

Računarstvo usluga i analiza podataka

**Raspoznavanje ljudske aktivnosti pomoću smartphone senzora i strojnog učenja**

PROJEKTNI ZADATAK

**Damir Stipančić**

Osijek, 2021

**SADRŽAJ**

[1. UVOD 1](#_Toc82453614)

[2. MODEL 2](#_Toc82453615)

[2.1 Podaci 2](#_Toc82453616)

[2.2 Obrada podataka 3](#_Toc82453617)

[2.3 Neuronska mreža 5](#_Toc82453618)

[2.4 Spremanje modela za Android kompatibilnost 8](#_Toc82453619)

[3. ANDROID APLIKACIJA 9](#_Toc82453620)

[3.1 Java klasa modela 10](#_Toc82453621)

[3.2 Implementacija sučelja 11](#_Toc82453622)

[4. REZULTATI 15](#_Toc82453623)

[5. ZAKLJUČAK 16](#_Toc82453624)

[6. LITERATURA 17](#_Toc82453625)

# 

# UVOD

Cilj ovog projektnog zadatka je spojiti različite platforme te pokušati postići jedan zanimljiv spoj dva različita okruženja koja na prvu možda i ne pripadaju zajedno. Fokus nije nužno stavljen više na jedno okruženje od drugog, nego na sam spoj oba okruženja i postizanje jednog spoja koji funkcionira i vrši obradu podataka i rezultate prikazuje besprijekorno.

Prvi dio projekta je i tzv. „backend“ i odrađen je u Pythonu[1]. Ovaj „backend“ se sastoji od implementacije neuronske mreže i treniranja iste na podacima koji su prethodno nabavljeni sa internet repozitorija.

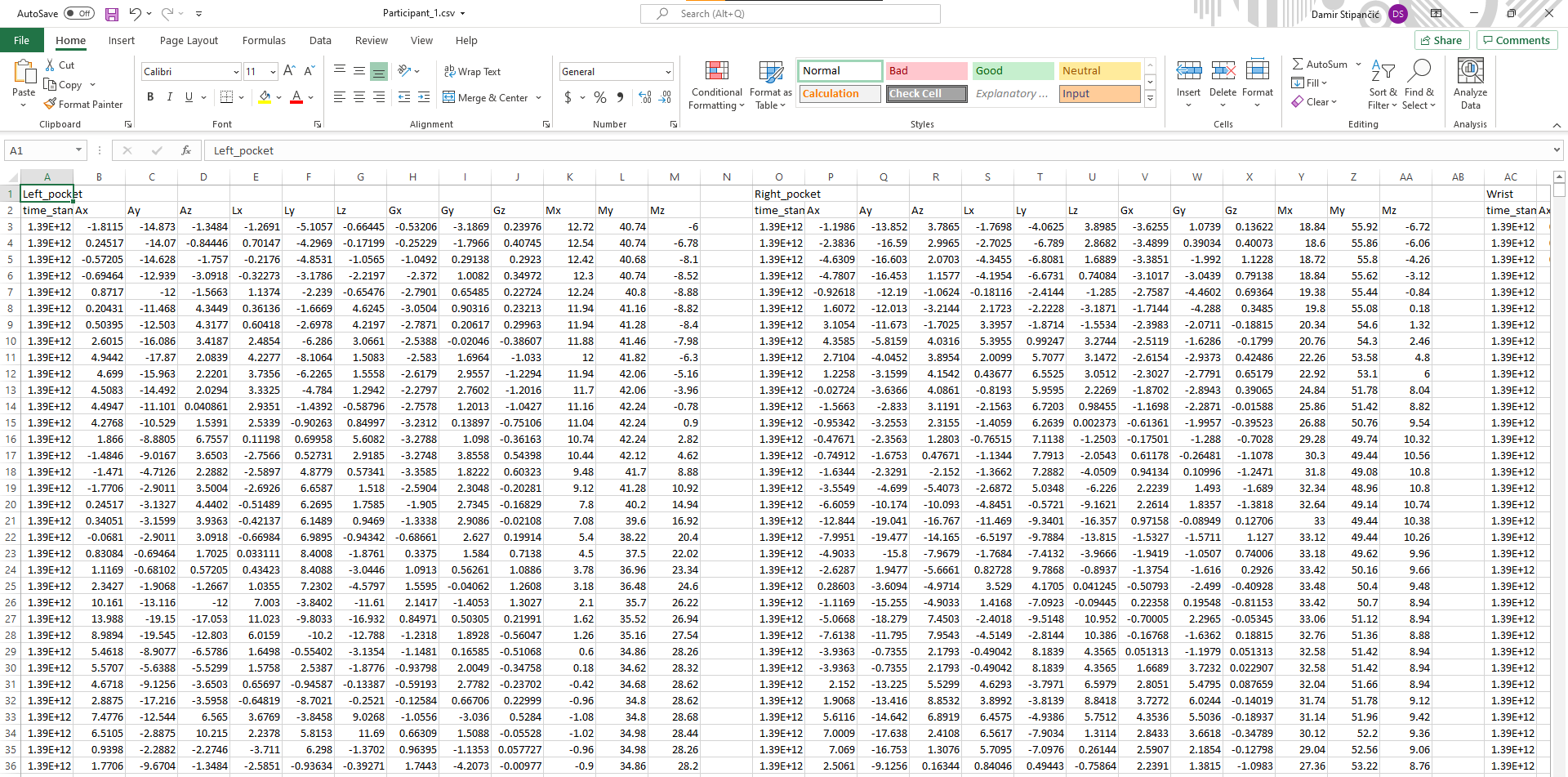
Drugi dio projekta predstavlja tzv. „frontend“ te se sastoji od Android[2] aplikacije unutar koje je implementiran Python model i koja onda prikazuje rezultate dobivene od strane modela.

Nažalost, ovaj projekt se ne sastoji od dijela u kojemu je model pružen kao web servis, tako da će sama Android aplikacija predstavljati jednu vrstu API-ja. Ovo je zbog jednostavne prirode Python modela koji se sastoji od LSTM neuronske mreže i nažalost ovakav model se ne može kreirati unutar Microsoft Azure[3] okruženja, te se također ne može pružiti kao servis jer jednostavno trenutno ne postoji podrška za LSTM input slojeve.

# MODEL

## Podaci

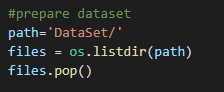
Za izradu neuronske mreže i treniranje iste iskorišten je „Sensor Activity Recognition DataSet“[4]. Alternativna opcija je bila „Smartphone-Based Recognition of Human Activities and Postural Transition Data Set“[5]. Oba skupa podataka sadrže podatke koji dolaze direktno sa senzora smartphone uređaja i mogu se iskoristiti za trening modela. Podaci su prikupljeni od 3 različita senzora, akcelerometar, žiroskop te senzor linearnog ubrzanja. Također, podaci su prikupljani u različitim pozicijama smartphone uređaja no za ovaj projekt iskorišteni su podaci kada se uređaj nalazi u lijevom ili desno džepu. To je najrealniji scenarij i najlakši za testiranje. Podaci su također podijeljeni u 10 različitih .csv datoteka, po jedna za svaku osobu koja je vršila prikupljanje.



Slika 2.1: *Trening podaci*

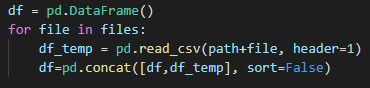
## Obrada podataka

Prije same izrade neuronske mreže i treninga, potrebno je pripremiti i obraditi podatke. Prvo je potrebno objediniti sve .csv datoteke, pošto su svi podaci podijeljeni prema osobi koja ih je prikupljala. Ovaj proces je vrlo jednostavan, koristi se „os“[6] biblioteka unutar Python okruženja i samo se spreme sve datoteke u varijablu te se izbaci zadnja datoteka unutar direktorija, pošto je to samo readme datoteka i ne sadrži podatke.



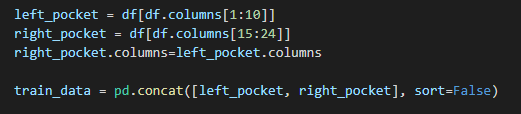
Slika 2.2: *Prikupljanje svih datoteka sa podacima*

Nakon toga, za lakšu manipulaciju i spajanje svih podataka u jednu datoteku, potrebno je prebaciti sve podatke u DataFrame tip podatka i izvršiti spajanje istovremeno. Za ovo iskorištena je „pandas“[7] biblioteka.



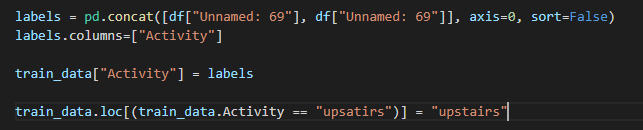
Slika 2.3: *Spajanje podataka u jednu datoteku*

Nakon ovog koraka, potrebno je odabrati podatke samo za lijevi i desni džep. Tada ih spajamo u jednu datoteku i popravljamo nazive stupaca tako da su jednaki.



Slika 2.4: *Odabir željenih podataka i spajanje u jedan skup*

Nakon ovoga, potrebno je ubaciti i klase svakog retka. Dohvaćamo ih iz originalnog skupa te ih spajamo u jedna stupac, te ih ubacujemo unutar „train\_data“ skupa koji smo prethodno već pripremili. Na kraju popravljamo jednu pogrešku tako da imamo pravilan broj klasa.



Slika 2.5: *Dohvaćanje klasa svakog retka i završetak pripreme podataka*

Zadnji korak pripreme podataka je preobrazba klasa u brojčane vrijednosti. Trenutno, tip kretanja za svaki redak je predstavljen kao tekst, odnosno String vrijednost. Potrebno je pretvoriti ih u odgovarajuće brojčane vrijednost. Ovo postižemo pomoću „LabelEncoder“ klase unutar „scikit-learn“[8] biblioteke. Svakoj aktivnosti će biti pridružena odgovarajuća brojčana vrijednost. U ovom slučaju, vrijednosti će biti od 1 do 7, dodijeljene abecedno.

Podaci su sada kompletni i spremni za ubacivanje u model.



Slika 2.6: *Kodiranje klasa*

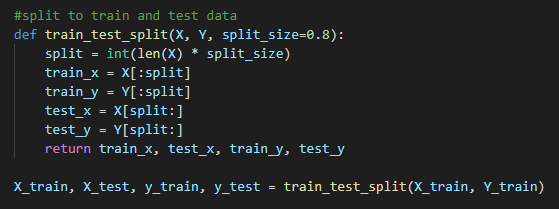
## Neuronska mreža

Za model strojnog učenja odabrana je LSTM(eng.: „Long short-term memory“) neuronska mreža. Ovaj tip neuronske mreže sadrži LSTM ulazni sloj koji omogućava da neuronska mreža kao ulaz primi skup kontinuiranih vrijednosti, umjesto samo jednu konstantnu vrijednost. Ovo je potrebno zbog same prirode podataka koji se koriste za treniranje i načina na koji model izvršava klasifikaciju i predikciju.

Naime, model će primati 2D matricu dimenzija 100x9. Za svaki tip senzora primiti će po 3 vrste podataka(X,Y i Z os) i to po 100 uzoraka za svaki. Jednostavno nema smisla vršiti predikciju na samo jednom uzorku sa svakog senzora. Ova vrijednost mora biti veća, u ovom slučaju 100 tako da se dobije kontinuirana vrijednost. Tako će model učiti te naravno onda i vršiti predikciju.

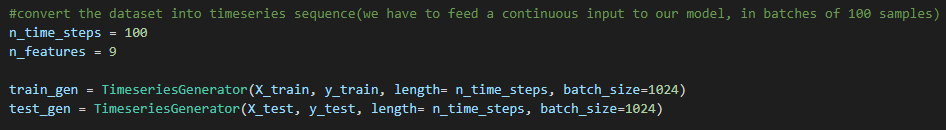
Model je izrađen dakle u Pythonu pomoću „keras“[9] biblioteke. Ova biblioteka sadrži alate potrebne za izradu neuronskih mreža i za njihovu validaciju, trening i testiranje.

Prvi korak je podijeliti podatke na trening i test podatke. Za ovo je implementirana vlastita funkcija koja kao ulaz prima trening podatke i potrebne klase za svaki niz podataka. Podjela je 80/20.



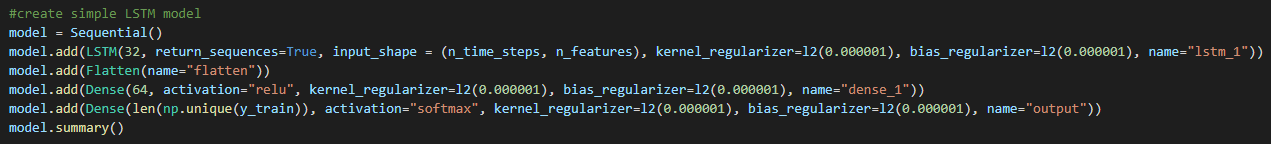
Slika 2.7: *Podjela podataka*

Nakon ovoga, potrebno je podijeliti podatke na nizove od po 100 podataka, kako bi se mogli predati modelu za trening. Ovo se postiže pomoću „TimeseriesGenerator“ klase iz „keras“ biblioteke. Definira se interval i broj različitih obilježja(stupaca) i to se preda klasi i spremi se u varijablu.



Slika 2.8: *Podjela podataka u nizove od 100 podataka*

Neuronska mreža je sekvencijalnog tipa, odnosno, izlaz iz svakog sloja je ulaz za sljedeći sloj. Sastoji se od 4 sloja. Prvi sloj je LSTM sloj koji prima kao ulaz matricu dimenzija 100x9. Sljedeći sloj je FLATTEN sloj koji pretvara ulaznu matricu u 1D matricu. Sljedeći sloj je skriveni sloj te zadnji sloj je output sloj. Ovaj sloj kao izlaz daje niz od 7 vrijednosti. Ove vrijednosti su postotak predikcije za svaku klasu. Svaki sloj također sadrži parametre za prevenciju tzv. „overfittinga“ odnosno sprečavanja da se model navikne na trening podatke i onda ne obavlja dobru predikciju na novim podacima.



Slika 2.9: *Implementacija neuronske mreže*

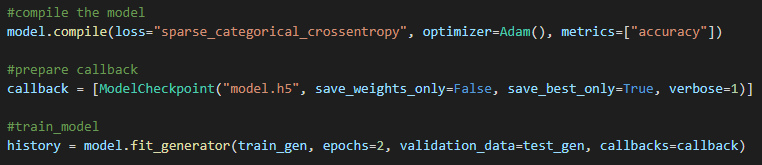
Nakon izrade, prikazane su informacije o neuronskoj mreži.

A picture containing text

Description automatically generated

Slika 2.10: *Detaljni prikaz informacija o neuronskoj mreži*

Nakon ovoga, potrebno je izvršiti validaciju, trening te spremanje modela. Model se sprema u intervalima tokom treninga te se tako omogućava spremanje najbolje modela. Spremljeni model ima 90% točnosti.



Slika 2.11: *Trening i spremanje modela*

## Spremanje modela za Android kompatibilnost

Sljedeći korak je spremanje modela tako da on bude kompatibilan sa Android platformom. Ovo je jako bitan korak jer se model ne može implementirati unutar Android platforme u .h5 formatu. Model je potrebno pretvori u .pb format, koji je spreman za tzv. „inferenciju“ odnosno pozivanje unutar Android platforme. Ovo se postiže pomoču tzv. „freeze graph“ funkcije unutar Pythona. Ova funkcija omogućava spremanje grafa modela u .chkp formatu i onda konverziju iz tog formata u .pb format. Jedino u ovom formatu je moguće pozvati model unutar Android platforme i izvršiti potrebnu predikciju.



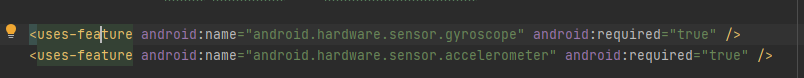
Slika 2.12: *Spremanje modela za korištenje unutar Android okruženja*

Definirani su potrebni nazivi za input i out slojeve te samo ime modela. Tada se ispisuje i sprema trenutno stanje modela u obliku „session graph“ datoteke te se onda vrši pretvaranje u .pb datoteku. Ova datoteka se ubacuje direktno unutar Android projekta te se poziva u kodu sa prethodno definiranim parametrima.

# ANDROID APLIKACIJA

Android aplikacija služi kao direktno sredstvo primjene modela strojnog učenja te ujedno i kao korisničko sučelje. Aplikacija je jednostavnog dizajna, samo će prikazivati predikciju modela strojnog učenje u vidu postotaka za svaku pojedinu klasu.

Prvi korak je omogućiti samo korištenje senzora unutar aplikacije. Ovo se postiže unutar Android Manifest dokumenta u kojem se postavljaju definicije za potrebne senzore, u ovom slučaju senzor za brzinu i žiroskop.



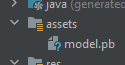
Slika 3.1: *Definiranje potrebnih senzora*

Nakon toga potrebno je definirati potrebne biblioteke, u ovom slučaju to je tensorflow biblioteka. Pošto je model kreiran pomoću tensorflow keras biblioteke, tensorflow će se iskoristiti i unutar Android aplikacije za potrebe dohvaćanju modela. Biblioteka se definira unutar gradle dokumenta aplikacije.



Slika 3.2: *Definiranje potrebne tensorflow biblioteke*

U svrhu pripreme potrebno je još samo ubaciti model, koji je prethodno spremljen u obliku .pb datoteke, unutar „assets“ direktorija unutar Android projekta.

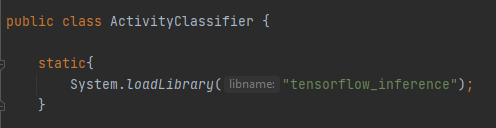


Slika 3.3: *Ubacivanje modela unutar Android projekta*

## Java klasa modela

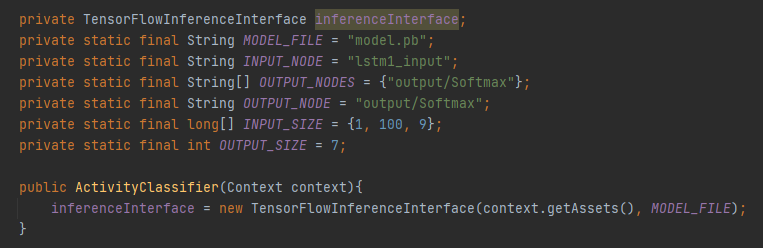
Prvi korak implementacije modela unutar Android aplikacije jest definiranje klase koja će predstavljati model. Kreira se nova klasa naziva „ActivityClassifier“ i ona će sadržavati sve potrebne elemente modela.

Prvo se učitava „tensorflow\_inference“ biblioteka. Ona sadrži sve potrebno za izvođenje predikcije modela unutar Android aplikacije.



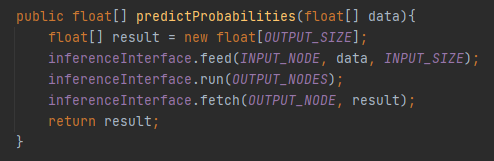
Slika 3.3: *Učitavanje tensorflow\_inference biblioteke*

Nakon toga, definiraju se potrebni parametri koji opisuju model. Ime modela, ime input i output sloja(ova imena moraju biti jednaka imenima koja su prethodno definirana prilikom pretvaranja modela u .pb datoteku) te dimenzije input i output podataka. Također se definira inference interface varijable pomoću koje će se pozivati potrebne metode.



Slika 3.4: *Definiranje potrebnih varijabli*

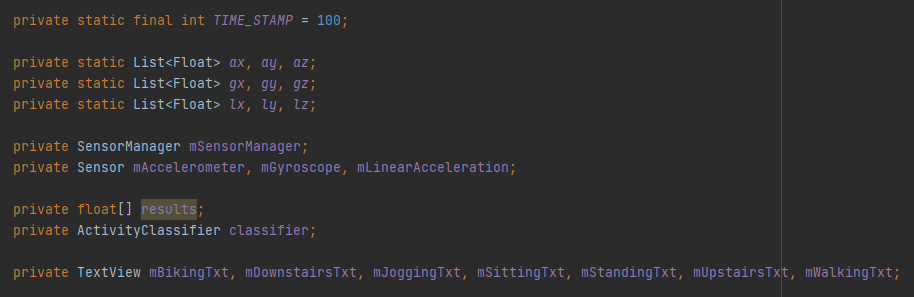
Na kraju se samo definira funkcija koja će biti zadužena za obavljanje predikcije. U njoj definiramo novi niz float vrijednosti koji će sadržavati izlaz iz modela, te uz pomoću funkcija „feed(), run() i fetch()“, koje su dio inference interface klase, predajemo ulaz modelu, izvodimo predikciju te dohvaćamo rezultate.



Slika 3.5: *Izvođenje predikcije modela i dohvaćanje rezultata*

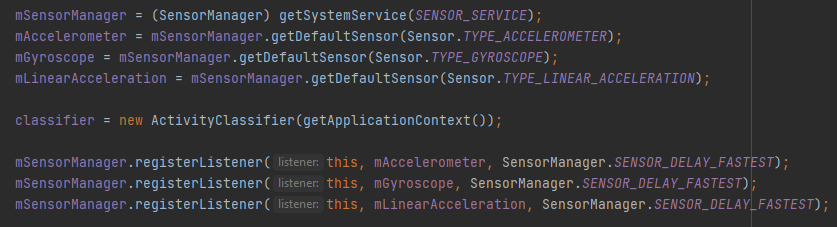
## Implementacija sučelja

Kao zadnji korak implementacije aplikacije potrebno je definirati korisničko sučelje u upravljanje senzorima. Prvo se definiraju potrebne varijable za UI elemente, liste koje će sadržavati podatke sa senzora te same varijable potrebne za upravljanje senzorima.



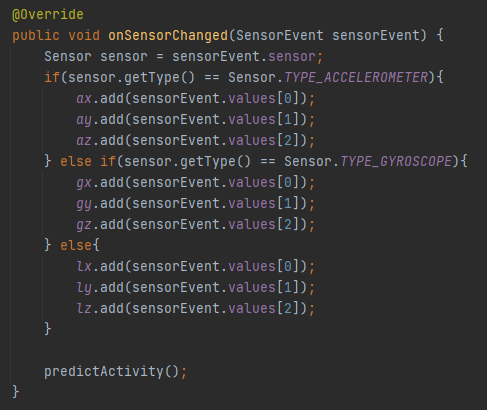
Slika 3.6: *Definiranje potrebnih varijabli*

Kao sljedeći korak, potrebno je definirati i dohvatiti senzore za upravljanje te je potrebno postaviti tzv „listener“ elemente na svaki od njih. Pomoću ovih elemenata moguće je „slušati“ promjene na senzorima i dohvaćati potrebne podatke.



Slika 3.7: *Dohvaćanje senzora i postavljanje „listener“ elemenata*

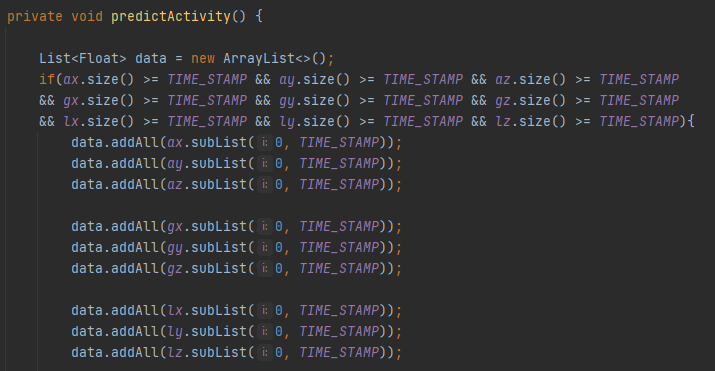
Pošto su implementirani „listener“ elementi, potrebno je također implementirati metodu unutar koje se obavljaju potrebne akcije kada dođe do promjene senzora ili vrijednosti na senzoru. Naime, kada god se promijeni senzor, obavit će se dodavanje vrijednosti unutar liste koja će sadržavati sve potrebne vrijednosti.



Slika 3.8: *Dodavanje potrebnih vrijednosti u zajedničku listu*

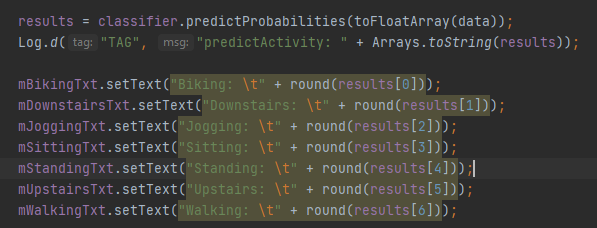
Na kraju se također poziva i funkcija koja će obaviti potrebnu predikciju i osvježiti UI elemente.

Ova funkcija dakle provjerava da li su vrijednosti sa svih senzora dosegle 100 iteracija ili više, te se onda ti nizovi od 100 vrijednosti dodaju u konačnu listu koja će predstavljati kontinuirani ulaz za model.



Slika 3.9: *Dohvaćanje intervala od 100 vrijednosti*

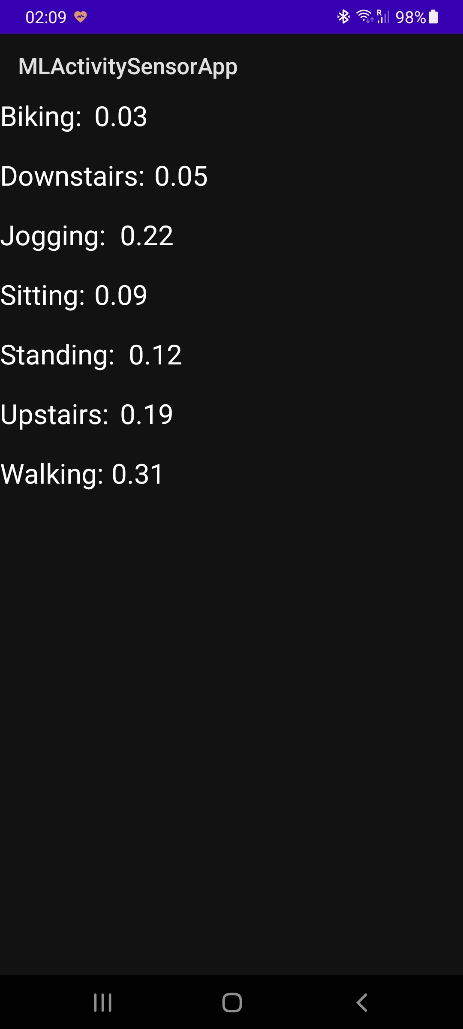
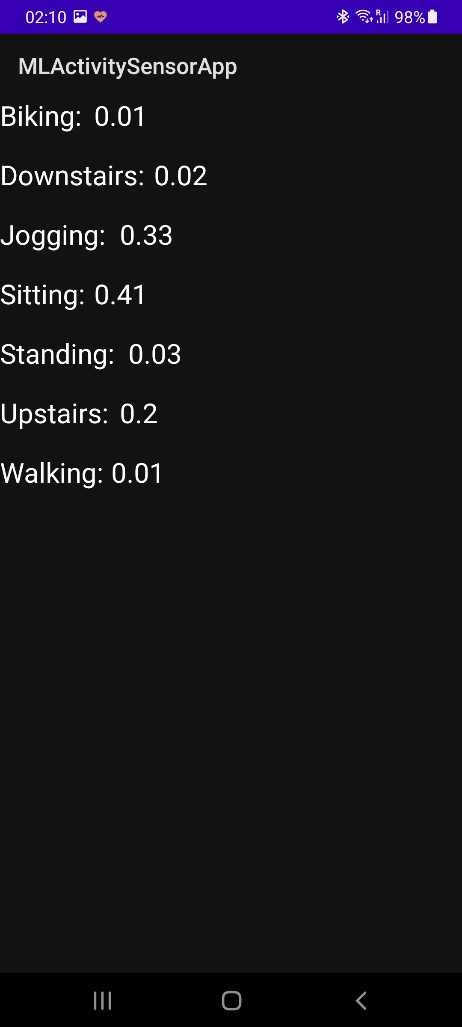
Nakon toga, poziva se funkcija koja vrši predikciju te se rezultat sprema u listu. Nakon toga, rezultati se objavljuju unutar UI elemenata aplikacije za lakši pregled.



Slika 3.10: *Izvršavanje predikcije i postavljanje rezultate na UI*

# REZULTATI

Konačan rezultat aplikacije:



Slika 4.1: *Rezultati prikazani unutar aplikacije*

Aplikacija prikazuje vjerojatnosti za svaku od aktivnosti u trenutku izvođenje te se nove vrijednosti pribavljaju svakih par sekundi kako stižu očitavanja sa senzora.

# ZAKLJUČAK

Ovaj projekt je predstavio jedan zanimljiv „cross-platform“ problem gdje je glavni izazov bio spojiti dva okruženja. I Python i Android okruženje su jako fleksibilni i zbog toga je ovaj projekt bio moguć. Danas je sve više podržano strojno učenje na raznim platformama i postoje razni načini kako se to može izvesti.

Jedini problem ovog projekta je to što se model jednostavno ne može pružiti kao web servis. Zbog jedinstvene prirode same neuronske mreže i njene primjene, nije bilo moguće podići sam model kao web servis. Microsoft Azure jednostavno ne podržava LSTM tipove neuronskih mreža, unutar vlastitih ML alata ili kao web servis koji se nalazi na Azure oblaku.

Druga ideja je bila iskoristiti Flask i Google Cloud no i tu je podrška jako slaba te taj proces jednostavno nije izvediv.

Projekt je unatoč tome uspješno izvršen i postignuto je potrebno ponašanje aplikacije i samog modela strojnog učenja.

# LITERATURA

[1] <https://www.python.org/doc/>

[2] <https://developer.android.com/docs>

[3] <https://azure.microsoft.com/en-us/>

[4]<https://www.researchgate.net/publication/266384007_Sensors_Activity_Recognition_DataSet>

[5] <http://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions>

[6] <https://docs.python.org/3/library/os.html>

[7] <https://pandas.pydata.org/>

[8] <https://scikit-learn.org/stable/>

[9] <https://keras.io/>