Универзитет у Београду
Електротехнички факултет

Примена алгоритама машинског учења у препознавању корисника на основу сензора покрета

Дипломски рад

Ментор:

Проф. др Захарије Радивојевић

Кандидат:

Леон Јовановић
2015/0377

Београд, Септембар 2020.

САДРЖАЈ

1. Увод

Машинско учење је област која се бави подстицањем рачунара да делују без изричитог програмирања. Машинско учење као појам постоји још од 50-их година прошлог века али није доживела нагли успон све до појаве интернета и до довољног развитка процесорских јединица. Интернет је био потребан да би алгоритми машинског учења имали довољно података, а јачи процесори су били неопходни да би обрадили велике количине података за прихватљиво време.

Неуралне мреже, као један од алгоритама машинског учења, је доживео још већи раст у популарности у последњих 20 година када је даљи развој хардверских компонената омогућио појаву дубоког учења. Дубоко учење представља скуп комплекснијих архитектура неуралних мрежа који је у стању да реши много комплексније проблеме у односу на његовог претходника.

Прикупљање података о корисницима је један од најважнијих циљева већине највећих ИТ компанија и светских влада. Већину скупљених података долазе из директних извора као што су камера, микрофон или геолокација, због чега се за њихово коришћење углавном тражи дозвола од корисника. Подаци са неких сензора мобилних телефона се и даље сматрају сувише сировим да би могли да произведу било какве битне информације, па се за приступ подацима жироскопа и акцелерометра не тражи одобрење корисника. Циљ овог дипломског рада је провера да ли је могуће из података тих уређаја сазнати неке информације о кориснику.

У последњих пар година вршило се неколико истраживања чија је тема била обрада података са сензора помоћу алгоритама машинског учења, али већина истраживања користи податке са свих сензора што подразумева камеру, микрофон и геолокацију. Једно истраживање са универзитета Станфорд је успело да постигне следеће резултате само помоћу података са жироскопа: успешност од 50% да препозна говорника од 10 различитих говорника, 65% успешност да препозна цифру од 0 до 9 ако алгоритам учи и тестира на једном говорнику, 26% успешност да препозна цифре ако има више говорника и 77% успешност да препозна цифру ако говори један говорник и ако се подаци узимају са више телефона у просторији.

Први део дипломског рада је било прикупљање података, који ће касније бити унети у алгоритам. Један тест представља 60 секунди бележења података са акцелерометра и жироскопа у јединственим ситуацијама на фреквенцији од 200Hz. Једна ситуација представља комбинацију особина као што су број, пол, локација, старост и језик говорника. За одређени временски период направљено је 315 тестова.

Други део је обрађивање података ради њихове припреме за убацивање у алгоритам и прављење различитих архитектура дубоких неуралних мрежа.

Трећи део рада је итерирање различитих скупова података кроз различите параметре свих модела неуралних мрежа које су направљене у претходном кораку и бележење статистике прецизности.

У другом поглављу ће бити описан преглед постојећег решења који се бави сличним проблемом као и овај рад. Биће описан истраживачки рад са његовим резултатима као и објашњено шта су предности, а шта ограничења тог рада у односу на овај дипломски рад.

Треће поглавље ће се бавити целокупним решењем дипломског рада и биће објашњене технологије које су коришћене. Такође ће се бавити и резултатима које смо добили током истраживања.

У четвртом поглављу ћемо прокоментарисати резултате као и њихову примену у реалном свету. Биће описана ограничења током рада, као и технике које би могле да побољшају резултате које смо добили.

2.ПРЕГЛЕД ПОСТОЈЕЋИХ РЕШЕЊА

У овом поглављу су наведене најважније информације везане за постојећа решења. Иако постоје многи истраживачки радови на ову тему у наставку ћемо се бавити истраживачким радом [1] са универзитета Станфорд јер је најсличнији теми којој се бави овај дипломски рад.

2.1. Анализа истраживачког рада са универзитета Стенфорд

У овом раду сакупљање података са мобилних уређаја је рађено са *Nexus 4* и *Samsung Galaxy S III* телефоном као и *Nexus 7* таблетом. *Nexus* користи *InvenSense*, док *Samsung* користи *STMicroelectronics* жироскопе и акцелерометре, што омогућује разноврснije и самим тим тачније податке. Извршено је 220 тестова од којих су 110 јединствени (сваки тест је поновљен два пута). У тестовима је учествовало 10 различитих говорника од којих су 5 мушког пола и 5 женског пола. Изговарали су следеће речи: „zero“, „oh“, „two“, …, „nine“.

Од алгоритама машинског учења користили су *Support Vector Machine (SVM)* који се користи за препознавање између две категорије, *Gaussian Mixture Model (GMM)* који се користи за груписање неозначених података и *Dynamic Time Warping (DTW)* који се користи за препознавање говора тако што препозна сличности између две секвенце.

Са горе наведеним подацима и алгоритмима су добили следеће резултате:

- Препознавање пола говорника (Табела 2.1.1.)
- Препознавање једног говорника из групе од пет (Табела 2.1.2.)
- Препознавање цифара са више говорника (Табела 2.1.3.)
- Препознавање цифара са једним говорником (Табела 2.1.4.)

Табела 2.1.1. Идентификација пола говорника

	SVM	GMM	DTW
Nexus 4	80%	72%	84%
Samsung Galaxy S III	82%	68%	58%

Табела 2.1.2. Идентификација говорника

	SVM	GMM	DTW
Mixed female/male	23%	21%	50%
Female speakers	33%	32%	45%
Male speakers	38%	26%	65%

Табела 2.1.3. Идентификација цифара са више говорника

	SVM	GMM	DTW
Mixed female/male	10%	9%	17%
Female speakers	10%	9%	26%
Male speakers	10%	10%	23%

Табела 2.1.4. Идентификација цифара са једним говорником

SVM	GMM	DTW
15%	5%	65%

У табелама 2.1.2., 2.1.3. и 2.1.4. нису приказани подаци са телефона *Samsung Galaxy S III* јер су они знатно лошији у односу на *Nexus 4* телефон. Препознавање пола помоћу *SVM* и *DTW* алгоритма у овом раду су постигли боље резултате него помоћу дубоких неуралних мрежа које користимо у овом дипломском раду, о чему ћемо причати више у наредним поглављима. Са друге стране, прецизност препознавања речи, односно цифара са једним или више говорника код сва три алгоритма која је постигнута у раду са универзитета Стенфорд је значајно нижа него што је постигнуто у овом раду.

Предности горе наведеног истраживања је то да користе филтрирање шумова током претпроцесовања података јер шумови могу да збуне алгоритам и самим тим подаци које убацују у алгоритме су прецизнији и тачнији. Такође, обрађују и случај где, уколико се у просторији налази више уређаја који мирују, могу да се узму подаци са више телефона и самим тим добију прецизније информације о особама које причају у просторији.

Недостатак тог рада је мали број тестова, јер повећање броја примера, посебно код истраживања где их иницијално нема пуно, веома често доводи до великог побољшања у резултатима. Такође, број типова тестова је прилично мали, с обзиром на то да постоји још велики број типова информација које можемо потенцијално сазнати из података са жироскопа и акцелерометра.

Као што се може приметити у табелама 2.1.3. и 2.1.4., прецизност препознавања цифара са једним или више говорника није задовољавајућа (посебно са више говорника) што нам говори или да је проблем који су покушали да реше сувише комплексан за алгоритме који су користили или да нису користили довољно примера да алгоритам може да научи или оба. За решавање сличног проблема у овом дипломском раду се користе алгоритми који су способни да реше комплексније проблеме, као и већи број примера што је углавном доводило до бољих резултата.

3.Преглед детаља решења

У овом поглављу ће детаљно бити објашњен поступак целокупног рада и долазак до крајњих резулатата. У наставку ће бити описан поступак прикупљања података, као и рад апликације која бележи те податке. Биће детаљно приказане методе за претпроцесирање података, као и алати и алгоритми који су коришћени за обрађивање података.

3.1. Сакупљање података

Без обзира који алгоритам машинског учења користимо, најбитније је да имамо скуп података који ће бити тачан, довољно велики и да обухвати све ситуације на начин да се не деси да постоји ситуација у којој нема или има премало одрађених тестова. Наставак потпоглавља ће се бавити прегледом свих тестова, прављењем апликације за сакупљање података и спецификацијама уређаја који су били коришћени.

3.1.1. Преглед свих ситуација за прављење тестова

На почетку је битно да за све тестове које будемо извршавали над овим подацима постоје довољан број тестова за сваку категорију. Прављењем свих могућих комбинација у малом програму на Јава програмском језику осигурано је да се није превидела ниједна категорија.

```
String data = "";
for(int i = 0; i<3; i++) //Broj jezika
      for(int j = 0; j<3; j++) //Distanca od izvora
            for(int k = 0; k<3; k++) //Izvor zvuka
                  for(int l = 0; l<3; l++) //Broj govornika
                        for(int m = 0; km<5; m++) //Lokacija testa
                              for(int n = 0; n<3; n++) //Pol govornika
                                    for(int o = 0; o<3; o++) //Broj godina
                                                                  govornika
                                    {
                                    if(notPossibleTest(i,j,k,l,m,n,o))
                                                continue;
                                    data = "";
                                          data += writeLanguage(writer,i);
                                          data += writeDistance(writer,j);
                                          data += writeSource(writer,k);
                                          data += writeNumPers(writer,l);
                                          data += writeLocation(writer,m);
                                          data += writeGender(writer,n);
                                          data += writeYears(writer,o);
                                    }
```

Слика 3.1.1. Програмски код за исписивање свих могућих тестова

На слици 3.1.1. се види програмски код који представља итерирање кроз све могуће особине једног теста. Функција *notPossibleTest* прескаче ситуације које нису могуће због ограничених ресурса. На излазу кода, у тескт фајл, се исписује резултат где је свака особина одвојена табовима да би *Microsoft Excel* могао лако да пребаци податке из текст фајла у табелу.

Тестови су подељени у седам категорија да би се осигурала разноврсност. Број особа је подељен у 3 подкатегорије од 1, 2 и 3 или више особа. Локације треба да симулирају различите

позадинске шумове и подељени су на стан, аутомобил, парк, ресторан и улицу. Дистанца од извора звука је тестирана на 0-2, 3-5 и 5 или више метара. Језик говорника је био тестиран на српски, енглески и немачки језик. Да бисмо симулирали све могуће ситуације, није било могуће ослањати се само на говор уживо, већ се за ситуације са више људи, старијих или млађих особа и других ситуација, користио говор преко звучника телефона или телевизора. Извор звука је у свим изводљивим ситуацијама био говор уживо, док се у осталим ситуацијама користио говор из филмова, серија или са *Youtube* платформе. Обзиром да ниво звука могуће мењати у филмовима, серијама и *Youtube* платформи, пре сваког снимања се звук подешавао да што боље имитира глас уживо.

3.1.2. Подаци

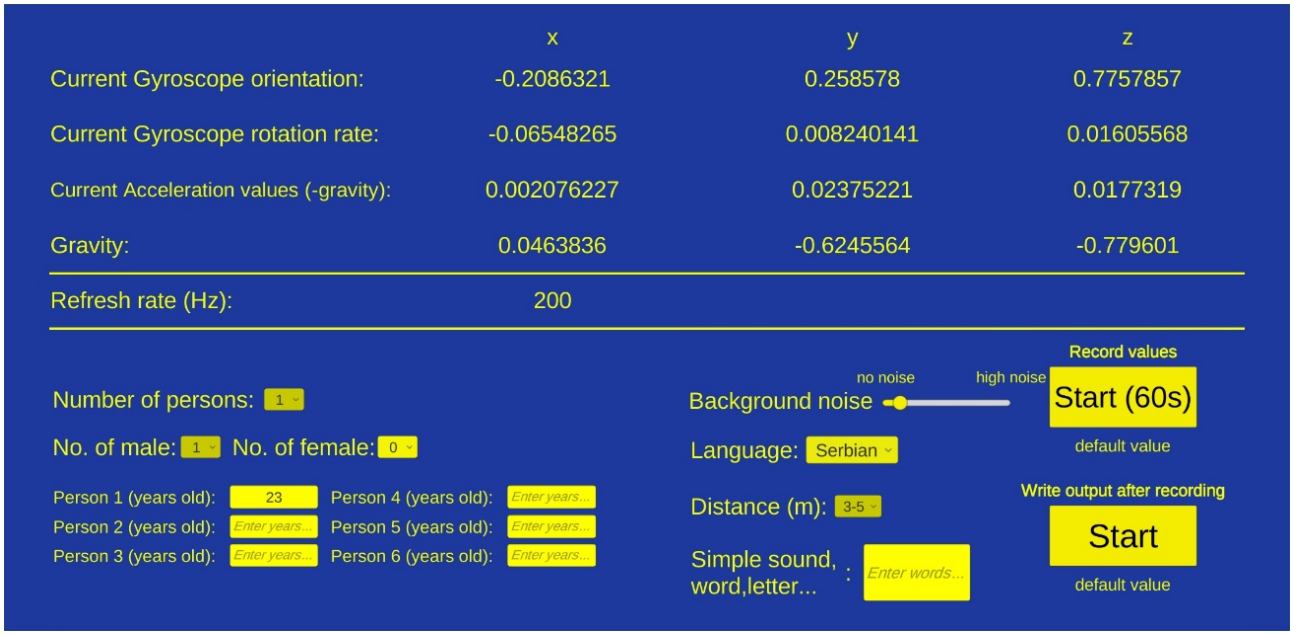
Један извршен тест садржи улазне податке и излазне ознаке. Подаци са жироскопа и акцелератора се бележе 60 секунди док телефон мирује. Подаци су у облику координата *x*,*y* и *z* осе, а у једној секунди се забележи 200 координата обзиром да је одабрана фреквенција 200Hz.

Излазне ознаке су подаци који ће служити за проверу резултата на излазу алгоритма. Они обухватају све типове тестова које ћемо обављати над улазним подацима. Позадински шум нам говори колико шума тј. буке постоји у окружењу док траје тест као и број особа који прича током теста, број мушких и женских особа, језик говора, као и специфичан случај где се изговарају слово „а“, и речи „књига“, „*learning*“ и „*schmetterling*“ појединачно. Сваки од тих излазних ознака ће бити посебно тестиран да се провери колико је прецизно могуће предвидети ознаку као резултат на излазу алгоритма где смо довели један улазни податак.

3.1.3. Апликација за бележење података

Обзиром да су потребни подаци жироскопа и акцелерометра са мобилног телефона, било је потребно направити апликацију која ће моћи да приступи тим подацима и забележи их у фајлове. Такође било је потребно и направити интерфејс где ће се уносити особине теста, да би подаци које касније уносимо у алгоритам били означени.

Апликација је прављена у програму *Unity 5* и програмском језику *C#* ради лаког и брзог преноса на мобилни уређај. Као што се може видети на слици 3.1.2., прва половина екрана представља вредности које можемо да прочитамо са жироскопа и акцелерометра, као и вредности гравитације по осама. Гравитација утиче на вредности акцелерометра тако да је било неопходно прво одузети вредности акцелерометра од вредности гравитације по осама да би се добила тачна очитавања са акцелерометра. Приказује се и вредност фреквенције одабирања коју смо одабрали да буде 200Hz, да бисмо касније од тих података могли да направимо податке од по 50 и 100Hz, о чему ћемо причати у наставку текста.



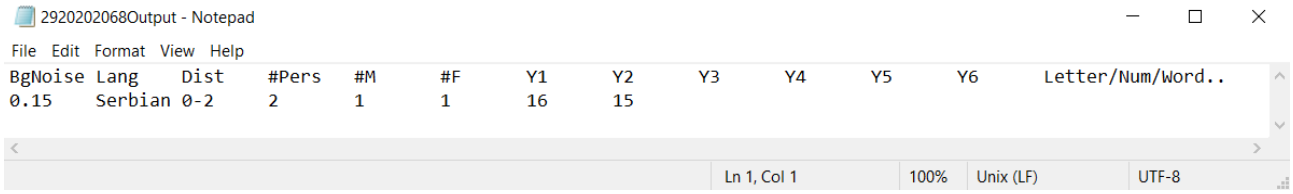
Слика 3.1.2. Апликација за бележење података

Доња половина су подаци које корисник мора сам да унесе да би прецизно знали у којој ситуацији је тест извршен. Уколико су нека поља сувишна, њих смо остављали празним. За снимање улазних података потребно је притиснути прво, горње дугме *Start* након чега ће одбрајавати тајмер до 60 секунди. За тих 60 секунди апликација чита вредности са сензора и уписује их у 3 одвојена текст фајла. Први део назива фајла је тренутни датум и време како би смо касније податке лакше сортирали а други део назива је опис података које снимамо. Постоје 3 различита типа података, од којих 2 долазе са жироскопа и то су оријентација у простору и ротациона брзина телефона а са акцелерометра подаци представљају убрзање по осама. Апликација уписује координате симултано у сва три текст фајла, у којима ће на крају бити исписано по 12000 *x*, *y* и *z* координата. Пример таквог фајла видимо на слици 3.1.3. Такође паралелно се и прави аудио фајл који садржи вредности микрофона у временском периоду од 60 секудни. Аудио фајл се не користи у било каквом смислу за дубоке неуралне мреже, већ за проверу и корекцију излазних ознака уколико дође до грешке.

Слика 3.1.3. Пример улазних података са сензора акцелерометар

Када се заврши 60 секунди прикупљања података са телефона у стању мировања, треба унети или проверити уколико су већ унешене ознаке датог теста. Одређивање позадинског шума се уносила ручним померањем слајдера где је минимум представљао да нема шума, док максимум представља веома јак позадински шум а на излазу се појављује вредност од 0 до 1. Остале ознаке за уношење су прецизне и јасне. Следећи корак је притиснути друго, доње дугме

Start да би се све вредности ознака уписале у пети фајл чији назив има датум и време као претходна четири фајла и опис „*Output*“ као што се може видети на слици 3.1.4.



Слика 3.1.4. Пример излазних података

3.1.4. Спецификација коришћених уређаја

Читање података са жироскопа и акцелератора се извршавало са телефона *Samsung Galaxy S9*. Наведени телефон користи жироскоп и акцелерометар који су део сензорног пакета *LSM6DSL* [2] компаније *STMicroelectronics*. *LSM6DSL* пакет садржи *3D* дигитални жироскоп и *3D* дигитални акцелерометар.

Жироскоп и акцелератор имају 4 режима рада: искључен режим, режим слабих перформанси, нормалан режим и режим високих перформанси. У режиму слабих перформанси је могуће користити фреквенције 12.5, 26 и 52 Hz. У нормалном режиму дозвољене фреквеције су 104 и 208 Hz. У режиму високих перформанси је могуће користити цео опсег фреквенцији од 12.5 Hz до 6.66 kHz.

У ситуацијама где је било потребно пуштати говор са неког уређаја, коришћена су два уређаја у зависности од локације. Први уређај је *iPhone XS* који користи дуални звучник на горњој и доњој страни телефона. Јачина звука гласа у просеку износи 71.5 dB, а просечна јачина звука када се пушта музика износи 75.7 dB [3]. Други уређај је телевизор *49UJ670V* од произвођача *LG*.

3.2. Обрађивање података

У овом поглављу ће бити разјашњено на који начин су подаци били обрађивани да би били спремни за улазак у дубоку неуралну мрежу. То подразумева конвертовање координата да буду истог формата, прављење додатних скупова података из оригиналног скупа као и нормализација и насумично мешање података.

3.2.1. Форматирање координата

Приликом детектовања координата и њиховог уписивања у текст фајлове, уколико број децимала пређе одређену величину дати број буде конвертован у експоненцијалан формат. Да би сви бројеви били у децималном формату, направљена је мала скрипта у Python програмском језику која анализира три текст фајла који садрже улазне податке по тесту и конвертује експоненцијални у децимални формат.

```
def replaceToFloat(text):
    spliting = re.split('E-|\t|\n',text)
    power = int(spliting[1]) #Broj nakon E-0 koji govori koliko decimala treba da dodamo
    num = float(spliting[0]) #Broj pre E-0
    d = decimal.Decimal(spliting[0]) #Broj decimala
    n = abs(d.as_tuple().exponent)+power #Ukupan br decimala
    return "{:."+str(n)+"f}".format(float(num/(10**power))) #Formatiramo da fiksiramo ukupan broj decimala
def convert(path1,filename1):
    file = open(path1 +"\\"+ filename1, "r+") #Otvori fajl
    lines = file.readlines() #Uzmi sve linije
    count = 0
    rewrite = ""
    for line in lines: #Uzimaj liniju po liniju
        if(re.search('E-0', line)): //Ukoliko sadrži format koji tražimo
            numbers = re.split("\t|\n", line)
            line = ""
            if(re.search('E', numbers[0])): #Ako se nalazi na X osi
                line += replaceToFloat(numbers[0]) + '\t'
            else:
                line += numbers[0] + '\t'
            if(re.search('E', numbers[1])): #Ako se nalazi na Y osi
                line += replaceToFloat(numbers[1]) + '\t'
            else:
                line += numbers[1] + '\t'
            if(re.search('E', numbers[2])): #Ako se nalazi na Z osi
                line += replaceToFloat(numbers[2]) + '\n'
            else:
                line += numbers[2] + '\n'
        rewrite += line
    file.close()
for count, filename in enumerate(os.listdir(path)): #Uzima sve nazive fajlova sa navedenih putanja
    match = re.match(r"([0-9]+)([a-zA-Z]+)", filename, re.I) #Deli naziv na brojeve i slova
    if(match):
        items = match.groups()
        if items[1]!="Output" and items[1]!="Recording": #Ako je fajl sa koordinatama
            convert(path,filename)
```

Слика 3.2.1. Програмски код за форматирање координата

Програмски код, наведен на слици 3.2.1. се користи тако што се у *path* променљиву унесе путања до директоријума који садржи све текст фајлове са подацима са жироскопа и акцелератора. У првом делу се извршава *for* петља која итерира кроз све фајлове у задатом фолдеру и враћа редни број и име фајла. Помоћу *re* библиотеке [4] правимо регуларни израз који детектује узастопни низ бројева па низ слова и групише. То радимо да би за сваки фајл који смо добили из урађених тестова проверили да ли је улазног типа тј. да ли садржи координате.

Функција *convert* отвара фајл и за сваку координату у сваком реду проверава да ли садржи експоненцијални формат писања. Уколико садржи, позива функцију *replaceToFloat* која враћа задату вредност у децималном формату коју уписујемо у нови ред. Уколико не садржи тражени формат, програм уписује у нови ред стару вредност. Када прође кроз све три координате уписује нови ред у остале редове које ћемо на крају уписати у отворен фајл.

Функција *replaceToFloat* конвертује вредност из експоненцијалног у децимални формат тако што прво детектује број децимала који тренутно има и број децимала који је сажет у

експоненцијални формат. Коришћењем уграђене функције *format* која форматира варијабле типа *string* враћамо вредност са бројем децимала који смо добили у претходном кораку.

3.2.2. Поравнавање дужина улазних фајлова

Један извршен тест траје 60 секунди са фреквенцијом од 200Hz, што значи да ће сваки фајл са подацима са сензора имати око 12000 редова. Обзиром да није могуће да сви фајлови имају тачно 12000 линија јер уређај не бележи увек тачно 200 вредности у секунди, у неким ситуацијама се дешава да имају мало мање или више од очекиваног броја.

Једно решење би било да фајлове који су преко 12000 линија исечемо а фајлове са мање редова да допунимо празним тј редовима испуњеним нулама. Друго решење би било да нађемо минималан број редова и да све остале фајлове смањимо до тог броја. На слици 3.2.2. је приказана имплементација другог решења.

```
#Racuna broj linija i ukoliko je manji od minimuma postavlja na minimum
def findMin(path,filename, min):
    file = open(path +"\\"+ filename, "r")
    lines = file.readlines()
    count = 0
    for line in lines:
        count = count + 1
    if(count < min):
        min = count
    file.close()
    return min

def alignFiles(path):
    with open(path, 'r') as fin:  #Otvori fajl za citanje
        data = fin.read().splitlines(True)
    with open(path, 'w') as fout: #Otvori fajl za pisanje i izbrisi sve podatke
        fout.writelines(data[0:11988]) #Upisi nove podatke

for count,filename in enumerate(os.listdir(path)):
    match = re.match(r"([0-9]+)([a-zA-Z]+)", filename, re.I)
    items = match.groups()
    if items[1]!="Output" and items[1]!="Recording":
        alignFiles(path+"\\"+filename)
```

Слика 3.2.2. Програмски код за поравнавање улазних фајлова

Функција *findMin* нам омогућава да лако дођемо до минималног броја редова итерирањем кроз фајлове и редове свих фајлова. Итерирање кроз фајлове се извршава као и у претходном примеру. Функција *alignFiles* отвара задати фајл, узима све редове, брише претходне податке и исписује запамћене редова при чему се узима само првих 11988 редова.

3.2.3. Фреквенција одабирања

Оригинална фреквенција одабирања од 200Hz је коришћена да бисмо могли да направимо засебне скупове података који ће имати 50 и 100Hz. Прављење скупова података са више различитих фреквенција одабирања нам служи да откријемо колика је најмања фреквенција која ће нам давати задовољавајуће резултате. Такође што је мања фреквенција одабирања, мањи је и број координата по фајлу што омогућава брже извршавање алгоритама, а самим тим и брже итерирање кроз различите параметре.

Први корак представља дуплирање оригиналног фолдера са подацима два пута, да би се добила три фолдера од којих ће сваки представљати податке са фреквенцијом одабирања 50,100 и 200Hz.

```
def takeEveryXRow(path,num):
    with open(path, 'r') as fin:
        data = fin.read().splitlines(True)
    with open(path, 'w') as fout:
        for x in range(0, len(data), num): #U zavisnosti od num, preskaci num-1 redova
            fout.writelines(data[x])

for count,filename in enumerate(os.listdir(path)):
    match = re.match(r"([0-9]+)([a-zA-Z]+)", filename, re.I)
    items = match.groups()
    if items[1]!="Output" and items[1]!="Recording":
        takeEveryXRow(path+"\\"+filename, 4)
```

Слика 3.2.3. Програмски код за мењање фреквенције одабирања

Након дуплирања фолдера, потребно је било применити горе наведени код са слике 3.2.3. на два нова фолдера. Функција *takeEveryXRow* функционише тако што од свих редова отвореног фајла узима сваки други или четврти ред у зависности од прослеђеног броја при позиву фукције. На излазу ће сваки фајл имати 2997 и 5994 редова за 50 и 100Hz, респективно.

3.2.4. Интезитет вектора

За улазне податке осим формата координата, коришћен је и интезитет вектора сваког реда координата. Као што је наведено у претходном потпоглављу смањење броја података повећава број могућих тестираних параметара због многоструко убрзаног рада алгоритма. Као и у случају смањивања фреквенције одабирања, коришћењем израчунатог интезитета вектора уместо x, y и z координате се губи један део информација, што значи да је током тестирања потребно тестирати обе комбинације за све фреквенције.

```
def create_vec(path,filename):
    #Otvori fajl sa koordinatama za citanje
    file1 = open(path +"\\"+ filename, "r")
    #Otvori fajl sa vektorima za pisanje
    file2 = open(path +"\\"+ "vector" + filename, "w")

    lines = file1.readlines()
    count = 0
    for line in lines:
        #svaki red odvoji po tabu ili novom redu da se dobiju koordinate
        numbers = re.split("\t|\n", line)
        #Izracunaj vektor po formuli sqrt(x^2 + y^2 + z^2)
        vector = math.sqrt(float(numbers[0])**2 + float(numbers[1])**2 +
            float(numbers[2])**2)
        file2.write(str(vector)+ '\r') #Upisi novi red

    file1.close()
    file2.close()

for count, filename in enumerate(os.listdir(path)):
    match = re.match(r"([0-9]+)([a-zA-Z]+)", filename, re.I)
    items = match.groups()
    if items[1]!="Output" and items[1]!="Recording":
        create_vec(path,filename)
```

Слика 3.2.4. Програмски код за рачунање интезитета вектора

Скупове података где су вредности интезитети вектора правимо тако што дуплирамо три већ постојећа скупа података са различитим фреквенцијама. На сваки нови фолдер примењује се програмски код дат на слици 3.2.4., који итерира кроз све фајлове у задатом фолдеру. Сваки фајл отвори и за сваки ред израчуна интезитет вектора и упише у нови текст фајл који има исти назив као и фајл са координатама осим што назив почиње са „vector“. Интензитет се рачуна по формули $vector = \sqrt{x^2 + y^2 + z^2}$.

3.2.5. Прављење мултидимензионих матрица

Да би убацивање у алгоритам било могуће, неопходно је извући податке из свих текст фајлова једног фолдера и убацити их у матрицу потребних димензија. Матрице излазних ознака ће увек бити димензија (315,1), јер за сваки тест имамо једну излазну ознаку. За улазне податке ће димензије матрице варирати у зависности од тога да ли користимо координате или векторе, а величина димензија ће зависити од фреквенце одабирања.

```
import numpy as np
def createCoordsInput(path, input_type):
    i = 0
    #Druga dimenzija zavisi od frekvence odabiranja
    #2997(50Hz) | 5994{100Hz} | 11988(200Hz)
    data_examples = np.zeros((315,2997,3))
    for count,filename in enumerate(os.listdir(path)):
        match = re.match(r"([0-9]+)([a-zA-Z]+)", filename, re.I)
        items = match.groups()
        #input_type moze da bude
        #Acceleration, GyrocopeRotRate ili GyroscopeAttitude
        if items[1]==input_type:
            with open(path+"\\"+filename, 'r') as fin:
                data = fin.read().splitlines(True)
            for x in range(0, len(data), 1):
                items = re.split("\t|\n",data[x])
                items = items[:len(items)-1]
                data_examples[i,x,0] = items[0]
                data_examples[i,x,1] = items[1]
                data_examples[i,x,2] = items[2]
            i = i+1;
    return data_examples
```

Слика 3.2.5. Програмски код за унос података у матрице

Функција *createCoordsInput*, приказана на слици 3.2.5., у зависности од типа улазног фајла и фреквенције одабирања на излазу креира тродимензионалну матрицу. Прва димензија матрице представља број примера, који је увек 315. Друга димензија представља број линија тј. координата по фајлу и она у зависности од фреквенције може бити 2997, 5994 или 11988 за 50, 100 или 200Hz респективно. Трећа димензија представља број координатних оса, што је у овом случају три.

Матрицу правимо помоћу библиотеке *numpy* [5] програмског језика *Python*. Библиотека *numpy* је веома битна за обраду података дубоких неуралних мрежа, зато што се готово увек користе матрице веома великих димензија или великог броја димензија, што ова библиотека подржава. Такође креирање или обављање било каквих операција над матрицама је веома ефикасно обзиром да *numpy* користи разне оптимизационе алгоритме за рад са матрицама.

Потребно је прво направити празну матрицу потребних димензија, што се извршава са *numpy* функцијом *zeros*. Та функција направи матрицу датих димензија и испуни је нулама. Затим отворимо фајл уколико је типа *input_type* који описује да ли су подаци у фајлу оријентација у простору, угаона брзина са жироскопа или убрзање по осама са акцелератора. Сви подаци унутар фајла су размакнути или ознаком за таб или ознаком за нови ред, као што је објашњено раније у овом поглављу. Користећи то, помоћу регуларних израза је лако могуће раздвојити редове података на координате и убацити их у претходно креирану матрицу.

```
def create_vector_input(path, input_type):
    i = 0
    #Kreirana matrica ima dve dimenzije
    #315 je broj testova, a 2997 je broj vektora po fajlu
    data_examples = np.zeros((315,2997))
    for count,filename in enumerate(os.listdir(path)): #za svaki vektor
                                                                fajl
        #regex
        match = re.match(r"(vector)([0-9]+)([a-zA-Z]+)", filename, re.I)
        items = match.groups()
        #input_type moze da bude
        #Acceleration, GyrocopeRotRate ili GyroscopeAttitude
        if items[2]==input_type:
            with open(path+"\\"+filename, 'r') as fin:
                data = fin.read().splitlines(True)
            for x in range(0, len(data), 1):
                data_examples[i,x] = data[x]
            i = i+1;
    return data_examples
```

Слика 3.2.6. Програмски код за унос вектора у матрицу

Као и у претходном примеру функција за убацивање података из фајлова са вредностима које представљају интензитет вектора, са слике 3.2.6., прво прави матрицу испуњену нулама. У овом случају трећа димензија није потребна јер уместо три координате по сваком реду имамо само један интензитет вектора.

```
def create_output(output_type):
    #Putanja je fiksirana jer je output svugde isti
    path = r"D:\Users\Leon Jovanovic\Desktop\GyroData\Data50Hz"
    data_labels = np.zeros((315,1))
    j=0
    #output_enums je funkcija koja u zavisnosti od
    #testa koji zelimo da izvrsimo vraca redni broj
    out = output_enums(output_type)
    for count,filename in enumerate(os.listdir(path)):
        match = re.match(r"([0-9]+)([a-zA-Z]+)", filename, re.I)
        items = match.groups()
        if items[1]=="Output":
            with open(path+"\\"+filename, 'r') as fin:
                data = fin.read().splitlines(True)
            items = re.split("\t|\n",data[1])
            #language_mapping mapira svaki string jezika
            #u redni broj od nula do dva
            data_labels[j,0] = language_mapping(items[out])
            j = j+1;
    return data_labels
```

Слика 3.2.7. Програмски код за унос излазних података у матрицу

Код уписивања излазних ознака, у посебну матрицу се извршава исти код, на слици 3.2.7., невезано да ли је улазни податак координата или вектор, као и да ли је фреквенца одабирања 50, 100 или 200Hz. Из истог разлога је и путања до фајлова излазних ознака фиксирана на један од фолдера. Као што можемо видети на слици 3.1.3 датотека са излазним ознакама има 13 потенцијалних вредности одвојени табовима, стога је потребно израчунати

које место треба да гледамо у зависности од жељеног теста. Функција *output_enums* управо то извршава и враћа бројеве од 0 до 12.

Након што смо излазне ознаке претворили у низ стрингова, узимамо добијено место у низу и прослеђујемо функцији *language_mapping* која у зависности од језика враћа број од 0 до 2 и на тај начин се матрица попуњава само тим вредностима. Коришћена је функција *language_mapping* зато што у горе наведеном коду је потребно да направимо матрицу са излазним ознакама за језик да би алгоритам покушао да научи о ком језику се ради на основу улазних података. Поред горе наведене функције постоје и функције *years_mapping*, *words_mapping*, *noise_mapping* и *distance_mapping* које раде на сличном принципу мапирања улазног стринга у број од 0 па навише.

Ради боље прегледности и веће ефикасности кода горе наведене функције су енкапсулиране у две функције. Оне ће бити позиване из друге Python датотеке након импортовања дате датотеке као што се може видети на слици 3.2.8.

```
def create_dataset(path, input_type, output_type):
    data_examples = create_input(path, input_type)
    data_labels = create_output(path, output_type)
    return (data_examples, data_labels)

def create_vector_dataset(path, input_type, output_type):
    data_examples = create_vector_input(path, input_type)
    data_labels = create_output(path, output_type)
    return (data_examples, data_labels)
```

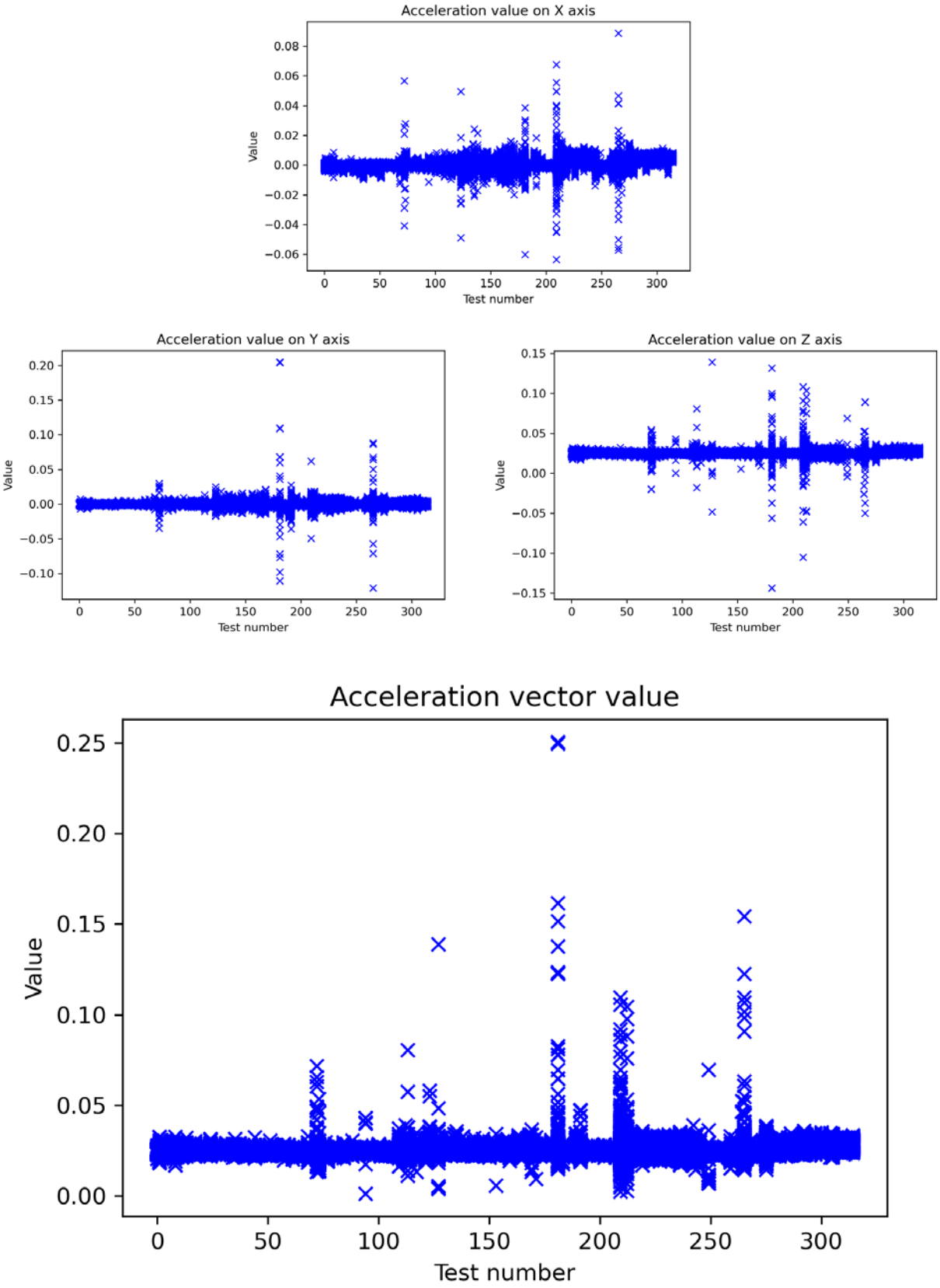
Слика 3.2.8. Програмски код за прављење скупа података

Као што се види на слици 3.2.9., након тога је могуће правити скупове података са различитим улазним и излазним подацима као и са различитим фреквенцијама одабирања. Креирају се варијабле *data_input_norm*, *data_output_norm*, *data_vector_input_norm* и *data_vector_output_norm* које представљају матрице које ће бити обрађиване у наставку текста.

```
from CreateDataset import *
path1 = r"D:\Users\Leon Jovanovic\Desktop\GyroData\Data50Hz"
path2 = r"D:\Users\Leon Jovanovic\Desktop\GyroData\VectorData50Hz"
pathOut = r"D:\Users\Leon Jovanovic\Desktop\GyroData\Results"
freq = 50
input_type = "Acceleration"
output_type = "#Pers"
#Input types: "Acceleration|GyroscopeRotRate|GyroscopeAttitude
#Output types: BgNoise|Lang|Dist|#Pers|#M|#F|Y(Y1    Y2    Y3    Y4    Y5
      Y6)|L/W(Letter/Num/Word..)
data_input_norm, data_output_norm = create_dataset(
                                                   path1,
                                                   input_type,
                                                   output_type
                                                   )
data_vector_input_norm, data_vector_output_norm = create_vector_dataset(
                                                                        path2,
                                                                        input_type,
                                                                        output_type
                                                                        )
```

Слика 3.2.9. Програмски код за позивање функције која прави скуп података

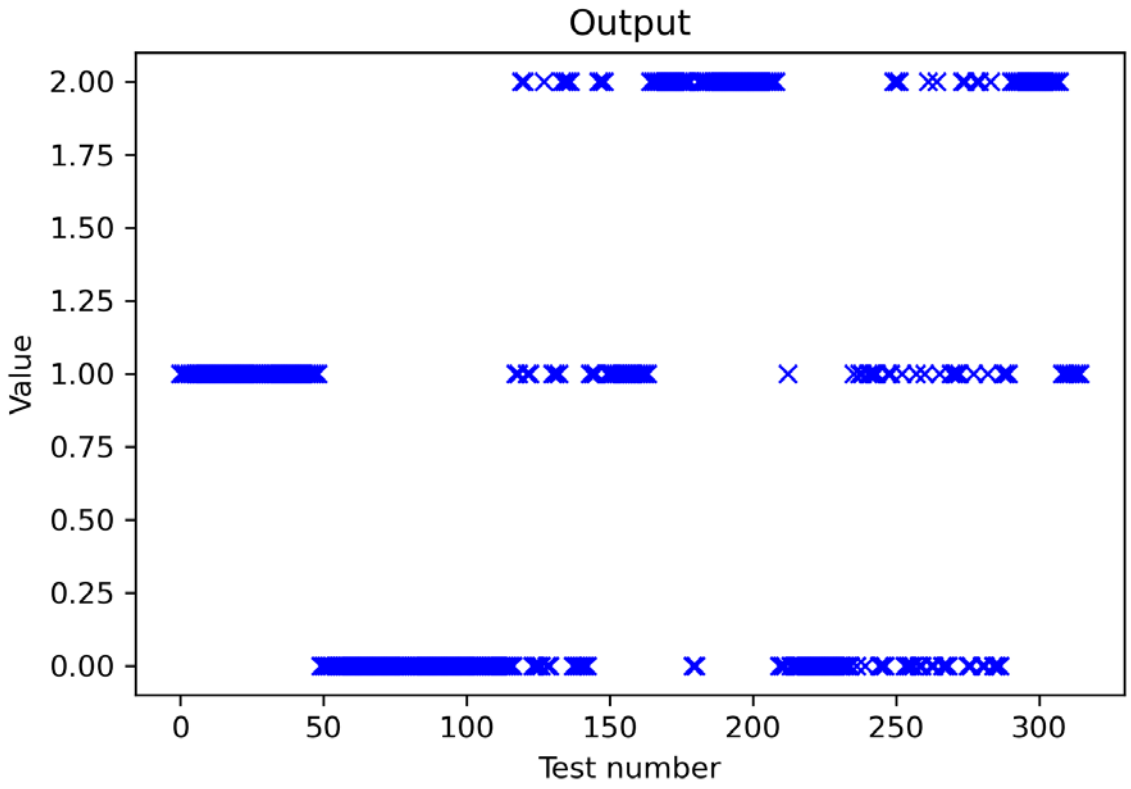
Сада можемо да искористимо направљене матрице да исцртамо податке на графику. Обзиром да *x*, *y* и *z* координате су међусобно независне, нема потребе да их приказујемо заједно на *3D* графику, стога ће се оне приказивати као и вектори на *2D* графику. На слици 3.2.10. можемо видети изглед улазних података акцелерометра са фреквенцијом 50Hz. Они представљају координате података који су прочитани током свих 315 тестова. Као што се може видети на слици, њихове вредности су веома мале, због чега ћемо их у наставку текста скалирати по осама.



Слика 3.2.10. Графички приказ улазних података акцелерометра на 50Hz

На слици 3.2.11 можемо видети излазне ознаке за проверу препознавања језика. Нула представља српски, један енглески а два немачки. Као што можемо приметити, примери су прилично груписани што није неочекивано кад се узме у обзир да су тестови прикупљани по одређеном редоследу. У наставку текста ћемо причати о томе како измешати податке да би избегли овакво груписање које доста штети учењу алгоритма.

Пример са слике 3.2.11. представља графички приказ свих тестова по x оси и њихових излазних вредности за језик по y оси. X оса иде од 0 до 315 (број тестова), а y оса су вредности 0 (српски), 1 (енглески) и 2 (немачки).



Слика 3.2.11. Графички приказ излазних ознака за препознавање језика

3.2.6. Мешање података

Иако су улазни подаци прилично разноврсни, уколико у алгоритам убацимо податке који нису измешани, већ их убацујемо редом како су прављени, може доћи до лоших резултата. То се дешава јер је могуће да алгоритам учи на једном делу података који су обележени само једном или са две различите ознаке (нпр. српски и енглески), а тестира прецизност на подацима који су обележени трећом ознаком (нпр. немачки). У таквој ситуацији није могуће очекивати од алгоритма да научи нешто што није ни видео током тренирања параметара.

Да би избегли такву ситуацију потребно је измешати целокупан скуп података. Иако су нам скупови улазних и излазних података два одвојена скупа, битно је да буду измешани на исти начин да би осигурали да улазни податак на одређеној локацији одговара излазној ознаци на истој локацији.

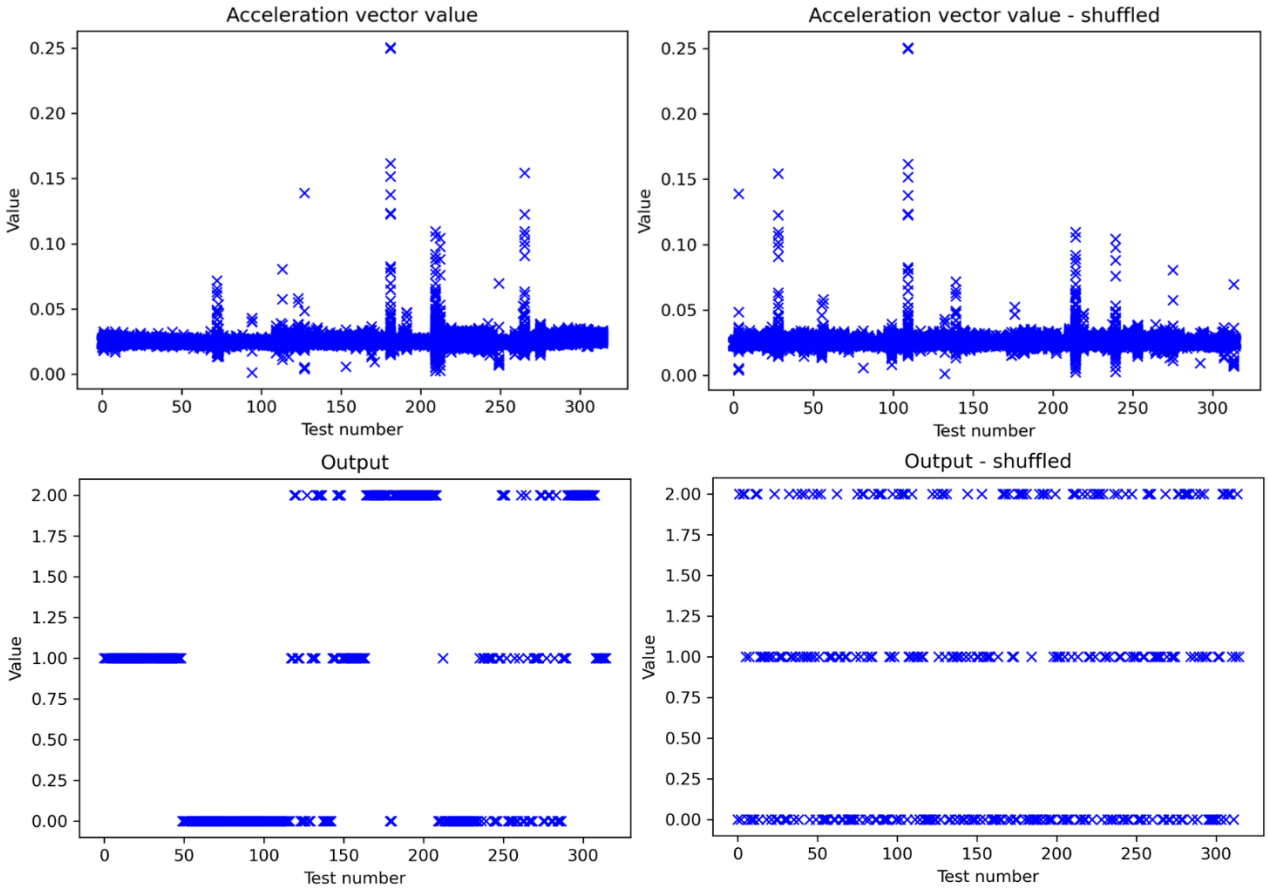
То смо осигурали коришћењем библиотеке *sklearn.utils* [6] која садржи функцију *shuffle*. Функција *shuffle* за аргумент може да прими две различите матрице као и број који представља клицу. Клица представља број који се даје генератору псеудослучајних бројева да би он могао да генерише следеће бројеве. Користећи исти редни број клице, функција на слици 3.2.12. ће промешати обе матрице по истом редоследу као што је и потребно.

```
from sklearn.utils import shuffle
rnd = 3
data_input_norm, data_output_norm = shuffle(
                                            data_input_norm,
                                            data_output_norm,
                                            random_state=rnd
                                            )

data_vector_input_norm, data_vector_output_norm = shuffle(
                                                    data_vector_input_norm,
                                                    data_vector_output_norm,
                                                    random_state=rnd
                                                    )
```

Слика 3.2.12. Позив функције за мешање

Број клице је узет насумично и могао је да буде било који други број. Резултате мешања можемо видети на слици 3.2.13. где се јасно види да не постоји превелико груписање међу тестовима, што значи да смо постигли жељени успех.

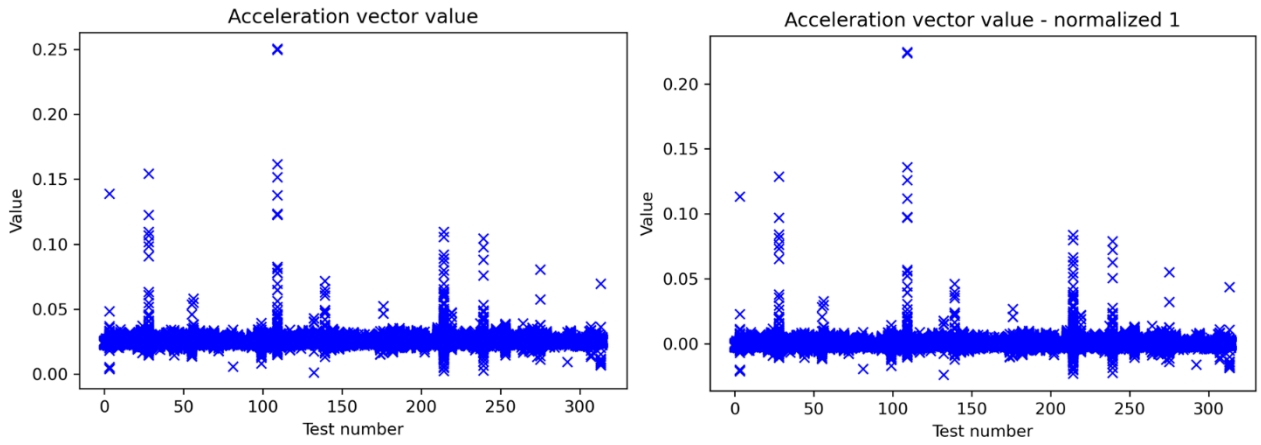


Слика 3.2.13. Графички приказ података пре и после мешања

3.2.7. Нормализација података

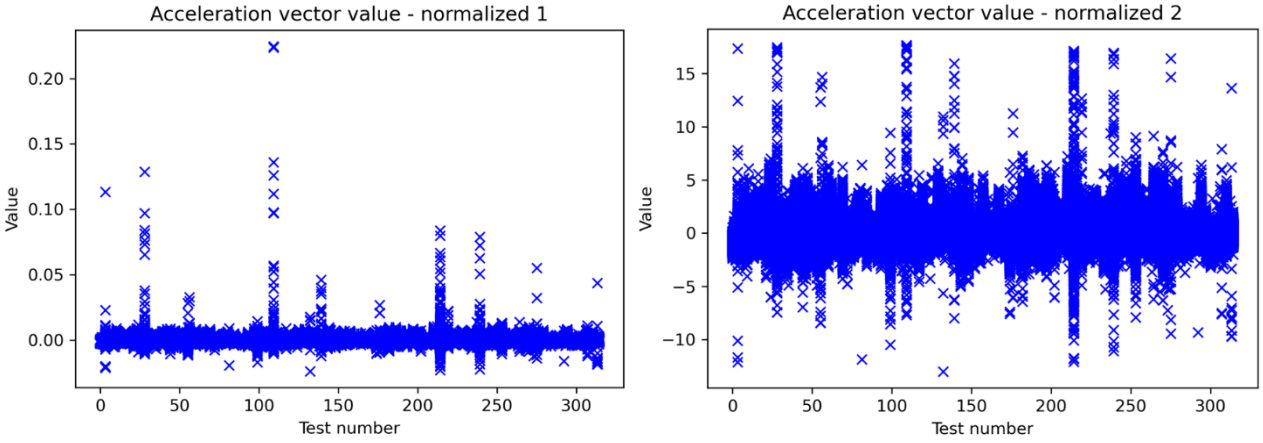
У теорији није неопходно вршити нормализацију на подацима, посебно ако има само једна особина као код интензитета вектора, али у пракси се показало да нормализација побољшава перформасе дубоких неуралних мрежа. У ситуацијама где има више од једне особине, у нашем случају координате *x*, *y* и *z*, потребно је нормализовати да би се све три независне координате свеле на исте границе. Уколико се то не уради, могуће је да ће промена тежина у неуралној мрежи током рада алгоритма више утицати на једну особину а мање на другу особину и самим тим довести до лошијих резултата.

Постоје више различитих техника нормализације података, а једна од најчешћих и она која је коришћена у овом раду је нормална расподела. Нормална расподела се рачуна у два корака. Први корак је рачунање просека свих вредности по једној оси $\mu = \frac{1}{m}\sum_1^m x^{(i)}$, а затим његово одузимање од улазних података по тој оси $x = x - \mu$. То доводи до тога да се подаци центрирају у координатном почетку осе по којој смо сабирали. Разлика се може приметити на слици 2.3.14. где нема промене у самом изгледу скупа података, али ако се погледају осе види се да сада већина података се налази око нуле.



Слика 3.2.14. Графички приказ пре и после првог дела нормализације

Други корак је рачунање просека квадрираних вредности скупа података по једној оси (квадрирање се врши по елементу матрице) по формули $\sigma^2 = \frac{1}{m}\sum_1^m x^{(i)^2}$, а затим кореном добијене вредности делимо скуп података добијен из претходног корака $x = \frac{x}{\sigma}$. Другим кораком смо постигли да раширимо скуп података по оси где су подаци били збијени, што се види и на слици 3.2.15.



Слика 3.2.15. Графички приказ пре и после другог дела нормализације

Имплементација нормализације није комплексна па самим тим је пожељно радити уколико је то могуће обзиром да убрзава алгоритам дубоких неуралним мрежа. Као што је већ наведено уколико постоји само једна особина у подацима, као што је случај код интезитета вектора, није толико неопходно, али постоје могућност да алгоритам има боље резултате над нормализованим подацима.

Ради тестирања обављено је учење дубоке неуралне мреже са одређеним параметрима на улазним подацима који нису били нормализовани и на подацима који су били нормализовани. За сваки скуп података је вршено 10 учења да би се нашао максималан и просечан резултат. У доле наведеним резултатима се тестирало препознавање језика са векторским подацима акцелерометра на фреквенци од 50Hz.

Табела 3.2.1. Резултати алгоритма са и без нормализације

	Функција губитка	Најбоља прецизност	Просечна прецизност
Без нормализације	1.115	34.92%	34.92%
Са нормализацијом	1.266	46.03%	39.20%

Као што се може приметити резултати без нормализације су слични насумичном одабирању тј. 33.33%, док у случају када се примени нормализација остварују се значајно бољи резултати.

3.2.8. Тренинг, валидациони и тест скуп

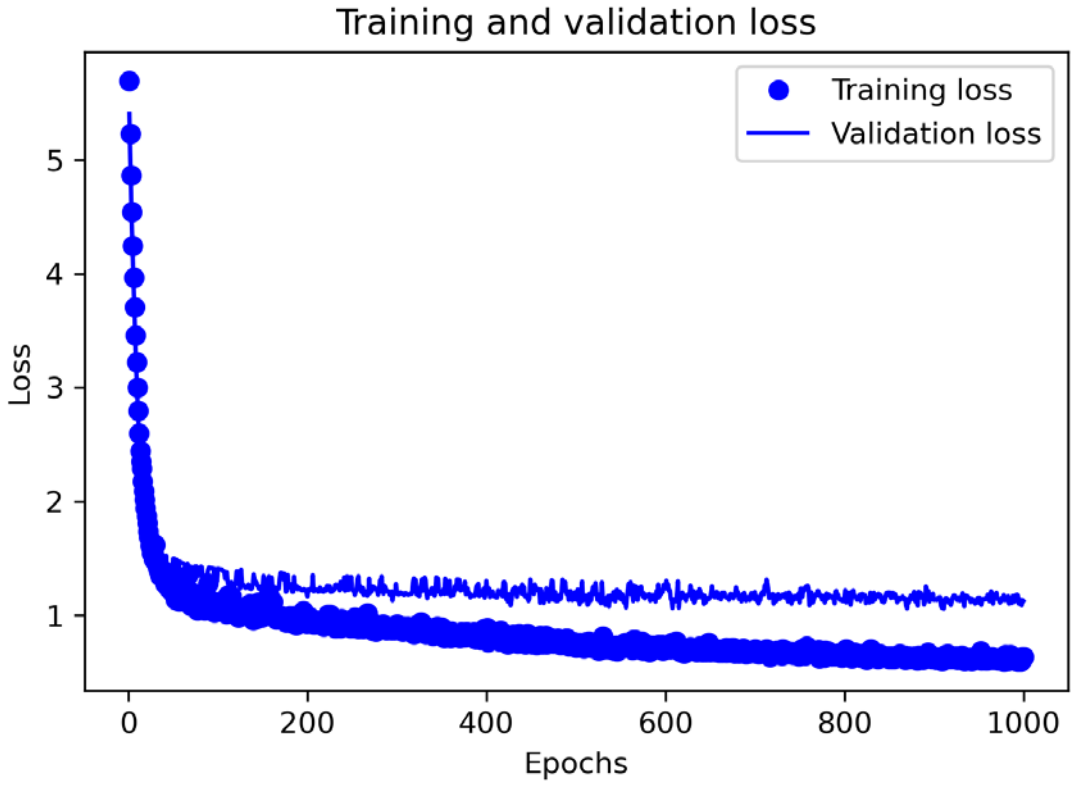
Након мешања и нормализације скупа података, подаци су спремни за поделу на тренинг, валидациони и тест скуп. Уколико се то не би радило, већ би се и учење и тестирање извршавало на истом скупу података, резултати би највероватније били веома добри, али нетачни. У том случају, алгоритам ће веома добро научити скуп података који му дамо и резултати тестирања на истом скупу ће бити прецизни, али ако га тестирамо на подацима ван нашег иницијалног скупа, резултати ће бити веома лоши. Да бисмо добили тачан резултат, на крају учења, алгоритам треба да тестира истрениран модел на скупу података који до сада није видео, али да тај скуп података долази од истих тестова као и тренинг скуп на коме је алгоритам учио.

Због горе изложених проблема, скуп података се мора поделити на тренинг и тест сет. Уколико се користе само та два скупа података, постоје велике шансе да ће се на крају тренирања десити оверфитинг (енгл. *overfitting*) или андерфитинг (енгл. *underfitting*).

Оверфитинг је појава када истренирани модел сувише добро научи податке који су му дати за тренирање, стога када тестира на тест скупу резултати ће бити знатно лошији. Ова појава се најчешће решава прикупљањем још података или регуларизацијом о којој ћемо прицати у наставку рада.

Андерфитинг је појава када модел не научи довољно добро на тренинг скупу па резултати буду лоши и након тренирања и тестирања. Андерфитинг се решава повећањем комплексности дубоке неуралне мреже (већи број скривених јединица или слојева) или да пустимо алгоритам да дуже учи. Такође је могућа ситуација где алгоритам не може да научи иако је довољно комплексан и довољно дуго ради. У таквој ситуацији решење је најчешће применити други алгоритам или другу архитектуру неуралних мрежа (конволуциону, рекурентну...).

Да бисмо што лакше уочили оверфитинг или андерфитинг, уместо да чекамо крај целог тренирања да бисмо тестирали на тест скупу, потребно је направити још један скуп од иницијалног скупа – верификациони скуп. Он служи да након сваке епохе тренирања провери досад истренирани модел на верификационом скупу. Када графички прикажемо напредак на тренинг и верификационом скупу, можемо јасно распознати да ли је андерфитинг или овертинг као што се може видети на слици 3.2.16.



Слика 3.2.16. Графички приказ оверфитинга

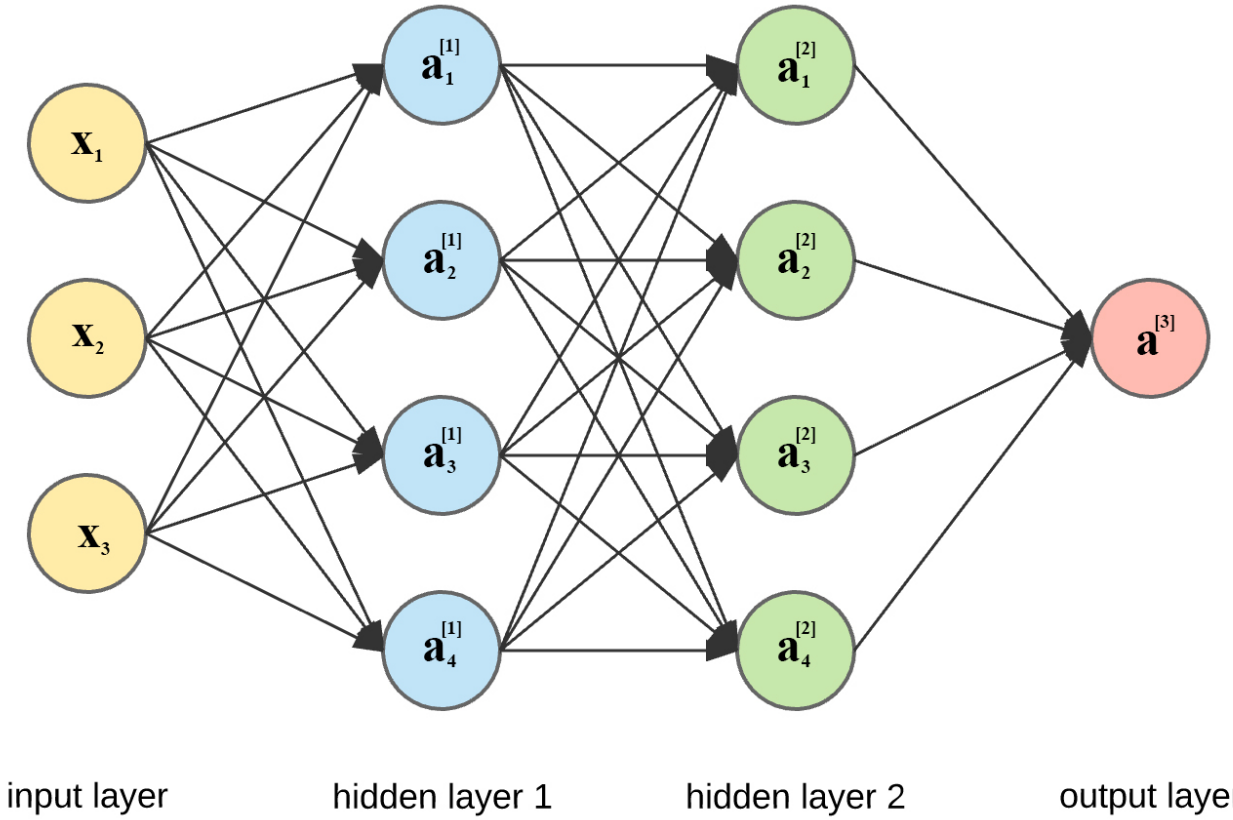
Обзиром да имамо мали број тестова (315) подела на тренинг, валидациони и тест скуп се извршава по 60, 20 и 20% респективно. То значи да ће тренинг скуп имати 189 , валидациони скуп 63 а тест скуп 63 примера.

3.3. Дубока неуралнa мрежа

Ово поглавље ће се бавити подешавањима свих параметара дубоке неуралне мреже као и њиховим међусобним односима. Детаљно ће бити објашњено када се који оптимизациони алгоритми, активационе функције и други параметри користе ради повећања успешности алгоритма у зависности од циља који желимо да постигнемо.

3.3.1. Неурална мрежа

Један од најпопуларнијих алгоритама машинског учења представљају неуралне мреже. Оне представљају мрежу „неурона“ која обухвата улазни слој, скривене слојеве и излазни слој. Пример са слике 3.3.1. представља типичну неуралну мрежу са два скривена слоја.

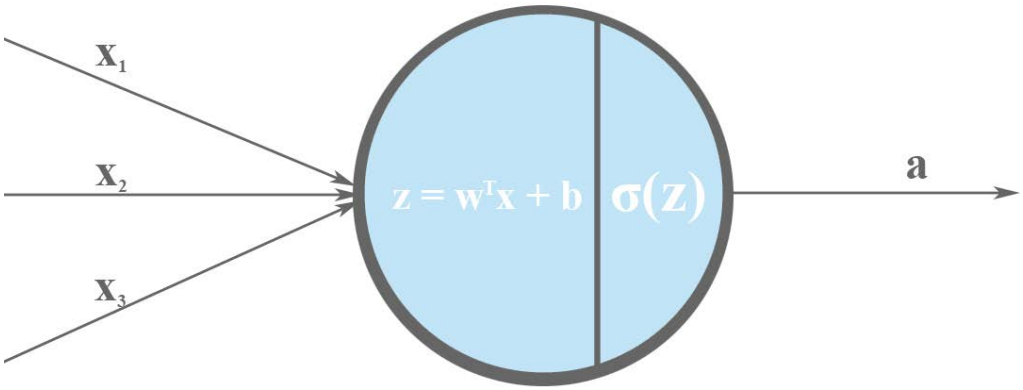


Слика 3.3.1. Пример неуралне мреже

При уласку у једну јединицу, улазни подаци се убацују у функцију $w^T x + b$ где су w и b специфични параметри сваке скривене јединице сваког скривеног слоја. Након израчунавања функције, резултат се убацује у активациону функцију да би се омогућило добијање нелинеарне функције на излазу. Резултат активационе функције се шаље или у следеће скривене јединице или у излазне јединице уколико је у питању последњи скривени слој. То се може видети илустровано на слици 3.3.2.

Неуралнa мрежа функционише тако што се у улазни слој убаце подаци из тренинг скупа, затим прослеђују свакој јединици у следећем скривеном слоју, а када скривена јединица првог слоја прими податке, она прослеђује свакој јединици следећег слоја и тако док се не дође до излазног слоја који на излаз шаље предикцију. То представља једну итерацију пропагације унапред.

Када се добије на излазу предикција, рачуна се функција губитка која показује колико је резултат неуралне мреже далеко од излазне ознаке тј. очекиваног резултата. Када се израчуна функција губитка, то се користи да бисмо добили нове вредности параметара w и b свих скривених јединица кроз које смо прошли у пропагацији унапред. Оне се рачунају „ходом уназад“ тј. изводом функције губитка по параметрима w и b се добија померај, који се множи са α што представља корак учења, а затим се тај производ одузима од старих вредности w и b да би се добиле нове вредности. Ово представља једну итерацију пропагације уназад.



Слика 3.3.2. Графички приказ једне скривене јединице

Понављањем пропагације унапред и уназад доводи вредности параметара w и b до оптималних вредности које резултују у што мањој функцији грешке или што већем проценту успешности.

Дубоке неуралне мреже представљају комплекснију архитектуру обичних неуралних мрежа где број скривених јединица по слоју је веома велики, док број слојева може да буде и двоцифрен. Комплексније архитектуре омогућавају да алгоритам реши неке теже проблеме као што је тема овог дипломског рада.

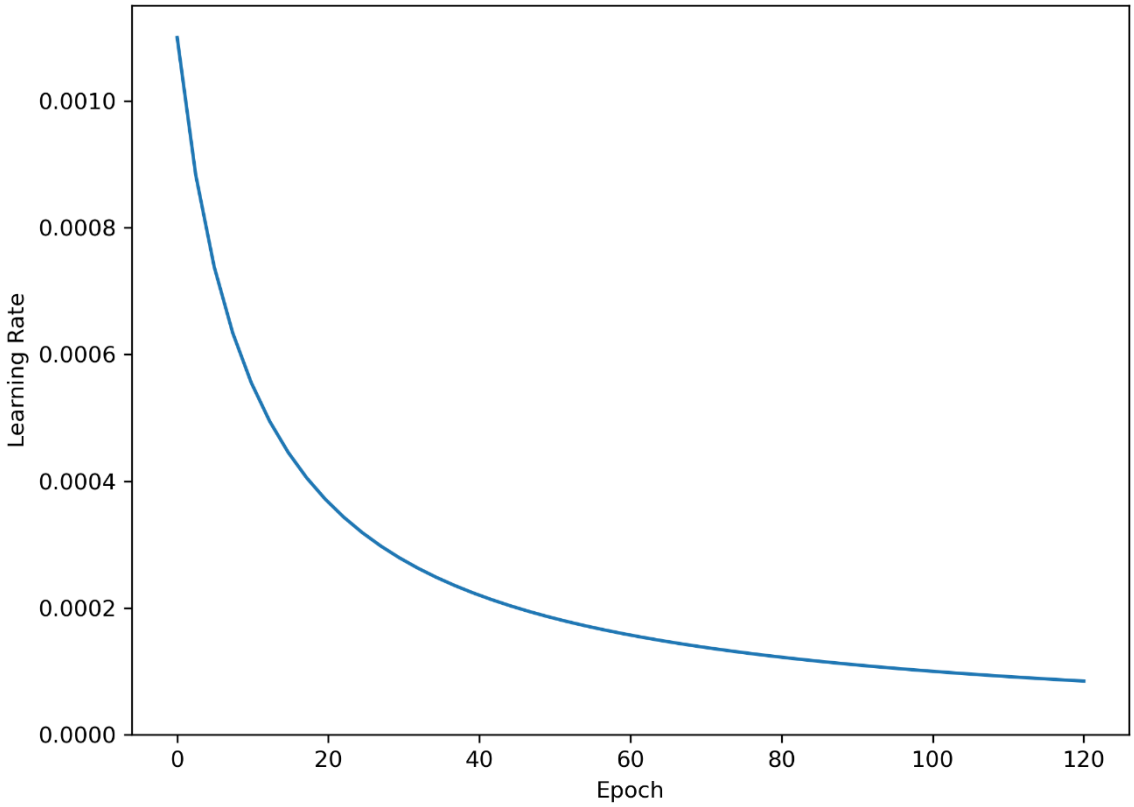
Рад са дубоким неуралним мрежама одликује подешавање великог броја параметара тзв. хиперпараметара који описују једну архитектуру. Најважнији хиперпараметри су корак учења, оптимизациони алгоритам, број скривених слојева, број скривених јединица, степен опадања корака учења и величина мини-серија. Иако су горе наведени параметри најбитнији, постоје још доста параметара који утичу на резултат алгоритма.

3.3.2. Степен опадања корака учења

Корак учења је најбитнији хиперапараметар који се подешава у раду са неуралним мрежама. Он представља колики корак ћемо направити након сваке епохе ка тачном циљу.

Већи корак омогућава брже конвергирање функције губитака ка нули али постоји опасност да буде превелики и да „промаши“ глобални минимум. Са друге стране ако је корак учења премали онда је потребно велики број итерација да би нашао локални или глобални минимум.

Такође чак и ако је корак учења оптималан, може да се деси да не може да погоди минимум па се константно ротира око њега. Да би се то избегло уводи се степен опадања који, како се број епоха повећава, величина корака учења се смањује и на тај начин омогућава брзо проналажење глобалног минимума на почетку и сигурно конвергирање ка минимуму малим корацима на крају алгоритма. Визуелни приказ се може видети на слици 3.3.3.



Слика 3.3.3. Графички приказ опадања корака учења са порастом броја епоха

3.3.3. Регуларизација

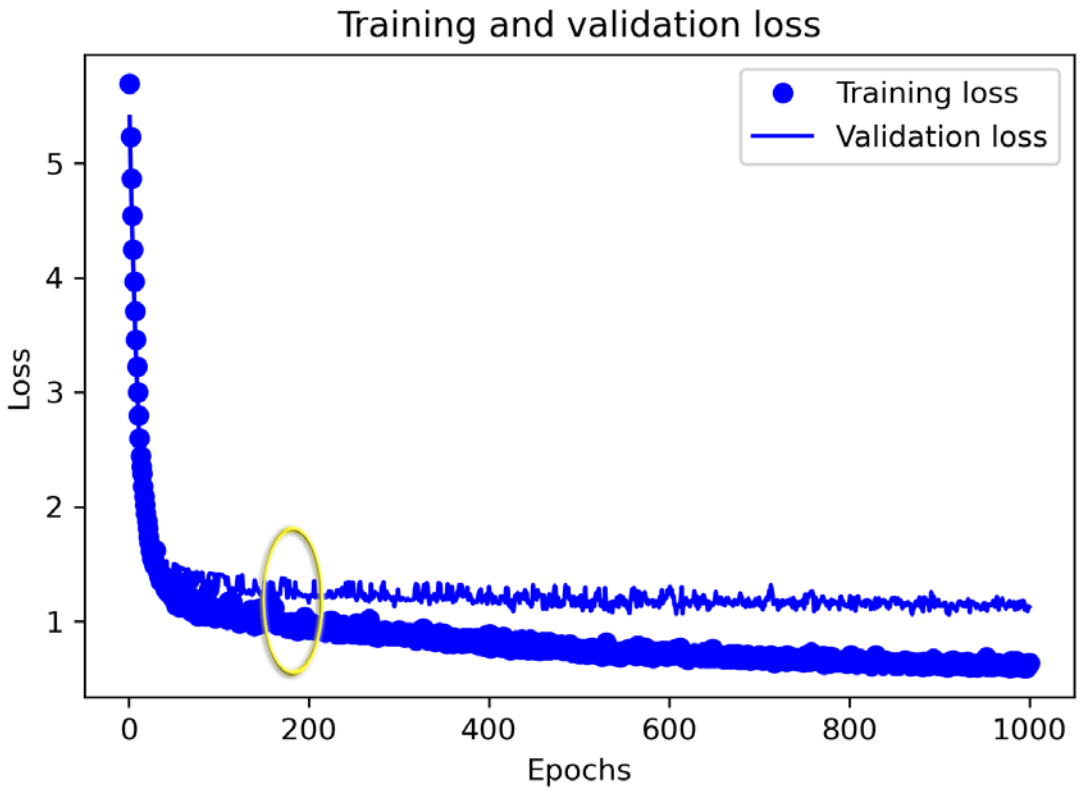
Као један од решења оверфитинга, регуларизација је доста битан појам код дубоких неуралних мрежа. Регуларизација представља технику која се користи за смањивање грешке одговарајућим уклапањем функције на дати тренинг скуп да би избегли превелико уклапање функције тзв. оверфитинг. Постоји више техника за регуларизацију а у овом дипломском раду су се користиле две или три технике у исто време.

L2 регуларизација је најпопуларнија и своди се на то да на крају сваке пропагације унапред, при рачунању функције губитка, дода вредност која зависи од параметра λ (*lambda*). Додавањем вредности на фнкцију губитка постиже да се функција упрости и самим тим спречава оверфитинг.

Dropout регуларизација је техника насумично гашење неких скривених јединица, што доводи до тога да се смањи комплексност архитектуре, а самим тим и претерано уклапање

функције. Параметар dropout регуларизације тј. проценат скривених јединица који ће бити угашен по слоју је обично 0.5 (50%) али може и да се мења у зависности од проблема.

Early stopping регуларизација је техника која прекида учење алгоритма у моменту када праћена вредност крене да се повећава или смањује. Најчешће се користи ако постоји ситуација где након одређеног броја епоха, функција губитка валидационог скупа се не спушта, а функција губитка тренинг скупа наставља да се смањује што је јасан показатељ оверфитинга. У тим ситуацијама, учење се прекида и узимају се или последње добијене вредности параметара *w* и *b* или њихове најбоље добијене вредности. Пример ове технике се може видети на слици 3.3.4. где је заокружен моменат када би ова техника зауставила рад алгоритма.



Слика 3.3.4. Графички приказ *Early Stopping* технике на примеру са оверфитингом

3.3.4. Оптимизациони алгоритми

Како мењати параметре *w* и *b* и корак учења α да би се функција грешке смањивала, је дефиниција рада оптимизационих алгоритама. Најшире коришћен је *SGD* (*Stochastic gradient descent*) при чему се у већини случајева користи унапређена верзија са моментом. Поред алгоритма *SGD*, укратко ће бити описани и *RMSprop* и *Adam*.

SGD са моментом је алгоритам који има параметар β (момент) који у суштини представља колико вреднујемо претходну вредност помераја и обично је 0.9. Када добијемо померај, помножимо га са 1- β и саберемо са производом β и старог помераја. Нови померај помножимо са кораком учења и одузимамо од старе вредности параметара.

RMSprop алгоритам је структурно сличан алгоритму *SGD* али је рачунање помераја математички другачије. Своди се на квадрирање добијеног помераја, затим множење са 1- β, сабирање са производом β и старог помераја и на крају дељење са кореном старог помераја пре множења са кораком учења и одузимање те вредности од старе вредности параметара.

Adam функционише као мешавина претходна два алгоритма, где рачуна на оба начина помераје а затим подели померај алгоритма *SGD* са кореном помераја алгоритма *RMSprop*, а затим помножи са кораком учења и одузме од старе вредности параметара. У овом раду је такође коришћен и *AdaMax* [7] алгоритам који је верзија алгоритма *Adam* са бесконачном нормом.

3.3.5. Мини-серије

Код података са веома великом количином примера користе се мини-серије (енгл. mini-batch) да би поделили податке на групе и те групе обрађивали независно. То доводи до тога да се алгоритам извршава брже јер не мора да пролази кроз све примере сваки пут, а углавном не губи на успешности.

Генерално се за ситуације где постоји испод 2000 примера не ради имплементација мини-серија, а иако овај рад има 315 примера и требало би да припада тој категорији, у неким ситуацијама је било потребно користити мини-серије. Разлог за коришћење мини-серије је то што током рада алгоритма на једном лаптопу, после 10, 15 итерација *Tensorflow* пријављивао грешку да нема више меморије. То се дешавало зато што *Tensorflow* није ослобађао меморију када заврши једно учење о чему ће се детаљније причати у наредном потпоглављу. Како је имплементација мини-серија није утицала на прецизност добијених модела, примењене мини-серије су и убрзале алгоритам као и смањиле шансу да алгоритам престане са радом услед губитка меморије.

3.3.6. Код алгоритма

Да би тестирање свих могућих комбинација хиперпараметара прошло што лакше, потребно је:

- прављење целе архитектуре
- компајлирање модела
- уклапање модела за тренирање
- дефинисање *Early Stopping callback* функције
- дефинисање степена опадања корака учења
- одабир оптимизационог алгоритма

и све друге параметре параметризовати унутар једне функције, као што се може видети на слици 3.3.5. Такође, и сам позив функције је потребно окружити *for* петљама да би се нашла максимална и просечна вредност резултата.

```
s = train(
        totalLoops = 10,  #Broj puta koliko ce uciti isti model
        activF = 'relu',  #Aktivaciona funkcija skrivenih jedinica
        i_size = 2997,    #Izravnan broj ulaznih podataka u prvom sloju
        numLayers = 3,    #Broj skrivenih slojeva nakon prvog sloja
        numUnits = 1024,  #Broj skrivenih jedinica u svakom sloju
        outputUnits = 3,  #Broj jedinica u output sloju
        alfa = 0.0011,    #Velicina koraka ucenja
        optFlag = True,   #Koristi stepen opadanja koraka ucenja (True)
        optim = "rmsprop",                #Optimizacioni algoritam
        pat = 11,                         #Early stopping strpljenje
        train_e = train_vector_examples,  #Trening set ulaznih podataka
        train_l = train_vector_labels,    #Trening set izlaznih oznaka
        test_e = test_vector_examples,    #Test set ulaznih podataka
        test_l = test_vector_labels,      #Test set izlaznih oznaka
        epoch = 120,        #Broj epoha
        verb = 0,           #Ispisuj preciznost I fiju gubitka nakon epohe
        mini_batch = 63     #Podeli u mini-serije velicine
        )
```

Слика 3.3.5. Програмски код за позивање функције за тренирање модела

Такође је потребно графички приказати функцију губитка да би могло да се види како који параметар утиче на резултате алгоритма. Код целе функције може се видети у прилогу овог документа. Итерирањем кроз различите хиперпараметре долазимо до најбољих резултата које након 10 итерација учења памтимо најбољу и просечну прецизност. Заједно са вредношћу функције губитка и свим параметрима датог модела, пишемо ове две вредности у текст фајл одвојено међусобно табовима ради лакшег импортовања у *Microsoft Excel*. Функцију позивамо укупно три пута, што значи да бележимо три различита резултата по мењању параметара ради повећане тачности резултата.

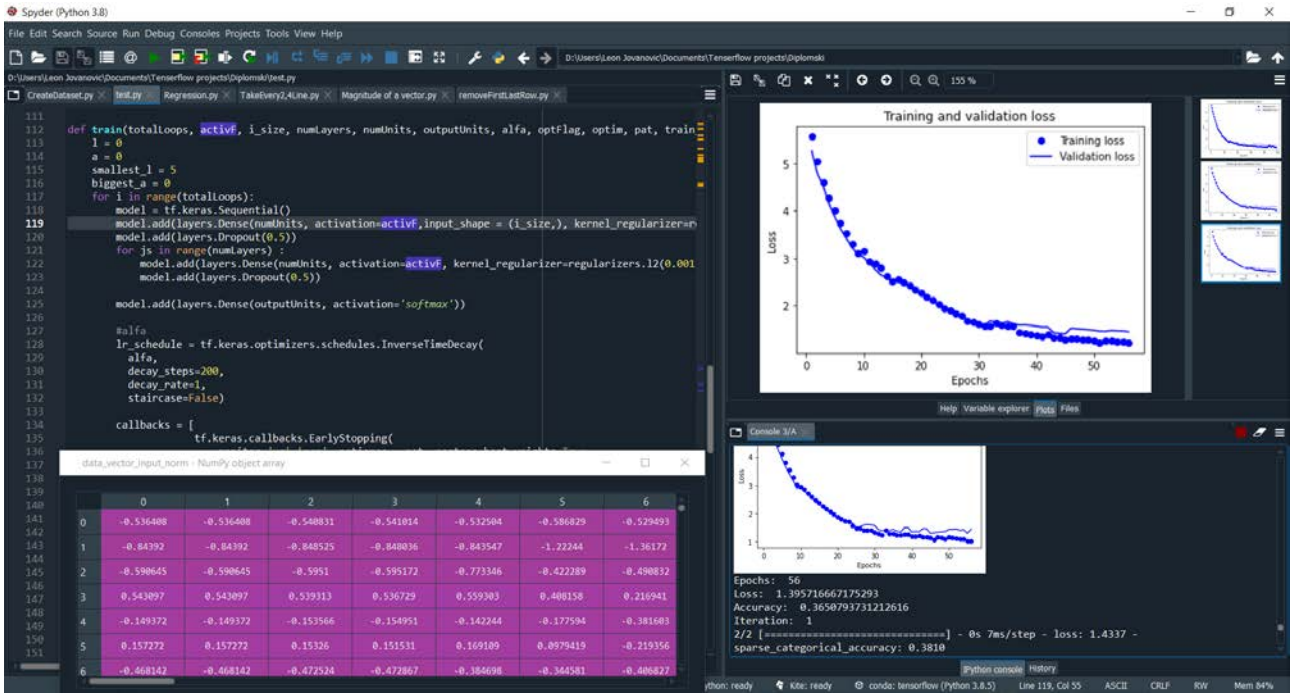
3.4. Коришћени програми и уређаји

У наставку потпоглавља ће бити детаљно описани окружења, програми, библиотеке и хардверски уређаји који су били коришћени током овог истраживачког рада. Коришћен је програм *Anaconda* [8] и *Spyder* [10], библиотеке *Tensorflow* и графичке картице *Nvidia GeForce 940MX* [14] и *Nvidia GeForce MX150* [15].

3.4.1. Anaconda – Spyder

Дипломски рад је рађен у виртуалном окружењу направљеном у програму *Anaconda*. У датом виртуелном простору су инсталиране потребне библиотеке, међу којима је *matplotlib* [9] коришћена за графички приказ података, као и за графички приказ корака учења и функције губитка.

У направљеном виртуалном простору је инсталиран програм *Spyder* који је коришћен као извршно окружење где се користи *Python* програмски језик. Најчешће се користи у истраживачким радовима јер је практичан за сагледавање коришћених променљивих и графика.



Слика 3.4.1. *Spyder* извршно окружење

3.4.2. Tensorflow

Tensorflow је *open-source* библиотека за напредне математичке функције, али има широку примену у машинком учењу. Најчешће се користи за прављење и коришћење дубоких неуралних мрежа. Од верзије 2.0 користи библиотеку *Keras* [11] на површини, па је самим тим преузео и синтаксу те библиотеке. *Keras* је библиотека ексклузивно прављена за неуралне мреже.

Обзиром да најновија верзија 2.3.0 има познату грешку који јој онемогућује да буде покренута на графичкој картици коришћеној за потребе овог рада, коришћена је претходна верзија *tensorflow-gpu 2.2.0rc2*. *Tensorflow* званично подржава само *Nvidia* графичке картице, па је било потребно инсталирати *CUDA Toolkit* [12], што представља окружење за прављење апликација које користе *Nvidia* графичку картицу, као и *cuDNN* [13] библиотеку за коришћење графичке картице за дубоке неуралне мреже.

```
model = keras.Sequential([
      layers.Dense(512, activation='relu', input_shape(input_size,)),
      layers.Dense(512, activation='relu'),
      layers.Dense(1, activation='sigmoid')
])

loss = tf.keras.losses.BinaryCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.RMSprop(0.001)

model.compile(
              loss=loss,
              optimizer=optimizer,
              metrics=['accuracy']
             )

model.fit(
          train_examples,
          train_labels,
          validation_split = 0.25,
          epochs = 120,
          verbose= 0,
          batch_size = 63
          )
lossResult, accuracyResult = model.evaluate(test_examples, test_labels)
```

Слика 3.4.2. Програмски код за прављење једног модела

На слици 3.4.2. се може видети једнставнији пример имплементације неуралне мреже средње величине у библиотеци *Tensorflow*. Први корак је прављење архитектуре неуралне мреже помоћу *Sequential* модела који се користи када је архитектура линеарна и постоји само један улазни и излазни тензор. Тензор представља мултидимензионални низ елемената неког типа. Након тога се у исти модел убацују скривени слојеви и на крају излазни слој. *Dense* слој представља регуларан густо повезан слој. Последњи *Dense* слој представља излазни слој стога је излаз, у горе наведеном коду, бинаран.

Други корак је компајлирање датог модела уз прослеђивање функције губитка, оптимизационог алгоритма и метрике у односу на коју ћемо гледати успешност алгоритма. У овом случају се користила бинарна функција губитка и *RMSprop* алгоритам, а метрика је прецизност. Трећи корак је тренирање направљеног модела са задатим параметрима. Потребно је да проследимо улазни и излазни тренинг скуп, као и проценат представљен у децималном облику који говори колико ћемо података узети од тренинг скупа за валидациони скуп. *Verbose* параметар нам говори да ли да приказује резултате учења након сваке епохе.

Последњи корак након завршетка учења модела је евалуација добијених параметара на скупу који смо током претпроцесовања података одвојили као тест скуп. Функција *evaluate* враћа коначне вредности функције губитка и прецизности. Ове четири функције представљају генералну структуру већине модела неуралних мрежа прављених у библиотеци *Tesorflow* али свака од њих може да се прошири великим бројем параметара који ће знатно побољшати резултате рада алгоритма.

3.4.3. Графичке картице

Tesorflow је могуће користити и на процесору, али је неколико десетина пута спорије него извршавање на графичкој картици, стога је коришћена *GPU* верзија библиотеке Tensorflow. Коришћене су две различите графичке картице: *Nvidia GeForce 940MX* и *Nvidia*

GeForce MX150. Обе графичке картице се користе за лаптопове и имају 384 језгара и 2GB меморије, при чему је *MX150* око 30% бржа од 940MX.

Тренирање модела на *Nvidia GeForce 940MX* графичкој картици је било проблематично због познате грешке у библиотеци *Tensorflow*. Проблем је настајао када се модел тренирао преко 10 или 20 пута заредом, када прекине рад алгоритма услед пуне меморије. Иако би *Tensorflow* морао да празни меморију након сваког учења јер су међусобно независна то се не дешава и подаци се акумулирају. Једини начин [16], да се меморија празни након сваког тренирања, је да се тренирање и евалуација тренирања обавља у засебним процесима који када се заврше отпуштају сву меморију. Такође нека врста заобилазног начина, што се најчешће и користило у овом раду, је рестартовање кернела сваки пут кад се то деси, што ослободи сву меморију, али је прилично неефикасно.

3.5. Преглед добијених резултата

Као што је објашњено у потпоглављу 3.1. циљ овог рада је да се провери колико добро је могуће научити дубоку неуралну мрежу да на основу улазних података препозна особине говорника. Извршени тестови су препознавање

- јачине позадинског шума
- језика говорника
- дистанце говорника од телефона
- броја говорника
- да ли је било који говорник мушка особа
- да ли је било који говорник женска особа
- броја година говорника
- слова „a“ и речи „књига“, „learning“ и „schmetterling“

У наставку текста ћемо табеларно приказати најбоље резултате сваког теста појединачно као и сумиран приказ свих тестова и категорија. Такође, у наредним табелама ћемо користити скраћенице за улазни тип података где први део може да буде Acc, GyAtt или GRR што представља податке са акцелератора, оријентацију у простору жироскопа и угаону брзину жироскопа, респективно. Други део може да буде C или V, што представља координате или вектор, респективно.

3.5.1. Сумирање категорија свих тестова

Као што се може видети на табели 3.5.1. сваки тест је категорисан у односу на врсту теста. Препознавање позадинског шума има четири, језик три, дистанца три, број особа шест, мушки пол две, женски пол две, године три и реч/слово четири категорије.

Табела 3.5.1. Приказ свих категорија

Позадински шум	Језик	Дистанца	Број особа	Мушко	Женско	Године	Реч / Слово
Нема шума	Српски	0-2м	1	Да	Да	<18	а
Слаб шум	Енглески	3-5м	2	Не	Не	18-30	књига
Средњи шум	Немачки	5+м	3			>30	learning
Јак шум			4				schmetterling
			5				
			6+				

3.5.2. Позадински шум

Као што се може видети у табели 3.5.2. дубока неуронална мрежа са успешности од 88.89% може да предвиди да ли се говорник налази у окружењу где нема шума или постоји слаб, средњи или јак шум. Добијени резултати су веома добри, уједно и најбољи резултати у оквиру овог дипломског рада.

Табела 3.5.2. Приказ резултата предвиђања позадинског шума

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**Acc - C**	**3**	**1024**	**rmsprop**	**0.0011**	**25%**	**88.89%**	**88.89%**
50	Acc - C	3	1024	rmsprop	0.0011	25%	88.89%	88.73%
50	Acc - C	3	1024	rmsprop	0.0011	25%	88.89%	88.57%
100	Acc - C	3	1024	rmsprop	0.0011	25%	88.89%	88.57%
100	Acc - C	3	1024	rmsprop	0.0011	25%	88.89%	88.09%

3.5.3. Језик

У табели 3.5.3. је приказан резултат предвиђања језика говорника, где су коришћене ситуације причања на српском, енглеском и немачком језику. Као што се може приметити, успешност алгоритма је боља од насумичног одабирања за око 13%, што значи да је алгоритам успео да препозна неке особине језика, али није довољно висока прецизност да би се сматрала задовољавајућом.

Табела 3.5.3. Приказ резултата предвиђања језика

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**Acc - V**	**3**	**1024**	**rmsprop**	**0.0011**	**33.33%**	**46.03%**	**39.20%**
50	Acc - V	2	2048	rmsprop	0.001	33.33%	46.03%	38.57%
50	Acc - V	3	1024	rmsprop	0.001	33.33%	46.03%	37.78%
50	Acc - V	3	1024	adam	0.001	33.33%	46.03%	36.67%
50	Acc - V	3	2048	rmsprop	0.001	33.33%	46.03%	36.19%

3.5.4. Дистанца

Дубока неуралнa мрежа са особинама датим у табели 3.5.4. је имала прецизност од 84.13% при препознавању да ли се говорник налази на дистанци од 0-2, 3-5 или преко 5 метара. Обзиром да је прецизност веома висока, овај тест се сматра успешним.

Табела 3.5.4. Приказ резултата предвиђања дистанце

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
100	**GyAtt - C**	**3**	**1024**	**rmsprop**	**0.0011**	**33.33%**	**84.13%**	**83.17%**
50	GyAtt - C	3	1024	rmsprop	0.0011	33.33%	84.13%	83.02%
50	GyAtt - C	3	1024	adam	0.0011	33.33%	84.13%	83.01%
100	GyAtt - C	3	1024	rmsprop	0.0011	33.33%	84.13%	82.86%
100	GyAtt - C	3	1024	rmsprop	0.0011	33.33%	84.13%	82.86%

3.5.5. Број особа

У табели 3.5.5. можемо видети да је алгоритам машинског учења имао најбољу прецизност од 61.90%, што представља 45.23% бољу предикцију него насумичним одабирањем, стога се сматра да је тест веома успешно извршен.

Табела 3.5.5. Приказ резултата предвиђања броја особа

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**GRR - C**	**3**	**1024**	**rmsprop**	**0.0011**	**16.67%**	**61.90%**	**58.89%**
50	GRR - C	3	1024	rmsprop	0.0011	16.67%	61.90%	58.41%
50	GRR - C	3	1024	rmsprop	0.0011	16.67%	61.90%	56.51%
100	GRR - C	3	1024	rmsprop	0.0011	16.67%	60.32%	58.09%
50	GRR - C	3	1024	adamax	0.011	16.67%	60.32%	56.19%

3.5.6. Мушки пол

Препознавање да ли је било који од говорника мушког пола је задовољавајућ са 68.25% успешности, али с обзиром на то да је резултат бинаран, потребно је детаљније се бавити овим тестом да би се постигли бољи резултати. У наставку су приказани резултати у табели 3.5.6.

Табела 3.5.6. Приказ резултата предвиђања мшуког пола

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**Acc - V**	**11**	**1024**	**rmsprop**	**0.001**	**50%**	**68.25%**	**50.21%**
50	Acc - C	3	512	adamax	0.01	50%	66.67%	57.62%
50	GyAtt - V	3	512	sgd	5	50%	63.49%	56.51%
50	GRR - C	11	1024	rmsprop	0.001	50%	63.49%	55.56%
50	GRR - V	11	1024	rmsprop	0.001	50%	58.73%	53.81%

3.5.7. Женски пол

Слично као и у претходном примеру, одређивање да ли постоји женска особа међу говорницима је извршено са прецизношћу од 69.84%, што се може видети у табели 3.5.7. Резултат се сматра задовољавајућим, али даљи рад на овом тесту би требао да побољша резултате

Табела 3.5.7. Приказ резултата предвиђања женског пола

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**GRR - V**	**3**	**512**	**rmsprop**	**0.001**	**50%**	**69.84%**	**61.59%**
50	GRR - V	3	512	rmsprop	0.001	50%	69.84%	61.59%
50	GyAtt - V	3	512	adamax	0.01	50%	69.84%	44.13%
50	GyAtt - V	3	512	adamax	0.01	50%	69.84%	44.13%
50	GRR - V	3	512	rmsprop	0.001	50%	68.25%	61.90%

3.5.8. Године

Као што можемо видети у табели 3.5.8., резултати препознавања да ли говорник или говорници припадају старосној групи испод 18, изнад 18 и испод 30 или изнад 30 година, су прилично успешни.Како је проценат насумичног одабирања 33.3%, постигнут резултат од 76.19% се сматра успешним.

Табела 3.5.8. Приказ резултата предвиђања година говорника

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**GRR - V**	**3**	**1048**	**adam**	**0.0011**	**33.33%**	**76.19%**	**64.13%**
50	Acc - C	3	1024	rmsprop	0.0011	33.33%	74.60%	64.60%
50	GyAtt - V	3	1024	adamax	0.01	33.33%	73.02%	71.90%
100	GyAtt - V	3	1024	adamax	0.01	33.33%	73.02%	71.27%
50	Acc - C	3	2048	adamax	0.011	33.33%	73.02%	71.27%

3.5.9. Реч / Слово

Препознавање речи и слова представља веома велики потенцијал за препознавање речи које особа прича у окружењу теелфона који мирује. Резултати представљени у табели 3.5.9. показују да постоји могућност препознавања слова „а“ и речи „књига“, „learning“ и „schmetterling“ са прецизношћу од 76.19%. Може се упоредити са резултатима из поглавља 2, где су добили прецизност од 65% са само 1 говорником, а горе наведени резултати су имали 4 различита говорника.

Табела 3.5.9. Приказ резултата предвиђања речи/слова

Фрек.	Тип података	Број скрив. слојева	Број скрив. јединица	Оптим. алг.	Корак учења	Прецизност насумичним одабирањем	Најбоља прецизност	Просечна прецизност
50	**GRR - V**	**3**	**2048**	**adam**	**0.0011**	**20%**	**76.19%**	**75.08%**
50	GRR - V	3	1024	adamax	0.011	20%	76.19%	74.92%
100	GRR - V	3	1024	adamax	0.011	20%	76.19%	74.92%
50	GRR - V	10	2048	adam	0.0011	20%	76.19%	74.92%
50	GRR - V	3	1024	adam	0.0011	20%	76.19%	74.76%

3.5.10.Сумирање резултата свих тестова

Табела 3.5.10 представља сумирање резултата свих тестова који смо обављали, заједно са насумичним одабирањем да би се јасно приказао колики напредак је успела да постигне дубока неурална мрежа над улазним подацима.

Најлошији резултати су добијени предвиђањем језика. Прихватљиве и задовољавајуће резултате су показала учења алгоритма да предвиди мушки и женски пол међу говорницима, док успешни резултати су добијени предвиђањем јачине позадинског шума, броја особа, дистанце говорника од телефона, броја година говорника и одређених речи / слова.

Табела 3.5.10. Приказ коначних резултата предвиђања свих тестова

Циљ	Број категорија	Прецизност насумичним одабирањем	Најбоља постигнута прецизност	Просечна постигнута прецизност
Позадински шум	4	25%	**88.89%**	**88.89%**
Језик	3	33.33%	**46.03%**	**39.21%**
Дистанца	3	33.33%	**84.13%**	**83.17%**
Број особа	6	16.67%	**61.90%**	**58.89%**
Мушки пол	2	50%	**68.25%**	**50.21%**
Женски пол	2	50%	**69.84%**	**61.59%**
Године	3	33.333%	**76.19%**	**64.13%**
Реч / Слово	5	20%	**76.19%**	**75.08%**

4.Закључак

Циљ овог дипломског рада је примена алгоритама машинског учења у препознавању неких карактеристика особе која прича у присуству уређаја који поседује жироскоп и акцелератор. За ове потребе, прво је било потребно направити апликацију која ће детектовати и бележити податке са жироскопа и акцелерометра. Затим је требало обрадити те податке и проверити да ли је могуће научити модел машинског учења да може са задовољавајућом прецизношћу да предвиди дату особину говорника. Овај дипломски рад се бавио дубоким неуралним мрежама као алгоритмом машинског учења и његовим резултатима над направљеним улазним подацима. На основу резултата алгоритма се закључује да је могуће препознати одређене карактеристике говорника.

С обзиром на то да су резултати овог рада показали да је могуће сазнати одређене информације о кориснику, а услови овог истраживања су били ограничени и временски и недостатком одговарајућих ресурса, може се претпоставити да велике корпорације, које на располагању имају практично неограничене ресурсе, могу на овај начин сакупити велику количину битних информација о својим корисницима.

Алгоритам дубоких неуралних мрежа који је направљен у овом раду је био успешан у препознавању јачине позадинског шума, дистанце од уређаја, броја година говорника, броја говорника и одређених слова и речи. Препознавање позадинског шума је највеће са 88.89%, затим дистанца, године и реч/слово са 84.13, 76.19 и 76.19%, респективно. То нам омогућава да имамо приступ битним информацијама о особама које причају у непосредној близини уређаја. Са друге стране, препознавање језика говорника, као препознавање да ли постоји мушка или женска особа међу говорницима, немају велику прецизност, иако су резултати бољи од насумичног одабирања.

Резултати добијени у овом истраживању имају велики број примена. На пример, уколико се информација о јачини позадинског шума или о дистанци од уређаја упари са још неким информацијама о говорнику (нпр. геолокацијом), може се добити врло прецизна информација о просторији или окружењу где се говорник налази. Информација о старости говорника може бити значајна у одређивању циљне групе маркетинга неких производа, или се може користити као још једна информација која ствара бољу слику о профилу корисника у базама великих корпорација. Препознавање ограниченог броја речи и слова можда је најзанимљивије сазнање овог истраживачког рада, јер пружа највећи спектар очигледних примена. Као и у претходном примеру, најреалнија употреба препознатих речи (под претпоставком да је вокабулар препознатљивих речи мали) је препознавање назива производа, на основу којег ће се том кориснику приказивати рекламе за тај производ.

Ово истраживање било је ограничено малим бројем примера због немогућности да се креирају све могуће ситуације за прављење тестова. Такође, у фази тренирања алгоритма, рад је био ограничен недовољно јаким перформансама хардверских компонената. Уколико бисмо имали приступ већој количини података или рачунарима са бољим перформансама, могуће је да би резултати овог истраживања били бољи.

Једно од могућих проширења решења би била примена другачијих архитектура дубоких неуралних мрежа. Рекурентне неуралне мреже представљају један тип мрежа који на улаз примају податке који имају међусобну временску зависност и често се користе за препознавање говора. Како је наш скуп података зависан од времена, могуће је да би овај тип

неуралних мрежа давао боље резултате. Препоручљиво је пробати и конволуционе неуралне мреже, јер оне на улазу примају податке са три особине, а како ми имамо *x*, *y* и *z* осу, такође је могућ пораст у резултатима. Да би значај и примена постигнутих резултата били већи, препоручљиво је и проширивати скуп речи које ће алгоритам покушати да научи. Притом, се наравно, мора пазити да прецизност остане у прихватљивим границама. Извесно је да би у овом случају било потребно направити много већи број улазних података.

Литература

[1] "*LSM6DSL Documentation*", https://www.st.com/en/mems-and-sensors/lsm6dsl.html#documentation , приступано дана 25.09.2020.

[2] Yan Michalevsky, Dan Boneh, Gabi Nakibly, „*Gyrophone: Recognizing Speech From Gyroscope Signals*“, https://crypto.stanford.edu/gyrophone/files/gyromic.pdf, *BlackHat Europe 2014*, октобар 2014.

[3] "*Apple iPhone XS review*", https://www.gsmarena.com/apple_iphone_xs-review-1827p3.php, приступано дана 26.09.2020.

[4] „*Regular expression operations*“, https://docs.python.org/3/library/re.html, приступано дана 05.09.2020.

[5] „*Numpy*“, https://numpy.org/, приступано дана 04.09.2020.

[6] „*sklearn.utils.shuffle*“, https://scikit-learn.org/stable/modules/generated/sklearn.utils.shuffle.html , приступано дана 06.09.2020.

[7] Diederik P. Kingma, Jimmy Lei Ba, "*ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*", https://arxiv.org/pdf/1412.6980.pdf, приступано дана 14.09.2020.

[8] „*Anaconda*“, https://www.anaconda.com/, приступано дана 08.09.2020.

[9] „*Matplotlib*“, https://matplotlib.org/, приступано дана 18.09.2020.

[10] "*Spyder*", https://www.spyder-ide.org//, приступано дана 08.09.2020.

[11] "*Keras*", https://keras.io/, приступано дана 02.09.2020.

[12] "*CUDA Toolkit*", https://developer.nvidia.com/cuda-toolkit, приступано дана 01.09.2020.

[13] "*cuDNN*", https://developer.nvidia.com/cudnn, приступано дана 01.09.2020.

[14] "*GeForce 940MX*", https://www.techpowerup.com/gpu-specs/geforce-940mx.c2797, приступано дана 28.09.2020.

[15] "*GeForce MX150*", https://www.techpowerup.com/gpu-specs/geforce-mx150.c2959, приступано дана 28.09.2020.

[16] "How can I clear GPU memory in tensorflow 2?", https://github.com/tensorflow/tensorflow/issues/36465, приступано дана 19.09.2020.

Списак скраћеница

SVM	*Support Vector Machine*
GMM	*Gaussian Mixture Model*
DTW	*Dynamic Time Warping*
SGD	*Stochastic gradient descent*
GPU	*Graphics processing unit*
Acc	*Acceleration*
GyAtt	*GyroscopeAttitude*
GRR	*GyroscopeRotationRate*
C	*Coordinates*
V	*Vector*

Списак слика

Списак табела

A. Изворни код модела дубоке неуралне мреже

У овом прилогу биће дато целокупан код који представља модел дубоке неуралне мреже који се користио у овом дипломском раду.

```
def plot_learning_rate(ep,alfa):
    step = np.linspace(0,20*ep)
    lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
          alfa,
          decay_steps=200,
          decay_rate=1,
          staircase=False)
    lr = lr_schedule(step)
    plt.figure(figsize = (8,6))
    plt.plot(step/20, lr)
    plt.ylim([0,max(plt.ylim())])
    plt.xlabel('Epoch')
    _ = plt.ylabel('Learning Rate')

def train(totalLoops, activF, i_size, numLayers, numUnits, outputUnits,
alfa, optFlag, optim, pat, train_e, train_l, test_e, test_l, epoch, verb,
mini_batch):
    l = 0
    a = 0
    smallest_l = 5
    biggest_a = 0
    for i in range(totalLoops):
        model = tf.keras.Sequential()
        model.add(layers.Dense(numUnits, activation=activF,input_shape =
            (i_size,), kernel_regularizer=regularizers.l2(0.001)))
        model.add(layers.Dropout(0.5))
        for js in range(numLayers) :
            model.add(layers.Dense(numUnits, activation=activF,
                  kernel_regularizer=regularizers.l2(0.001)))
            model.add(layers.Dropout(0.5))

        model.add(layers.Dense(outputUnits, activation='softmax'))

        lr_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(
          alfa,
          decay_steps=200,
          decay_rate=1,
          staircase=False)

        callbacks = [
                    tf.keras.callbacks.EarlyStopping(
                        monitor='val_loss', patience = pat,
                        restore_best_weights = True
                    )
        ]
```

```
if(optFlag):
      if(optim == 'adam'):
          optimizer1=tf.keras.optimizers.Adam(learning_rate =
                                                    lr_schedule)
      if(optim == 'adamax'):
          optimizer1=tf.keras.optimizers.Adamax(learning_rate =
                                                    lr_schedule)
      if(optim == 'rmsprop'):
          optimizer1=tf.keras.optimizers.RMSprop(learning_rate =
                                                    lr_schedule)
      if(optim == 'sgd'):
          optimizer1=tf.keras.optimizers.SGD(learning_rate =
                                                    lr_schedule)



  else:
      if(optim == 'adam'):
          optimizer1=tf.keras.optimizers.Adam(learning_rate = alfa)
      if(optim == 'adamax'):
          optimizer1=tf.keras.optimizers.Adamax(learning_rate = alfa)
      if(optim == 'rmsprop'):
          optimizer1=tf.keras.optimizers.RMSprop(learning_rate =
                                                        alfa)
      if(optim == 'sgd'):
          optimizer1=tf.keras.optimizers.SGD(learning_rate = alfa,
                                                    momentum=0.9)

  model.compile(optimizer=optimizer1, loss  =
tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                metrics =
[tf.keras.metrics.SparseCategoricalAccuracy()])

  history = model.fit(
      train_e,
      train_l,
      validation_split = 0.25,
      epochs =epoch,
      callbacks=callbacks,
      verbose=verb,
      shuffle=False,
      batch_size = mini_batch
      )

  loss1, accuracy = model.evaluate(test_e, test_l)

  history_dict = history.history
  history_dict.keys()
  loss = history_dict['loss']
  val_loss = history_dict['val_loss']

  epochs = range(1, len(loss) + 1)
  plt.plot(epochs, loss, 'bo', label='Training loss')
  plt.plot(epochs, val_loss, 'b', label='Validation loss')
  plt.title('Training and validation loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.savefig('plot10.png', dpi=300)
  plt.show()
```

```
        print("Epochs: ", epochs.__len__())
        print("Loss: ", loss1)
        print("Accuracy: ", accuracy)
        l = l+loss1
        a = a+accuracy
        if(accuracy>biggest_a):biggest_a = accuracy
        if(loss1<smallest_l):smallest_l = loss1
        print("Iteration: ", i)
    a = a/totalLoops
    l = l/totalLoops
    print("Loss: ", l)
    print("Average accuracy: ", a)
    print("Best accuracy: ", biggest_a)
    plot_learning_rate(epoch,alfa)
    s = str(freq) + '\t' + input_type + '\t' + ('Coords' if
      train_e.shape[1]>train_vector_examples.shape[1] else 'Vector') + '\t'
      + "60|20|20" + '\t' + "Yes" + '\t' + str(rnd) + '\t' + "Yes" + '\t' +
      str(numLayers) + '\t' + str(numUnits) + '\t' + "0.001" + '\t' + "0.5"
      + '\t' + activF + '\t' + str(outputUnits) + '\t' + "Sigmoid" + '\t' +
      optim + '\t' + str(alfa) + '\t' + ('200\t1' if optFlag else '/\t/') +
      '\t' + "BinaryCrossentropy\tAccuracy" + '\t' + str(epoch) + '\t' +
      str(pat) + '\t' + str(smallest_l) + '\t' + str(biggest_a*100) + '\t'
      + str(a*100) + '\t\n'

    return s
```