan izlazni sloj. Proces učenja ove mreže je implementiran na sljedeći način. Ulazni sloj uzima ulazne značajke (npr. Piksele) za svaki uzorak treninga i stavlja kao argument u aktivacijsku funkciju koja izračunava hipotezu u skrivenom sloju. Aktivacijska funkcija (Slika 3) je normalno logistička regresija koja se koristi za klasifikaciju, međutim, druge alternative također su moguće. U gore opisanom slučaju, naš pojedinačni neuron odgovara točno ulaznom izlazu mapiranje koje je definirano logističkom regresijom.

Slika 2 Jednostavna neuronska mreža



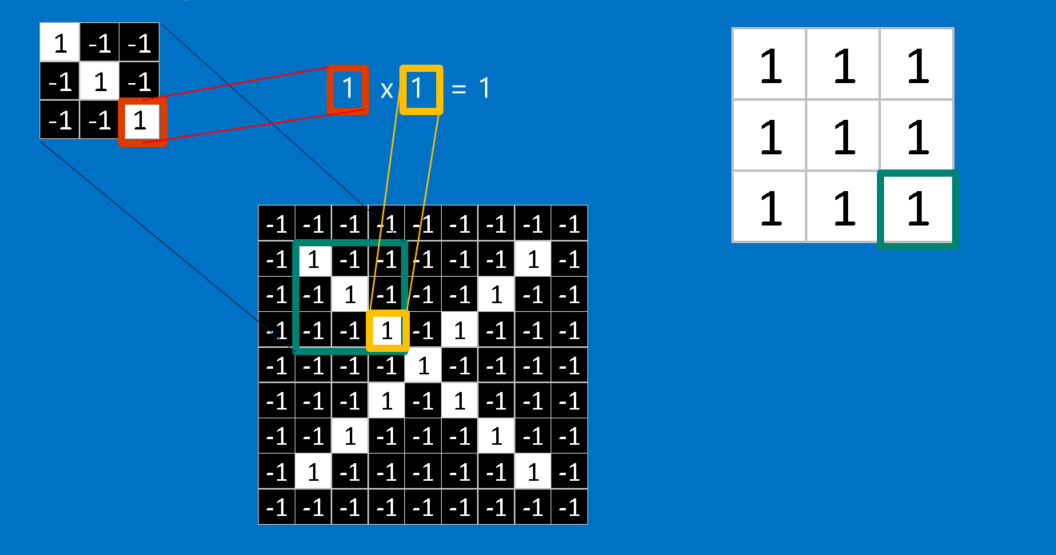
Slika 3 Aktivacijska funkcija

# Konvolucijske neuronske mreže

Konvolucijske neuronske mreže (CNN) analogne su tradicionalnim umjetnim neuronskim mrežama po tome što se sastoje od neurona koji se optimiziraju kroz trening. Svaki neuron će i dalje primati ulaz i izvoditi operaciju, što je temelj neuronskih mreža. Od ulaznih sirovih vektora slike do konačnog izlaza rezultata klase, cijela mreža će i dalje izražavati jednu perceptivnu rezultantnu funkciju (težinu). Zadnji sloj će sadržavati funkcije gubitka povezane s klasama. Jedina značajna razlika između CNN-ova i tradicionalnih neuronskih mreža jest da se CNN-ovi primarno koriste u području prepoznavanja uzoraka unutar slika. To nam omogućuje da u arhitekturu kodiramo značajke specifične za slike, čineći mrežu prikladnijom za zadatke usmjerene na sliku, dok istovremeno smanjujemo parametre potrebne za postavljanje modela.

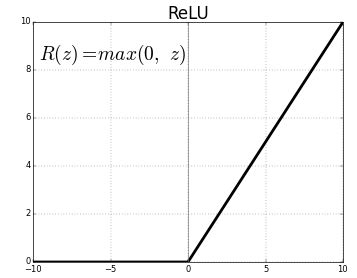
Zasićenje je u osnovi kada mreža nije u stanju učinkovito učiti zbog niza razloga. To je važan koncept većine, ako ne i svih algoritama strojnog učenja i važno je da se poduzmu sve mjere opreza kako bi se smanjili njegovi učinci. Ako bi naši modeli pokazivali znakove zasićenja, mogli bismo vidjeti smanjenu sposobnost određivanja generaliziranih značajki ne samo za našem dataset-u za trening, nego i za naš dataset za testiranje.

Prvi sloj u CNN-u je uvijek konvolucijski sloj. Konvolucijski sloj se sastoji od filtera (Slika 4) određene kvadratne veličine (npr. 5x5). Filter klizi po svim područjima ulazne slike. Područje po kojem filter prođe zove se receptivno polje. Filtar je također niz brojeva (eng. weights). Vrlo važna napomena je da dubina ovog filtra mora biti jednaka dubini ulaza kako bi matematika funkcionirala. Dok filter klizi po ulaznoj slici množe se vrijednosti u filtru s izvornim vrijednostima slike. Sva ta množenja sažeta u jedan broj koji predstavlja samo jedan položaj filtara na slici. Nakon toga se postupak ponavlja za svaku lokaciju na ulaznoj slici. Svaka jedinstvena lokacija na ulaznoj slici će proizvesti po jedan broj. Nakon što pomaknemo filtar preko svih lokacija dobivamo skup brojeva koje nazivamo mapom za aktivaciju ili mapom značajki. Svaki od ovih filtara može se smatrati identifikatorima značajki, kao naprimjer detektiranje poput ravnih rubova, jednostavnih boja i krivulja.



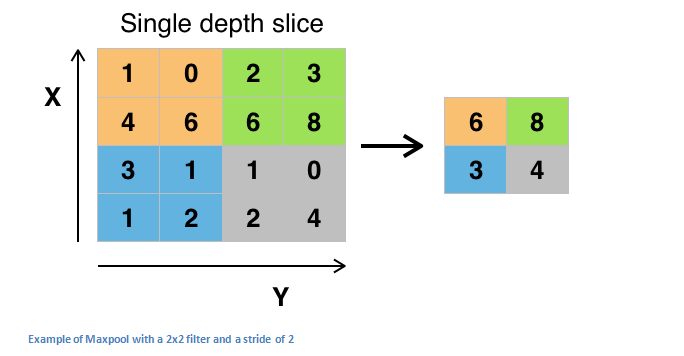
Slika 4 Primjer CNN filtera

Nakon svakog konvolucijskog sloja, konvencija je primjena nelinearnog sloja (ili aktivacijskog sloja). Svrha ovog sloja je uvesti nelinearnost u sustav u kojem u osnovi izračunavamo linearne operacije tijekom slojeva. U prošlosti su se koristile nelinearne funkcije kao što su tanh i sigmoid, ali se zaključilo da ReLU (Slika 5) funkcija radi puno bolje jer mreža može trenirati puno brže zbog računalne učinkovitosti bez značajne razlike u točnosti.



Slika 5 ReLU aktivacijska funkcija

Nakon ReLU slojeva dolazi pooling sloj. Također se naziva i sloj za downsampling. U ovoj kategoriji postoji nekoliko različitih slojeva pri čemu je najpopularniji maxpooling (Slika 6). To u osnovi uzima filtar, obično veličine 2x2 te ga primjenjuje na ulaz i ispisuje maksimalni broj u svakoj podregiji oko koje se filtar okreće. Intuitivno obrazloženje ovog sloja je da jednom kada znamo da je određena značajka u izvornom ulazu, bit će visoka vrijednost aktivacije, njezina točna lokacija nije toliko važna kao njezina relativna lokacija prema drugim značajkama. Ovaj sloj drastično smanjuje prostornu dimenziju ulaza. To ima dvije glavne svrhe. Prvi je da se količina parametara ili težina smanji, čime se smanjuje trošak računanja. Drugi je da će kontrolirati zasićenje. Ovaj se pojam odnosi na to kada je model tako prilagođen primjerima treninga da nije u mogućnosti generalizirati dobro za skupove provjere valjanosti i ispitivanja.

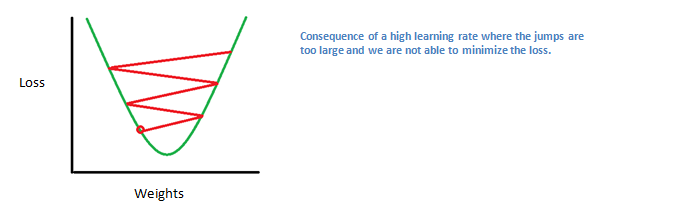


Slika 6 Max pooling

Način na koji računalo može prilagoditi svoje vrijednosti filtra je kroz proces koji se naziva backpropagation. Tako se širenje unatrag može podijeliti na 4 odvojena dijela, prolaz prema naprijed, funkciju gubitka, prolaz unatrag i ažuriranje težina (eng. weights). Tijekom prolaska prema naprijed, uzimamo sliku za trening i provedemo je kroz cijelu mrežu. Funkcija gubitka može se definirati na mnogo različitih načina, ali uobičajena je MSE (srednja kvadratna pogreška, Slika 7). Želimo doći do točke u kojoj je predviđena oznaka, izlaz mreže, ista kao i oznaka za obuku što znači da je naša mreža točno predvidjela, te trebamo optimizirati težine u mreži tako da matematičkim izračunom odredimo za koje bi vrijednosti naš ulaz dao traženi izlaz. Nakon toga prolazimo unatrag kroz mrežu koja određuje koji su utezi najviše pridonijeli gubitku i pronalazimo način za njihovo prilagođavanje kako bi se gubitak smanjio. Nakon što izračunamo ovaj derivat prelazimo na posljednji korak, odnosno ažuriranje težina. U ovom koraku uzimamo sve težine filtera i ažuriramo ih tako da se mijenjaju u suprotnom smjeru od gradijenta.



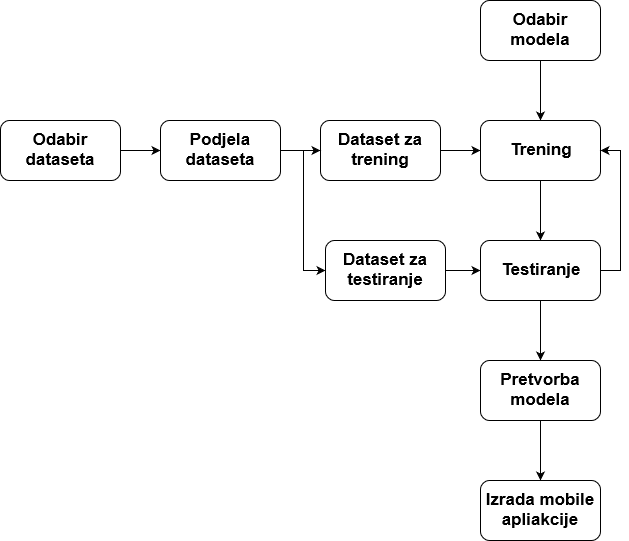
Slika 7 Srednja kvadratna pogreška



Slika 8 Traženje minimuma funkcije

# Postupci izrade projekta

Izrada projekta se sastoji od nekoliko koraka (Slika 9). Prvi dio projekta se izrađuje na računalu, a onda se u drugom dijelu sve iz prvog dijela prenosi na pametni telefon. Prvi korak se sastoji od odabira skupa podataka (eng. dataset) koji ćemo koristiti za trening modela koji će nam dati koordinate ključnih dijelova tijela za zadanu sliku. Drugi korak je odabir modela koji ćemo koristiti i trenirati. Nakon toga u trećem koraku treba postaviti okolinu za trening na računalu te istrenirati model. U idućem, četvrtom koraku treba istrenirani model pretvoriti u model prikladan za mobilnu upotrebu. Na kraju, u zadnjem koraku izrađuje se aplikacija za pametni telefon koja će koristiti istrenirani model i obavljati svoj zadatak.



Slika 9 Dijagram izrade projekta

## Odabir dataset-a

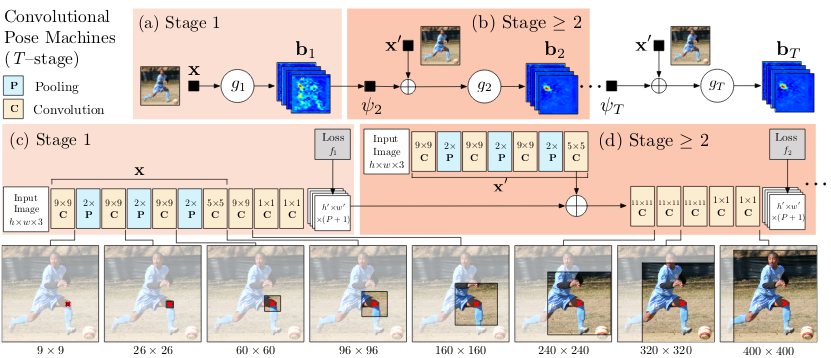
Trebamo odabrati dataset koji pogodan za naše potrebe, a to je velik skup slika te za svaku sliku su označeni bitni dijelovi ljudskog tijela koji će nam pomoći da odredimo poziciju čovjeka na slici (Slika 10). Dataset nisam sam radio zato što na trening je potrebno par desetka tisuća označenih slika te za što je potrebno jako puno vremena i teško izvedivo za jednu osobu. Na internetu postoje već gotovi dataset-ovi za naše potrebe i slobodni su za upotrebu. Ja sam se služio COCO dataset-om koji ima preko 200,000 slika gdje su notirane pozicije čovjeka na svakoj slici, ali se nisam služio cijelim dataset-om. S njihove [stranice](http://cocodataset.org/#download) se može skinuti dataset. Nakon toga treba dataset podijeli na dva dijela u omjeru 19:1, u prvom većem dijelu se nalaze podaci na kojem će se model trenirati, a u drugom dijelu se nalaze slike na kojem će se testirati napredak modela.



Slika 10 Human pose estimation

## Odabir modela

U ovom koraku biramo model kojeg ćemo trenirati i koristiti na kraju za određivanje pozicije čovjeka. Odabrao sam model CPM (Convolutional Pose Machines, Slika 11) zato što je taj model napravljen za mobilnu upotrebu kod koje je sirova snaga hardvera slabija naspram one na računalima tako da nam treba neki jednostavniji model koji će rezultat izračunati brže, ali zato nepreciznije. CPM se sastoji od niza prediktora osposobljenih za gusto predviđanje na svakoj lokaciji slike tj. generiranje karte vjerovanja (eng. belief map) za svaki dio tijela.



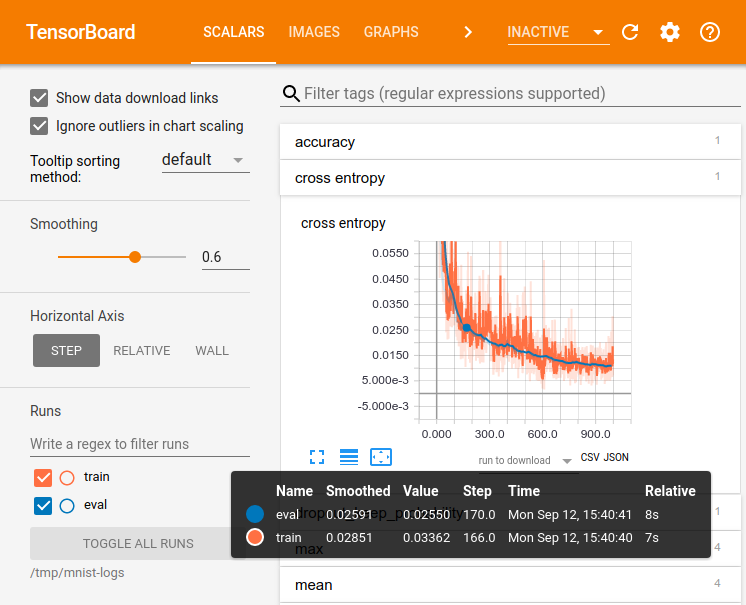
Slika 11 Convolutional Pose Machines

## Treniranje modela

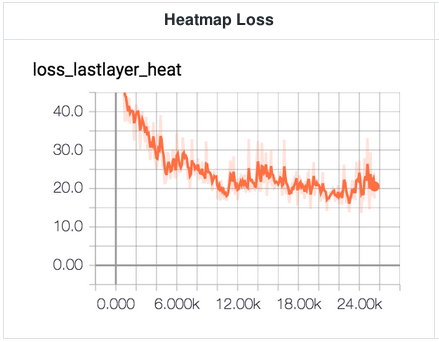
Treniranje se izvodi pomoću aplikacije TensorFlow. TensorFlow je sučelje za izražavanje algoritama za strojno učenje i implementaciju za izvođenje takvih algoritama. Izračun izražen pomoću TensorFlowa može se izvršiti s malom ili nikakvom promjenom na širokom rasponu heterogenih sustava, od mobilnih uređaja kao što su pametni telefoni i tableti do velikih distribuiranih sustava stotina strojeva i tisuća računalnih uređaja kao što su grafičke kartice, što nam omogućuje da se model istrenira na računalu na grafičkim karticama te se nakon toga prenese na pametni telefon za korištenje. Sustav je fleksibilan i može se koristiti za izražavanje raznih algoritama, uključujući algoritme za obuku i zaključivanje za modele dubokih neuronskih mreža, a koristi se za provođenje istraživanja i za uvođenje sustava strojnog učenja u proizvodnju u više od deset područja računalne znanosti i drugih područja, uključujući prepoznavanje govora, računalni vid, robotiku, pronalaženje informacija, obradu prirodnog jezika, izdvajanje geografskih informacija i računalno otkrivanje lijekova.

Za trening je potrebno još instalirati upravljački program za grafičku karticu kako bi mogli trenirati model na grafičkoj kartici jer se trening brže obavi na grafičkoj kartici zato što ima više računske snage nego procesor. Zatim treba instalirati OpenCV koji je library za obradu slike te ćemo koristiti za smanjenje slika na manju rezoluciju od onih za koje su nam zadane.

Za trening sam koristio već predefiniranu skriptu u programskom jeziku Python za treniranje ovakvog tipa modela gdje sam samo zamijenio putanju do modela i promijenio tako da koristi moj dataset kojeg sam ranije pripremio Prije korištenja svake slika kod treninga se treba rezolucija slike smanjiti od one definirane na 192x192 jer se sa smanjenjem rezolucije poboljšavaju performanse, ali se smanjuje preciznost programa. Rezoluciju 192x192 sam izabrao tako da sam probao više različitih rezolucija te odredio da je ta rezolucija najbolja po omjeru performansa i preciznosti. Konfiguraciju sam koristio zadanu od CPM modela gdje sam jedno promijenio veličinu ulazne slike i putanje da potrebnih datoteka.

Proces treniranje traje preko 12h za jedno cijelo treniranje. Developeri TensorFlow-a su napravili program TensorBoard (Slika 12) gdje je moguće pratiti proces treniranja. Program automatski dokumentira proces treniranja i crta grafove koju su od velike pomoći jer pomoću njih možemo vidjeti da li trening ide u pravom smjeru i kako napreduje u vremenu. Također možemo vidjeti kada možemo završiti trening jer se graf (Slika 13) na kraju praktički stabilizirao, a nije dobro previše trenirati jer bi moglo doći do zasićenja što znači da se je model naučio napamet za svaku sliku koju smo joj dali odgovore što nije dobro.

Slika 12 TensorBoard



Slika 13 Graf treninga

## Pretvorba modela pogodna za mobilnu upotrebu

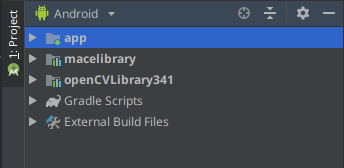
Developeri TensorFlow-a su napravili program koji optimizira modele za mobilnu upotrebu kako bi se povećale performanse na uređajima slabije kompjuterske snage uz što manji kompromis na preciznost. Program radi tako da se iz modela prvo izbace svi dijelovi koji služe za trening i praćenje napretka te nam nisu bitni za rad aplikacije, ali su ponekad korisni ako želimo tijekom rada aplikacije pratiti da li dolazi do nekih anomalija. Program još pretvara model tako da koristi neke zamjenske funkcije za izračun koje su manje precizne, ali su jeftinije za izračunat. Da bi pretvorili model u model pogodan za mobilnu upotrebu graf model treba prvo zamrznuti pomoću njihovog alata zvanog „freez\_graph” kojemu kao argumente damo ime modela kojeg želimo zamrznut, ime datoteke u kojoj će se nalaziti zamrznuti graf, ime datoteke koja sadrži podatke o istreniranom modelu te ime izlaznih točaka modela. Nakon toga se model može transformirati pomoću alata kojem dajemo ime ulaznog zamrznutog modela, ime izlaznog modela, ime ulaznih točaka, ime izlaznih točaka i konfiguraciju kako da alat transformira graf. Na kraju treba još graf pretvoriti u TFLITE verziju grafa da bi ga mogli pokretati na TensorflowLite koja je inačica Tensorflow-a ali za mobilne uređaje s manje kompjuterske snage koju ćemo mi koristiti na pametnom telefonu.

## Izrada mobilne aplikacije

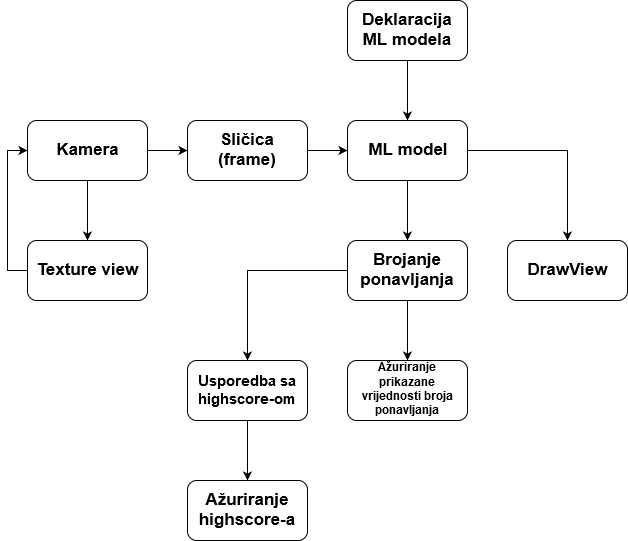
Da bi koristili graf koji smo istrenirali i trebamo kreirati aplikaciju za mobilni telefon koja će pokretati graf. Aplikaciju sam napravio za Android operacijski sustav u programskom okruženju Android Studio. Android Studio je službeno integrirano razvojno okruženje (eng. IDE) za Googleov operativni sustav Android, izgrađen na JetBrainsovom IntelliJ IDEA softveru i dizajniran posebno za Android razvoj. Uz osnovne Android Studio alate potrebno nam je još Android NDK. Android NDK je skup alata koji omogućuje implementaciju dijelova aplikacije u izvornom kodu, koristeći jezike kao što su C i C ++. On mi je pomogao da upotrijebim library koji je napisan u tom jeziku, a to je OpenCV kojeg ćemo opet koristiti za smanjenje rezolucije slike i Mace library koji nam omogućuje da koristimo GPU na pametnom telefonu radi boljih performansi.

# Kod

Projekt se sastoji od tri dijela (Slika 14). Prvi je moj kod koji je nalazi u direktoriju „app”. Drugi dio su library-i kojima se služim u aplikaciji a to su već spomenuti Mace i OpenCV. Library-esam dodao u projekt tako da sam ih skinuo s interneta i stavio u korijenski direktorij projekta. Library-i su već prekonfigurirani te ih Android Studio sam prepoznaje i sinkronizira s postavkama našeg projekta tako da odgovaraju. Zadnji dio su skripte za kompajliranje koje određuju Android Studio koje alate da koristi i koje verzije za kompajliranje zadanih datoteka. Svaki dio projekta ima svoje skripte za kompajliranje koje se na kraju povezuju jednom skriptom. U mojem projektu se nalaze dvije vrste skripti zato što se služim dvama različitim programskim jezicima, jedan od njih je Java te drugi C++. Za Javu se koriste Gradle skripte. Svaki Gradle predstavlja jedan ili više projekata. Projekt predstavlja JAR biblioteku ili web-aplikaciju ili može predstavljati ZIP koji je sastavljen od više JAR-a proizvedenih u drugim projektima. Te se projekt sastoji od različitih zadataka koje određuju kako će se kompajliranje izvodi. Gradle koristi Groovy programski jezik za pisanje skripti. Za C++ se koristi CMake skripte koje služe za kompajliranje C++ dijela projekta.



Slika 14 Stablo projekta



Slika 15 Dijagram projekta

## Manifest

Svaka aplikacija se sastoji od manifest datoteke koja se nalazi u posebnom direktoriju koja se zove **manifests**. Manifest datoteka za Android aplikaciju je resursna datoteka koja sadrži sve detalje potrebne za android sustav o aplikaciji. To je ključna datoteka koja radi kao most između android developera i android platforme. Pomaže programeru da prenese funkcionalnost i zahtjeve aplikacije na Android. Datoteka koja mora biti imenovana kao **AndroidManifest.xml** i smještena u korijen aplikacije. Manifest nam omogućuje definiranje paketa, API-ja, library-a potrebnih za aplikaciju. Također u njoj definiramo osnovni elemente aplikacije kao što su aktivnosti, usluge i sl. Ako koristimo neke određene resurse moramo imati dopuštenja za njih, a ta dopuštenja definiramo u manifest datoteci.

|  |
| --- |
| #app/manifests/AndroidManifest.xml  <manifest **xmlns:**android="http://schemas.android.com/apk/res/android"  package="com.example">  <uses-permission **android:**name="android.permission.CAMERA"/>  <uses-feature **android:**name="android.hardware.camera"/>  <uses-feature **android:**name="android.hardware.camera.autofocus"/>  <uses-sdk **android:**minSdkVersion="21"/>  <application  **android:**allowBackup="true"  **android:**icon="@drawable/ic\_launcher"  **android:**label="Pose estimation"  **android:**theme="@style/MaterialTheme">  <activity  **android:**name=".CameraActivity"  **android:**label="Pose estimation"  **android:**screenOrientation="portrait">  </activity>  </application>  </manifest> |

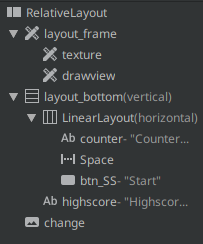
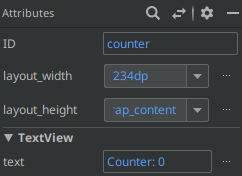
**<uses-permission>** je tag kojem opisujemo koje ovlasti treba aplikacija da bi se mogla koristiti, a u mojoj aplikaciji je potrebna kamera i tome pristupamo tako da kao ime ovlasti damo „android.permission.CAMERA”. Ako želimo još neke određene značajke neke ovlasti koristimo **<uses-feature>** tag. Za aplikaciju nam je potrebno hardverski pristupiti kameri i njezinim kontrolama („android.hardware.camera”), te značajki da automatski fokusira sliku („android.hardware.camera.autofocus”).

U manifestu moramo definirati minimalnu verziju SDK-a koju pametni mobitel može imati na sebi da bi mogao koristiti aplikaciju. U našem slučaju je to verzija 21, te se postavlja s tagom <uses-sdk> i argumentom „android:minSdkVersion="21"”.

Pod tagom **<application>** nalaze se komponente o aplikaciji. Postavivši varijablu allowBackup na true dopuštamo da se kreiraju sigurnosne kopije aplikacije. Ime aplikacije postavljamo u varijablu label. Ikona i tema su definirani pod resursima aplikacije te zato su samo definirane putanje do resursa.  
Pod tagom **<activity>** je definiran zadani activity koji će se pokrenuti kada se pokrene aplikacija. Definiramo mu ime varijablom name i zadanu orijentaciju ekrana a to je portret varijablom screenOrientation.

## Grafičko sučelje

U datoteci **fragment\_camera2\_basic.xml** nalazi se grafičko sučelje aplikacije. Datoteka je xml oblika ali su zato developeri napravili Layout editor koji nam omogućuje da s lakoćom napravimo grafičko sučelje aplikacije. Layout Editor Android Studia omogućuje nam da izradimo izgled aplikacije tako da povučemo komponente na zaslon umjesto da ručno pišemo izgled xml-a.

Grafičko sučelje aplikacije je jednostavno i sastoji se od tri dijela (Slika 16, Slika 18). Prvi dio su view-ovi, tu se nalaze dva od kojih je jedan **texture** pomoću kojeg prikazujemo kameru, a drugi je **drawview** pomoću kojeg crtamo ključne točke čovjekove pozicije. Drugi dio su gumbi i to su dva gumba, prvi je za promjenu kamere između prednje i stražnje, a drugi gumb je Start/Stop gumb za započinjanje ili završavanje vježbe. Zadnji dio su informacijski tekstovi koji nam prikazuju koliko smo ponavljanja vježbe napravili, a drugi koji nam je highscore. Svaki element ima svoj id (Slika 17) pomoću kojeg se prepoznaje u programskom dijelu aplikacije.

Slika 16 Stablo elemenata

Slika 17 Atributi elementa

Slika 18 Grafičko sučelje aplikacije

## CameraActivity

**CameraActivity** je prvi aktiviti koji će se pokrenuti jer smo ga definirali u manifestu. Cijeli program će ići preko toga jedinog aktivitija. Na početku datoteke se nalaze, kao i u svakom programskom jeziku, par linija u kojima učitavamo library-e koje ćemo koristiti. U ovom slučaju su to osnovi library-i Androida te OpenCV.

Tu definiramo javnu klasu **CameraActivity** koja nasljeđuje Activity klasu. Odmah na početku i učitavamo library od OpenCV-a te definiramo javnu statičku boolean varijablu **isOpenCVInit** u kojoj će se nalaziti true ako je OpenCV učitan i pokrenut ili false ako nije.

Ispod toga kreiramo callback funkciju za učitavanje OpenCV library-a. Ispod toga provjeravamo vrijednost varijable status u kojoj se nazali status da li je library uspješno učitan ili nije. To provjeravamo switch-em te u slučaju da je library uspješno učitan postavljamo **isOpenCVInit** u true, ako nije onda šaljemo callback roditelju klase da riješi konflikt.

|  |
| --- |
| #app/java/CameraActivity.java:**16**  **public** **class** **CameraActivity** **extends** Activity {  **static** {  System.loadLibrary("opencv\_java3");  }  **public** **static** **boolean** isOpenCVInit = **false**;  **private** BaseLoaderCallback mLoaderCallback = **new** BaseLoaderCallback(**this**) {  **@Override**  **public** **void** **onManagerConnected**(**int** status) {  **switch** (status) {  **case** LoaderCallbackInterface.SUCCESS:  isOpenCVInit = **true**;  **break**;  **default**: {  **super**.onManagerConnected(status);  }  **break**;  }  }  }; |

**SharedPreferences** omogućuje nam spremanje i dohvaćanje podataka u obliku para ključa i vrijednosti. Te kreiramo instancu **sp** tako da bi mogli pristupiti kasnije.

Funkcija **onCreate** se poziva kada pokrenemo aplikaciju ispočetka ili kada je pokrenemo iz pozadine. U slučaju da je aplikacija pokrenuta ispočetka argument **savedInstanceState** će biti prazan tj. null, a u drugom slučaju će se u argumentu nalaziti spremljeno stanje aplikacije u kojem je bila kad smo izašli iz nje. Na početku funkcije dohvatimo spremljene podatke pod imenom Highscore i dodatnim argumentom **Activity.MODE\_PRIVATE** kojim kažemo da tim podacima ne može pristupiti nitko izvan aplikacije. Ispod toga se poziva roditeljska funkcija da učita zadane postavke. Ako se aplikacije pokreće ispočetka, što provjeravamo da li je argument **savedInstanceState** null, onda zadani kontejner zamjenjujemo s **Camera2BasicFragment**-om u kojoj se nalazi sučelje aplikacije.

|  |
| --- |
| #app/java/CameraActivity.java:**47**  **public** **static** SharedPreferences sp;  **@Override**  **protected** **void** **onCreate**(Bundle savedInstanceState) {  sp = getSharedPreferences("Highscore", Activity.MODE\_PRIVATE);  **super**.onCreate(savedInstanceState);  **if** (**null** == savedInstanceState) {  getFragmentManager()  .beginTransaction()  .replace(R.id.container, Camera2BasicFragment.newInstance())  .commit();  }  } |

Funkcija **onResume** se pozove kada na primjer zaključamo pametni telefon i otključamo ga. Aplikacije se mora nastaviti. Na početku se pozove ista funkcija od roditeljske klase. Ponovno treba učitati library OpenCV. Prvo pokušamo sinkrono učitati library ako je još pokrenut i ako uspijemo onda samo javimo programu da smo uspjeli, ako ne onda asinkrono kažemo da se učita i kada završi neka pozove callback funkciju koju smo prije definirali.

|  |
| --- |
| #app/java/CameraActivity.java:**62**  **@Override**  **protected** **void** **onResume**() {  **super**.onResume();  **if** (!OpenCVLoader.initDebug()) {  OpenCVLoader.initAsync(OpenCVLoader.OPENCV\_VERSION, **this**, mLoaderCallback);  } **else** {  mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);  }  } |

Funkcija **onDestroy** se poziva kada se aplikacija ugasi. U funkciji se poziva ista roditeljska funkcija.

|  |
| --- |
| #app/java/CameraActivity.java:**72**  **@Override**  **protected** **void** **onDestroy**() {  **super**.onDestroy();  } |

**Camera2BasicFrament** je najbitnija datoteka projekta u kojem se nalazi većinski dio koda projekta. Fragment predstavlja ponašanje ili dio korisničkog sučelja u **FragmentActivity**. Može se kombinirati više fragmenata u jednoj aktivnosti kako bi izgradili višestruko korisničko sučelje i ponovno iskoristili fragment u više aktivnosti. Fragment se može zamisliti kao modularni dio aktivnosti, koji ima svoj životni ciklus, prima vlastite ulazne događaje, a koje možete dodati ili ukloniti dok se aktivnost izvodi. U projektu postoji samo jedan fragment **Camera2BasicFragment** koja nasljeđuje osnovnu klasu Fragment. Na početku deklariramo neke varijable koje ćemo koristiti u programu, a to su stringovi, objekti u grafičkom sučelju, klasifikator te tajmeri.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**70**  **public** **class** **Camera2BasicFragment** **extends** Fragment  **implements** FragmentCompat.OnRequestPermissionsResultCallback {  **private** **static** **final** String TAG = "PoseEstimationDemo";  **private** **static** **final** String FRAGMENT\_DIALOG = "dialog";  **private** **static** **final** String HANDLE\_THREAD\_NAME = "CameraBackground";  **private** **static** **final** **int** PERMISSIONS\_REQUEST\_CODE = **1**;  **private** **final** Object lock = **new** Object();  **private** **boolean** runClassifier = **false**;  **private** **boolean** checkedPermissions = **false**;  **private** TextView counterView;  **private** TextView highscoreView;  **private** Button StartStopBtn;  **private** ImageButton changeCameraBtn;  **private** AutoFitTextureView textureView;  **private** AutoFitFrameLayout layout\_frame;  **private** DrawView drawView;  **private** PoseEstimation classifier;  **private** **boolean** counterRunning;  **private** **boolean** timerRunning;  **private** **int** highscore; |

## Camera2BasicFrament

Ovdje definiramo osnovne callback funkcije, postoje četiri. **OnSurfaceTextureAvailable** se poziva kada se učitaju sve potrebne stvari i površina postane dostupna za korištenje. Funkcija se poziva s dva argumenta: visina i širina površine koju možemo koristiti i samo prosljeđujemo argumente funkciji openCamera, koja će otvoriti pregled kamere. **OnSurfaceTextureSizeChanged** se poziva ako se promijeni veličina površine te poziva se funkcija **configureTransfrom** koja promijeni veličinu pregleda kamere da se bude ista kao i površina. **OnSurfaceTextureDestroyed** i **onSurfaceTextrueUpdate** ne koristimo jer će se kasnije u kodu pobrinuti za to.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**116**  **private** **final** TextureView.SurfaceTextureListener surfaceTextureListener =  **new** TextureView.SurfaceTextureListener() {  **@Override**  **public** **void** **onSurfaceTextureAvailable**(SurfaceTexture texture, **int** width, **int** height) {  openCamera(width, height);  }  **@Override**  **public** **void** **onSurfaceTextureSizeChanged**(SurfaceTexture texture, **int** width, **int** height) {  configureTransform(width, height);  }  **@Override**  **public** **boolean** **onSurfaceTextureDestroyed**(SurfaceTexture texture) {  **return** **true**;  }  **@Override**  **public** **void** **onSurfaceTextureUpdated**(SurfaceTexture texture) {  }  }; |

U ovo dijelu koda se nalazi callback funkcije za kameru. Prva od njih je **onOpened** koja se pozove kada se aplikacija spoji s kamerom i u tom slučaju se otpušta **cameraOpenCloseLock** koji će biti objašnjen kasnije u kodu, sprema se trenutno korištena kamera i poziva se funkcija koja će kreirati pregled kamere. U funkciji **onDisconnected**, koja se poziva kada se aplikacija odspoji od kamere, otpuštamo **cameraOpenCloseLock** te gasimo trenutno korištenu kameru i resetiramo vrijednost varijable na null. Funkcija **onError** se poziva ako dođe do greške tijekom spajanja na kameru ili tijekom korištenja te u tom slučaju otpuštamo **cameraOpenCloseLock** i gasimo kameru, isto kao i u **onDisconnected** funkciji, te još na kraju provjeravamo ako se izvršava activity njega ugasimo.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**142**  **private** String cameraId = "0";  **private** CameraCaptureSession captureSession;  **private** CameraDevice cameraDevice;  **private** Size previewSize;  **private** **final** CameraDevice.StateCallback stateCallback =  **new** CameraDevice.StateCallback() {  **@Override**  **public** **void** **onOpened**(**@NonNull** CameraDevice currentCameraDevice) {  cameraOpenCloseLock.release();  cameraDevice = currentCameraDevice;  createCameraPreviewSession();  }  **@Override**  **public** **void** **onDisconnected**(**@NonNull** CameraDevice currentCameraDevice) {  cameraOpenCloseLock.release();  currentCameraDevice.close();  cameraDevice = **null**;  }  **@Override**  **public** **void** **onError**(**@NonNull** CameraDevice currentCameraDevice, **int** error) {  cameraOpenCloseLock.release();  currentCameraDevice.close();  cameraDevice = **null**;  Activity activity = getActivity();  **if** (**null** != activity) {  activity.finish();  }  }  }; |

Funkcija **showToast** služi da bi prikazala broj ponavljanja koje su napravljene. Funkcija se poziva a argumentom string text koja sadrži broj ponavljanja u string obliku. Tada se pristupa trenutnom activity-u te se na njoj pokreće funkcija **run** na dretvi grafičkog sučelja koja ažurira string koji se prikazuje korisniku.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**248**  **private** **void** **showToast**(**final** String text) {  **final** Activity activity = getActivity();  **if** (activity != **null**) {  activity.runOnUiThread(  **new** **Runnable**() {  **@Override**  **public** **void** **run**() {  counterView.setText("Counter: " + text);  drawView.invalidate();  }  });  }  } |

Aplikacija pri spajanju na kameru će sama odabrati najveću moguću rezoluciju za prikaz što je lijepo, ali zauzima više resursa. Funkcija **chooseOptimalSize** će odabrati najbolju rezolucija po zadanom omjeru. Na početku kreiramo dvije liste u kojima ćemo spremati moguće rezolucije. U prvoj listi će se nalaziti rezolucije koje su veće od površine na kojoj će se prikazivati, a u drugoj rezolucije koje su manje od površine prikaza. Iteriramo po svakoj opciji koja postoji te provjeravamo da li su visina i širina manje od maksimalne moguće i da li je u zadanom omjeru. Ako rezolucija zahtjeva trenutne uvijete onda provjeravamo da li je rezolucija veća ili manja od veličine površine na kojoj će se prikazivati, ako je veća onda se sprema u prvu listu, ako nije onda u drugu. Na kraju ako smo našli i jednu rezoluciju koja je veća od površine prikaza onda vraćamo onu rezoluciju koja je najmanja u listi, ako nismo onda provjeravamo dali smo našli ikakvu rezoluciju koja odgovara vraćamo najveću u listi koja postoji. Ako nismo našli ni jednu zadovoljavajuću rezoluciju ispisujemo log za našu informaciju i vraćamo prvu moguću.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**282**  **private** **static** Size **chooseOptimalSize**(  Size[] choices,  **int** textureViewWidth,  **int** textureViewHeight,  **int** maxWidth,  **int** maxHeight,  Size aspectRatio) {  List<Size> bigEnough = **new** ArrayList<>();  List<Size> notBigEnough = **new** ArrayList<>();  **int** w = aspectRatio.getWidth();  **int** h = aspectRatio.getHeight();  **for** (Size option : choices) {  **if** (option.getWidth() <= maxWidth  && option.getHeight() <= maxHeight  && option.getHeight() == option.getWidth() \* h / w) {  **if** (option.getWidth() >= textureViewWidth && option.getHeight() >= textureViewHeight) {  bigEnough.add(option);  } **else** {  notBigEnough.add(option);  }  }  }  **if** (bigEnough.size() > **0**) {  **return** Collections.min(bigEnough, **new** CompareSizesByArea());  } **else** **if** (notBigEnough.size() > **0**) {  **return** Collections.max(notBigEnough, **new** CompareSizesByArea());  } **else** {  Log.e(TAG, "Couldn't find any suitable preview size");  **return** choices[**0**];  }  } |

Funkcija **onViewCreated** se poziva kada se kreira view od **Camera2Basic** fragmenta. U toj funkciji uglavnom spajamo grafičko sučelje s programskim dijelom i kreiramo rukovoditelje događaja. Svaki element u grafičkom sučelju ima svoj ID te se preko njega spajamo na te elemente. Za highscore pristupamo **SharedPreferences**-u koji smo definirali u **CameraActivity**, te pristupamo varijabli tipa int pod ključem „highscore” i dodatno kao argument damo ako ta varijabla ne postoji neke vrati broj nula jer kod prvog pokretanja aplikacije ta varijabla neće postojati te je treba postaviti na nula. Nakon toga u grafičkom sučelju ažuriramo prikazanu informaciju za highscore.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**336**  **@Override**  **public** **void** **onViewCreated**(**final** View view, Bundle savedInstanceState) {  textureView = view.findViewById(R.id.texture);  counterView = view.findViewById(R.id.counter);  layout\_frame = view.findViewById(R.id.layout\_frame);  drawView = view.findViewById(R.id.drawview);  changeCameraBtn = view.findViewById(R.id.change);  StartStopBtn = view.findViewById(R.id.btn\_SS);  // High score  highscoreView = view.findViewById(R.id.highscore);  highscore = CameraActivity.sp.getInt("highscore", **0**);  highscoreView.setText("Highscore: " + String.valueOf(highscore)); |

U ovom dijelu funkcije se nalaze funkcije koje će se pozvati kada se pritisne određeni gumb u grafičkom sučelju. Prvi gumb je za promjenu kamere s prednje na zadnju ili obratno. Prvo od spojimo kameru tako da se pozove funkcija onPause koja se pobrine o tome, te se **cameraId** promijeni s „1” za prednju na „0” za stražnju ili obratno te se poziva funkcija **onResume** koja se ponovno spoji na kameru.

Drugi gumb je Start/Stop gumb za pokretanje ili zaustavljanje vježbe. Kada se pokrene vježba program odbrojava 3 sekunde prije nego što se pokrene vježba te zato na početku trebamo provjeriti da li traje odbrojavanje u slučaju da je gumb dva puta pritisnut. U slučaju da je gumb pritisnut i da vježba taje onda se zaustavlja vježba tako da se tekst na gumbu promijeni u „Start”. Boolean varijabla **counterRunning** se postavlja na false te provjeravamo da li je broj ponavljanja veći od highscore-a. U slučaju da je vrijednost highscore-a se postavlja na novu vrijednost te se sprema u **SharedPreferences** da bi bili spremljeni i pristupiti kada sljedeći put koristimo aplikaciju, te ažuriramo vrijednost koja se prikazuje u grafičkom sučelju. U slučaju da se vježba pokreče postavlja se tajmer koji će trajati 3000 ms i otkucat će svakih 1000ms i varijabla **timerRunning** se postavlja u true da bi mogli provjeriti da li tajmer traje. Svakih 1000ms ili 1s ažuriramo tekst na gumbu na kojem će odbrojavati vrijeme da bi korisnik znao da tajmer traje. Kada tajmer završi poziva se funkcija **onFinish** te se test na gumbu postavlja u „Stop”, **timerRunning** se postavlja u false i **counterRunning** u true i brojač ponavljanja u 0.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**354**  changeCameraBtn.setOnClickListener(**new** View.OnClickListener() {  **@Override**  **public** **void** **onClick**(View v) {  onPause();  cameraId = cameraId.equals("1") ? "0" : "1";  onResume();  }  });  StartStopBtn.setOnClickListener(**new** View.OnClickListener() {  **@Override**  **public** **void** **onClick**(View v) {  **if** (timerRunning)  **return**;  // Toggle between start/stop  **if** (counterRunning) {  StartStopBtn.setText("Start");  counterRunning = **false**;  **if** (counter > highscore) {  highscore = counter;  SharedPreferences.Editor editor = CameraActivity.sp.edit();  editor.putInt("highscore", highscore);  editor.apply();  highscoreView.setText("Highscore: " + String.valueOf(highscore));  }  } **else** {  // Create timer  **new** **CountDownTimer**(**3000**, **1000**) {  **public** **void** **onTick**(**long** millisUntilFinished) {  StartStopBtn.setText(String.valueOf(millisUntilFinished / **1000** + **1**));  }  **public** **void** **onFinish**() {  StartStopBtn.setText("Stop");  counterRunning = **true**;  timerRunning = **false**;  counter = **0**;  }  }.start();  timerRunning = **true**;  }  }  });  } |

Funkcija **onActivityCreated** se poziva kada se kreira activity. Prvo se poziva ista funkcija roditeljske klase te se kreira klasifikator za trenutni activity i ako **drawView** nije kreirane kreira se **drawView** većine koju klasifikator vraća. **DrawView** je view koji se nalazi preko prikaza kamere i na njoj će se ucrtavati informacije koje će vraćati klasifikator, a to su ključni dijelovi tijela.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**411**  **@Override**  **public** **void** **onActivityCreated**(Bundle savedInstanceState) {  **super**.onActivityCreated(savedInstanceState);  **try** {  classifier = **new** PoseEstimationFloatInception(getActivity());  **if** (drawView != **null**)  drawView.setImgSize(classifier.getImageSizeX(), classifier.getImageSizeY());  } **catch** (IOException e) {  Log.e(TAG, "Failed to initialize an image classifier.", e);  }  startBackgroundThread();  } |

Funkcija **onResume** se poziva kada se vratimo u aplikaciju dok je bila u pozadini. Prvo pozovemo istu funkciju roditeljske klase te pokrenemo pozadinsku dretvu funkcijom **startBackgroundThread**. Provjeravamo da li je **textureView** dostupan, ako je onda otvaramo kameru s veličinom od **textureView**-a.

Funkcija **onPause** se poziva kada izađemo iz aplikacije ali je ne ugasimo. Tada se zatvara kamera i zaustavlja pozadinska dretva kako ne bi trošili resurse.

Funkcija **onDestroy** se poziva kada ugasimo aplikaciju i u tom slučaju zatvaramo klasifikator i pozivamo istu funkciju roditeljske klase.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**424**  **@Override**  **public** **void** **onResume**() {  **super**.onResume();  startBackgroundThread();  **if** (textureView.isAvailable()) {  openCamera(textureView.getWidth(), textureView.getHeight());  } **else** {  textureView.setSurfaceTextureListener(surfaceTextureListener);  }  }  **@Override**  **public** **void** **onPause**() {  closeCamera();  stopBackgroundThread();  **super**.onPause();  }  **@Override**  **public** **void** **onDestroy**() {  classifier.close();  **super**.onDestroy();  } |

Funkcija **setUpCameraOutputs** postavlja varijable relativno prema kameri koja se koristi. Argumenti su visina i širina pregleda kamere. Na početku pristupamo trenutnom activity-u i menadžeru kamere. Dohvatimo mapu s karakteristikama kamere koju koristimo te odaberemo najveću moguću rezoluciju za imageReader. **ImageReader** nam služi da uzmemo jedan frame s pregleda kamere i taj frame ćemo davati kao ulaz modelu strojnog učenja. Nakon toga postavimo omjer rezolucija na sve view-e koje koristimo. U slučaju da je došlo do pogreške pogreška se obrati prema tipu koja je.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**459**  **private** **void** **setUpCameraOutputs**(**int** width, **int** height) {  Activity activity = getActivity();  CameraManager manager = (CameraManager) activity.getSystemService(Context.CAMERA\_SERVICE);  **try** {  String cameraId = **this**.cameraId;  CameraCharacteristics characteristics = manager.getCameraCharacteristics(cameraId);  StreamConfigurationMap map = characteristics.get(CameraCharacteristics.SCALER\_STREAM\_CONFIGURATION\_MAP);  Size largest =  Collections.max(  Arrays.asList(map.getOutputSizes(ImageFormat.JPEG)), **new** CompareSizesByArea());  imageReader = ImageReader.newInstance(largest.getWidth(), largest.getHeight(), ImageFormat.JPEG, **2**);  **int** displayRotation = activity.getWindowManager().getDefaultDisplay().getRotation();  **int** sensorOrientation = characteristics.get(CameraCharacteristics.SENSOR\_ORIENTATION);  Point displaySize = **new** Point();  activity.getWindowManager().getDefaultDisplay().getSize(displaySize);  **int** rotatedPreviewWidth = width;  **int** rotatedPreviewHeight = height;  **int** maxPreviewWidth = displaySize.x;  **int** maxPreviewHeight = displaySize.y;  previewSize =  chooseOptimalSize(  map.getOutputSizes(SurfaceTexture.class),  rotatedPreviewWidth,  rotatedPreviewHeight,  maxPreviewWidth,  maxPreviewHeight,  largest);  layout\_frame.setAspectRatio(previewSize.getHeight(), previewSize.getWidth());  textureView.setAspectRatio(previewSize.getHeight(), previewSize.getWidth());  drawView.setAspectRatio(previewSize.getHeight(), previewSize.getWidth());  } **catch** (CameraAccessException e) {  Log.e(TAG, "Failed to access Camera", e);  } **catch** (NullPointerException e) {  ErrorDialog.newInstance(getString(R.string.camera\_error))  .show(getChildFragmentManager(), FRAGMENT\_DIALOG);  }  } |

Funkcija **openCamera** se brine da se aplikacija spoji s kamerom po zadanim veličinama rezolucije kao argumentima. Funkcija prvo provjerava da li ima sve dozvole za pristup kameri. Ako nema onda radi zahtjev za dozvolu za kameru kojoj je kod 1. Ako ima dozvole onda postavlja varijablu checkedPermission na true. Nakon toga poziva funkciju **setUpCameraOutputs** koja postavlja varijable na osnovi kamere koju koristimo. Poslije dobavlja trenutno korišteni activity i pomoću njega menadžera kamera. Prvo postavljamo semafor cameraOpenCloseLock kojem postavljamo tajmer od 2500 milisekundi koji nam osigurava da ako za 2.5 sekunde se ne uspijemo spojiti na kameru onda baci exception i ne dopušta da se aplikacija ugasi dok se spaja na kameru jer je moguće da bi se aplikacija ugasila ali bi se i dalje pokušavao spojiti na kameru. Nakon toga se spajamo na kameru preko **cameraId**-a. Ako dođe do pogreške pogrešku obradimo prema njezinom tipu.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**575**  **private** **void** **openCamera**(**int** width, **int** height) {  **if** (!checkedPermissions && !allPermissionsGranted()) {  FragmentCompat.requestPermissions(**this**, getRequiredPermissions(), **1**);  **return**;  } **else** {  checkedPermissions = **true**;  }  setUpCameraOutputs(width, height);  Activity activity = getActivity();  CameraManager manager = (CameraManager) activity.getSystemService(Context.CAMERA\_SERVICE);  **try** {  **if** (!cameraOpenCloseLock.tryAcquire(**2500**, TimeUnit.MILLISECONDS)) {  **throw** **new** **RuntimeException**("Time out waiting to lock camera opening.");  }  manager.openCamera(cameraId, stateCallback, backgroundHandler);  } **catch** (CameraAccessException e) {  Log.e(TAG, "Failed to open Camera", e);  } **catch** (InterruptedException e) {  **throw** **new** **RuntimeException**("Interrupted while trying to lock camera opening.", e);  }  } |

Funkcija **closeCamera** služi da kada gasimo aplikaciju da se od spoji od kamere.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**617**  **private** **void** **closeCamera**() {  **try** {  cameraOpenCloseLock.acquire();  **if** (**null** != captureSession) {  captureSession.close();  captureSession = **null**;  }  **if** (**null** != cameraDevice) {  cameraDevice.close();  cameraDevice = **null**;  }  **if** (**null** != imageReader) {  imageReader.close();  imageReader = **null**;  }  } **catch** (InterruptedException e) {  **throw** **new** **RuntimeException**("Interrupted while trying to lock camera closing.", e);  } **finally** {  cameraOpenCloseLock.release();  }  } |

Funkcija **startBackgroundThread** služi kao funkcija koja će kreirati pozadinsku dretvu koja će pozivati funkciju **periodicClassify** u beskonačnoj petlji. Funkcija **stopBackgroundThread** služi za otpuštanje pozadinske dretve kada se aplikacija gasi.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**642**  **private** **void** **startBackgroundThread**() {  backgroundThread = **new** HandlerThread(“CameraBackground”);  backgroundThread.start();  backgroundHandler = **new** Handler(backgroundThread.getLooper());  backgroundHandler.post(periodicClassify);  }  **private** **void** **stopBackgroundThread**() {  backgroundThread.quitSafely();  **try** {  backgroundThread.join();  backgroundThread = **null**;  backgroundHandler = **null**;  **synchronized** (lock) {  runClassifier = **false**;  }  } **catch** (InterruptedException e) {  Log.e(TAG, "Interrupted when stopping background thread", e);  }  } |

Funkcija **periodicClassify** je funkcija koje se poziva pomoću pozadinske dretve i onda poziva funkciju **classifyFrame** ako je varijabla **runClassifier** postavljena na true.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**672**  **private** Runnable periodicClassify =  **new** **Runnable**() {  **@Override**  **public** **void** **run**() {  **synchronized** (lock) {  **if** (runClassifier) {  classifyFrame();  }  }  backgroundHandler.post(periodicClassify);  }  }; |

Funkcija **classifyFrame** je glavna funkcija aplikacije i u njoj se poziva klasifikacija frame-a te se radi broj ponavljanja. Na početku se provjerava da li je klasifikator pokrenut i da li postoji trenutni activity i da li je kamera spojena. Ako jedno od toga nije istinito onda se ispisuje na grafičkom sučelju da nije još sve spremno i izlazi se iz funkcije. Ako je sve istina onda se dohvaća trenutni frame i sprema se u bit mapu i ona se šalje na klasifikaciju. Klasifikator vraća u obliku string koliko je milisekundi bilo potrebno za klasifikaciju, te nakon toga možemo izbrisati memoriju koju je zauzela bit mapa jer je više ne trebamo. Nakon toga pozivamo funkciju **setDrawPoint** od drawView-a s kao argumentima izlaz klasifikatora te prag. Ako je varijabla **counterRunning** na false, što znači da vježba nije pokrenuta, izlazimo iz funkcije.

U varijabli squat imamo spremljenu opisanu vježbu za čučnjeve. Varijabla je niz ključnih točaka u jednom ponavljanju vježbe, te je svaka ta ključna točka je niz s koordinatama u kojima dio tijela treba biti relativno od početnih koordinata kada se krenulo s jednim ponavljanjem.

U vektor q se spremaju najviše zadnjih 1000 pozicija čovjeka i preko toga vektora i opisne vježbe se uspoređuje da je napravljeno barem jedno ponavljanje i ako je onda se vrijednost brojača ponavljanja poveća i ažurira se vrijednost prikazana na grafičkom sučelju za koliko je ponavljanja napravljeno.

|  |
| --- |
| #app/java/Camera2BasicFrament.java:**783**  **int**[][] squat = **new** **int**[][]{ {**0**, **0**, **0**, **0**, **0**}, {**0**, **6**, **12**, **6**, **0**} };  **int** counter = **0**;  Vector<Pair <Integer, Integer> > q =**new** Vector<>();  **int** diff = **3**;  **private** **void** **classifyFrame**() {  **if** (classifier == **null** || getActivity() == **null** || cameraDevice == **null**) {  showToast("Uninitialized model or invalid context.");  **return**;  }  Bitmap bitmap = textureView.getBitmap(classifier.getImageSizeX(), classifier.getImageSizeY());  String textToShow = classifier.classifyFrame(bitmap);  bitmap.recycle();  drawView.setDrawPoint(classifier.mPrintPointArray, **0.5f**);  **if** (!counterRunning)  **return**;  String output = "";  q.add(Pair.create(((**int**) classifier.mPrintPointArray[**0**][**0**]), ((**int**) classifier.mPrintPointArray[**1**][**0**])));  **for** (**int** off = **0**; off < q.size(); off++) {  **int** passed = **0**;  **int** firstX = q.get(off).first, firstY = q.get(off).second;  **for** (**int** i = off; i < q.size(); i++) {  **int** deltaX = abs(q.get(i).first - firstX - squat[**0**][passed]),  deltaY = abs(q.get(i).second - firstY - squat[**1**][passed]);  **if** (deltaX \* deltaX + deltaY \* deltaY <= diff \* diff) {  passed++;  }  **if** (passed == squat[**0**].length) {  counter++;  **break**;  }  }  **if** (passed == squat[**0**].length) {  q.clear();  **break**;  }  }  showToast(String.valueOf(counter));  } |

## PoseEstimation

U **PoseEstimation.java** datoteci se nalazi klasifikator i u njoj pristupamo klasifikatoru. U redu 33 se nalazi konstruktor za klasu **PoseEstimation**. U njoj dohvaćamo veličinu ulazne slike da bi mogli inicializirati niz u kojem ćemo spremiti ulaznu sliku i onda je veličine **visina \* širina \* 3** (svaki piksel se sastoji od 3 boje) te postavimo taj niz u nule. Nakon toga kreiramo putanju do grafa, kojeg smo istrenirali i koji je spremljen pod resursima projekta, kako bismo mu mogli pristupiti. Na kraju se kreira klasifikator gdje se određuje koliko će dretvi imati za izračun rezultata, koja pravila da koristi, koji će prioritet imati na procesoru i na kraju dajemo putanju do grafa. Kreiramo engine s imenom „cpm\_1” koji će se pokretati na grafičkoj kartici.

|  |
| --- |
| #app/java/PoseEstimation.java:**33**  PoseEstimation(Activity activity) **throws** IOException {  **int** lengthValues = getImageSizeY() \* getImageSizeX() \* **3**;  **float**[] floatValues = **new** **float**[lengthValues];  floatBuffer = FloatBuffer.wrap(floatValues, **0**, lengthValues);  kernelPath = Environment.getExternalStorageDirectory().getAbsolutePath() + File.separator + "mace";  File file = **new** File(kernelPath);  **if** (!file.exists()) {  file.mkdir();  }  **int** result = JniMaceUtils.maceMobilenetSetAttrs(**2**, **0**, **3**, **3**, kernelPath);  JniMaceUtils.maceMobilenetCreateEngine("cpm\_v1", "GPU");  } |

Pošto graf kao ulaz ne prima bitmapu za sliku nego niz brojeva moramo bitmapu pretvoriti u niz a to radimo u funkciji **convertBitmapToByteBuffer** kojoj prosljeđujemo bitmapu koju smo dobili kao argument **classifyFrame** funkciji. Prvo spremimo trenutno vrijeme u varijablu te pokrenemo klasifikator kojemu ako argument damo ulazne podatke što je slika u nizu i klasifikator vraća koordinate čovjeka kao niz parova. Poslije toga uzmemo opet trenutno vrijeme i oduzmemo ga s vremenom koje je bilo prije klasificiranje i dobijemo vrijeme koje je bilo potrebno da se klasificira taj frame. I vraćamo prošlo vrijeme u obliku stringa.

|  |
| --- |
| #app/java/PoseEstimation.java:**33**  String **classifyFrame**(Bitmap bitmap) {  convertBitmapToByteBuffer(bitmap);  **long** startTime = SystemClock.uptimeMillis();  mPrintPointArray = JniMaceUtils.maceMobilenetClassify(floatBuffer.array());  **long** endTime = SystemClock.uptimeMillis();  String textToShow = Long.toString(endTime - startTime) + "ms";  **return** textToShow;  } |

## DrawView

U datoteci **DrawView.java** se nalazi klasa **DrawView** koja služi za crtanje predviđanje čovjekova pozicije na slici prikaza. Na početku datoteke se nalaze definirane boje u heksadecimalno obliku za svaki dio tijela za koji predviđamo poziciju.

|  |
| --- |
| #app/java/DrawView.java:**17**  **public** **static** **final** **int** COLOR\_TOP = Color.parseColor("#980000");  **public** **static** **final** **int** COLOR\_NECK = Color.parseColor("#ff0000");  **public** **static** **final** **int** COLOR\_RSHOULDER = Color.parseColor("#ff9900");  **public** **static** **final** **int** COLOR\_RELBOW = Color.parseColor("#ffff00");  **public** **static** **final** **int** COLOR\_RWRIST = Color.parseColor("#00ff00");  **public** **static** **final** **int** COLOR\_LSHOULDER = Color.parseColor("#00ffff");  **public** **static** **final** **int** COLOR\_LELBOW = Color.parseColor("#4a86e8");  **public** **static** **final** **int** COLOR\_LWRIST = Color.parseColor("#0000ff");  **public** **static** **final** **int** COLOR\_RHIP = Color.parseColor("#9900ff");  **public** **static** **final** **int** COLOR\_RKNEE = Color.parseColor("#274e13");  **public** **static** **final** **int** COLOR\_RANKLE = Color.parseColor("#e6b8af");  **public** **static** **final** **int** COLOR\_LHIP = Color.parseColor("#0c343d");  **public** **static** **final** **int** COLOR\_LKNEE = Color.parseColor("#1c4587");  **public** **static** **final** **int** COLOR\_LANKLE = Color.parseColor("#073763");  **public** **static** **final** **int** COLOR\_BACKGROUND = Color.parseColor("#20124d"); |

Funkcija **onDraw** se poziva kada smo klasificirali određeni frame i sada trebamo nacrtati predviđene pozicije. Prvo se provjerava da li je **Painter** deklariran, ako nije onda ga deklariramo i postavimo mu stil da su oblici puni i omogućimo Antialiasing koje nam osigurava da će nam crte biti ravne a ne pikselaste. Prvo prođemo po svim točkama koje smo predvidjeli i za svaku nacrtamo krug s njezinom definiranom bojom. Nakon toga treba povezati te točke da bi tvorile kostur čovjeka.

|  |
| --- |
| #app/java/DrawView.java:**98**  **@Override**  **protected** **void** **onDraw**(Canvas canvas) {  **super**.onDraw(canvas);  **if** (mPaint == **null**) {  mPaint = **new** Paint();  mPaint.setStyle(Paint.Style.FILL);  mPaint.setAntiAlias(**true**);  }  canvas.drawColor(Color.TRANSPARENT);  **int** colorIndex = **0**;  **for** (PointF pointF : mDrawPoint) {  mPaint.setColor(mColorArray[colorIndex++]);  canvas.drawCircle(pointF.x, pointF.y, **8**, mPaint);  }  mPaint.setColor(Color.parseColor("#6fa8dc"));  mPaint.setStrokeWidth(**5**);  **if** (mDrawPoint.size() <= **0**)  **return**;  PointF p0 = mDrawPoint.get(**0**);  PointF p1 = mDrawPoint.get(**1**);  PointF p2 = mDrawPoint.get(**2**);  PointF p3 = mDrawPoint.get(**3**);  PointF p4 = mDrawPoint.get(**4**);  PointF p5 = mDrawPoint.get(**5**);  PointF p6 = mDrawPoint.get(**6**);  PointF p7 = mDrawPoint.get(**7**);  PointF p8 = mDrawPoint.get(**8**);  PointF p9 = mDrawPoint.get(**9**);  PointF p10 = mDrawPoint.get(**10**);  PointF p11 = mDrawPoint.get(**11**);  PointF p12 = mDrawPoint.get(**12**);  PointF p13 = mDrawPoint.get(**13**);  canvas.drawLine(p0.x, p0.y, p1.x, p1.y, mPaint);  canvas.drawLine(p1.x, p1.y, p2.x, p2.y, mPaint);  canvas.drawLine(p2.x, p2.y, p3.x, p3.y, mPaint);  canvas.drawLine(p3.x, p3.y, p4.x, p4.y, mPaint);  canvas.drawLine(p1.x, p1.y, p5.x, p5.y, mPaint);  canvas.drawLine(p5.x, p5.y, p6.x, p6.y, mPaint);  canvas.drawLine(p6.x, p6.y, p7.x, p7.y, mPaint);  canvas.drawLine(p1.x, p1.y, p11.x, p11.y, mPaint);  canvas.drawLine(p11.x, p11.y, p12.x, p12.y, mPaint);  canvas.drawLine(p12.x, p12.y, p13.x, p13.y, mPaint);  canvas.drawLine(p1.x, p1.y, p8.x, p8.y, mPaint);  canvas.drawLine(p8.x, p8.y, p9.x, p9.y, mPaint);  canvas.drawLine(p9.x, p9.y, p10.x, p10.y, mPaint);  } |

# Zaključak

Ovaj projekt, da bi ga mogao ostvariti, natjerao me da puno toga naučim neuronskim mrežama općenito te točnije o konvolucijskim neuronskim mrežama. S time sam se uhvatio za trend koji traje zadnjih par godina, a to je revolucija umjetne inteligencije koja je omogućena strojnim učenjem. Jako sam se zainteresirao za to područje i potaknulo na daljnje učenje o njoj što bi mi se kasnije moglo isplatiti u životu tako da se zaposlim u tom području. Mislim da je cilj aplikacije inovativan i da bi mogao postati široko korišten, ali da još nije u potpunosti gotov te da je više dokaz koncepta nego što je gotov proizvod te da ima još puno za raditi na njoj kako bi dostigla svoj cilj. Aplikacija bi se mogla razviti kao fitness trener koja bi pomagala pri rađenju vježbe gdje bi mogla upozoravati korisnika da li krivo izvodi vježbu. Moglo bi se još dodati da se kreira fitness plan za cijeli mjesec koliko serija za koju vježbu treba izvesti dnevno, te pošto aplikacija može brojati ponavljanja, može biti sigurna da je korisnik sva ta ponavljanja napravio a ne da je on kliknuvši na gumb potvrdio da je to napravio iako ne znači da je.

# Literatura

<https://en.wikipedia.org/wiki/Machine_learning>

<https://en.wikipedia.org/wiki/Artificial_neural_network>

<https://en.wikipedia.org/wiki/Convolutional_neural_network>

<https://www.youtube.com/watch?v=FmpDIaiMIeA>

<http://cocodataset.org/#home>

<https://arxiv.org/abs/1602.00134>

<https://github.com/ildoonet/tf-pose-estimation>

<https://www.tensorflow.org/lite/>

<https://mace.readthedocs.io/en/latest/>

<https://developer.android.com/studio/>