Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Távközlési és Médiainformatikai Tanszék

Czeba Bálint

TEVÉKENYSÉG MODELLEZÉS SZENZORADATOK ALAPJÁN MÉLY NEURÁLIS HÁLÓZATOKKAL ANDROIDOS OKOSTELEFONON

KONZULENSEK

Dr. Gyires-Tóth Bálint

Dr. Németh Géza

BUDAPEST, 2017

Tartalomjegyzék

Összefoglaló

Napjainkban a hordozható eszközök - mint például az okostelefonok, okosórák - a mindennapi életünk fontos részévé váltak. A jelenleg elterjedt okoseszközök használata közben jelentős mennyiségű – a felhasználóra jellemző – szenzoros (pl. gyorsulásmérő, orientációs szenzor, mágneses mezőt érzékelő szenzor, fényérzékelő- és közelségérzékelő szenzor, stb.) adatot lehet kinyerni, amikből következtetni lehet a felhasználó aktuális tevékenységeire és a környezetére. A felismert viselkedési mintázatok alapján intuitívabbá tehető az ember-gép kommunikáció azáltal, hogy például az eszköz automatikusan végrehajtja az aktuális kontextusban legvalószínűbb lépéseket, vagy azokat felajánlja a felhasználónak a manuális vezérlés helyett.

Az okoseszközökben a szenzorokat jellemzően 50 Hz-es, sok esetben akár magasabb frekvenciával is lehet mintavételezni, ami nehéz feladat elé állítja a témával foglalkozó kutatókat. Az ilyen jellegű, nagy mennyiségű adat a napjainkban aktívan kutatott mély neurális hálózatokkal feltevésem szerint feldolgozható és modellezhető, akár egy lépésben.

A mély neurális hálózatok az alkalmazott mesterséges intelligencia területén jelenleg talán a legaktívabban kutatott téma. Az elméleti megfontolásokat technológia fejlődése is segíti, például a GPU-k (Graphical Processing Unit, grafikai processzor) fejlődésével és ezen a területen történő felhasználásával jelentős mértékű gyorsulást értek el a neurális hálózatokon alapuló modellek tanítása során. Ezekkel a fejlesztésekkel olyan problémák is megvizsgálhatók, amik a korábbiak során az elégtelennek bizonyuló számítási kapacitás miatt túl sok időt vettek volna igénybe. A mély neurális hálózatok a rejtett rétegeinek számának növelésével az adatok egyre magasabb szintű absztrakcióinak kinyerésére képesek, melyek segítségével a korábbi gépi tanuló eljárásoknál pontosabb modellek építhetők.

Dolgozatomban áttekintem a már meglévő rendszereket és egyéb megoldásokat. A korábbi rendszerekhez képest új megközelítést alkalmazok az adatok feldolgozása során, a mély neurális hálózatok használatával. Elkészítettem egy mobilalkalmazást, amivel a szenzoros adatok rögzíthetők, valamint a rögzített adatokból történő modellek építéséhez szerveroldali tanítóalkalmazást is implementálok. A mobilalkalmazással több ember segítségével adatrögzítéseket végeztem és lehetővé tettem a betanított modellek valós életbeli tesztelését is. Eredményeimet részletesen dokumentáltam és objektív módszerekkel értékeltem ki.

Abstract

Nowadays portable devices - like smartphones and smartwatches - have become an important part of our daily lives. The currently available smart devices can record a great number of sensor data (for example: accelerometer, orientation sensor, magnetic field sensor, light and proximity sensor), which can be used to estimate users' activities. The recognized behavioral patterns can be used to improve human-computer interaction by automatically performing the predicted steps in the current context of the user, instead of manual execution.

In these smart devices the sensors can be sampled usually at 50 Hz or even at higher frequencies. This means a difficult task for researchers. The large amount of data could be processed using data-mining techniques or by taking advantage of the actively researched Deep Neural Networks (DNN).

Deep Neural Networks are the most actively researched topic in the Artificial Intelligence area. With the recent advancements in GPU (Graphical Processing Unit) accelerated computing and the deep learning algorithms, we are able to solve problems that were previously nearly impossible due to inefficient computational capacity. DNNs are capable of extracting higher abstraction levels of the data. These abstraction levels can be used to build more accurate models compared to the previously available machine learning solutions.

In this paper I review the scientific literature connected to my topic of research. I present a different approach for processing the sensor data using Deep Neural Networks. I have implemented a smartphone application that can be used to record sensor data during daily activities. To build models I provided a server-side training application. Using the smartphone application I recorded several person's activity data and the goal is that the smartphone application will be able to predict the recorded data in real time. I introduce the results in detail and evaluate them.

1 Bevezetés

Napjainkban a hordozható eszközök - mint például az okostelefonok, okosórák - a mindennapi életünk fontos részévé váltak. Már nem csak a felhasználók egymás közti kommunikációját teszik könnyebbé, hanem a munka, és a mindennapi tevékenységek során is sokszor használjuk őket. Az okoseszközök használata közben jelentős mennyiségű – a felhasználóra jellemző – szenzoros (pl. gyorsulásmérő, orientációs szenzor, mágneses mezőt érzékelő szenzor, fényérzékelő, közelségérzékelő, stb.) adatot lehet kinyerni, amikből következtetni lehet a felhasználó aktuális tevékenységeire és a környezetére [1], [2]. A felismert viselkedési mintázatok alapján intuitívabbá tehető az ember-gép kommunikáció azáltal, hogy az eszköz például automatikusan végrehajtja az aktuális kontextusban szükséges lépéseket, vagy azokat felajánlja a felhasználónak, ahelyett, hogy ezeket a lépéseket kézzel kelljen megtenni. Ilyen felhasználási lehetőség lehet például egy sporttevékenységeket rögzítő alkalmazás (például: Endomondo, Runtastic, Google Fit, stb.) számára az, hogy ha a felhasználó az adott tevékenységet végzi (futás, biciklizés, sétálás stb.) akkor automatikusan indítsa el a rögzítést. Az utóbbi években a fentebb említett felhasználási lehetőségeken túl egyre inkább előtérbe kerülő kutatási téma az okoseszközök adatainak felhasználása orvosi célokra is [3], [4].

Az okostelefonos környezetben elérhető szenzoros adatokat már több korábbi irodalomban is megvizsgálták a tevékenységfelismerés szemszögéből, viszont ott jellemzően nem mély neurális hálózatokon alapuló gépi tanuló módszereket alkalmaztak. A mély neurális hálózatok jelenleg is aktív kutatási téma számos alkalmazási területen [5] [6] [7] [8] [9]. A kutatásokat technológia fejlődése is segíti, például a GPU-k (Graphical Processing Unit, grafikai processzor) fejlődésével és ezen a területen történő felhasználásával jelentős mértékű gyorsulást értek el a neurális hálózatokon alapuló modellek tanítása során. Ezekkel a fejlesztésekkel olyan problémák is vizsgálhatók, melyek korábban az elégtelennek bizonyuló számítási kapacitás miatt túl sok időt vettek volna igénybe. A mély neurális hálózatok a rejtett rétegek számának növelésével az adatok különböző szintű absztrakcióinak kinyerésére, megtanulására képesek, melyekkel pontosabb modellek építhetők a korábbi gépi tanuló eljárásoknál.

Dolgozatom célja egy olyan rendszer létrehozása, ami képes felismerni az okostelefon felhasználójának aktuális tevékenységét a telefonon elérhető szenzoros és

egyéb adatforrások (például WiFi kapcsolat információi, lokációs adatok, stb.) alapján. A munka fő lépéseinek a következőket fogalmazom meg:

1. Androidos alkalmazás tervezése és implementálása okostelefonos szenzoradatok és egyéb adatforrások gyűjtésére
2. Az adatrögzítő alkalmazással különböző tevékenységek végrehajtása közben adatok rögzítése legalább hat ember részvételével
3. A rögzített adatok felhasználásával mély neurális hálózat alapú modellek kialakítása a felhasználó aktuális tevékenységének felismerésére és a létrehozott modellek objektív módszerekkel való tesztelése
4. Kliens-szerver architektúra kialakítása az adatok rögzítéséhez és az azok alapján készült modellek online futtatásához
5. A betanított modellek okostelefonon történő offline futtatásának lehetővé tétele
6. Demonstrációs mintaalkalmazás létrehozása a modellek valós életbeli teszteléséhez

Dolgozatomban áttekintem a már meglévő megoldásokat és a témámhoz kapcsolódó szakirodalmat. A korábbi rendszerekhez képest új megközelítést dolgozok ki az adatok feldolgozása során, mély neurális hálózatok használatával. Okostelefon alkalmazást tervezek, amivel nagy felbontással rögzíthetők a szenzoros adatok. A rögzített adatokból történő modellek építéséhez szerveroldali mély tanuláson alapuló rendszert terveztem és ismertetem az implementálás lépéseit. TCP/IP alapú kliens-szerver architektúrát dolgoztam ki a szenzoradatok beküldésére és a betanított modellek eredményének visszaküldésére. A kommunikáció kidolgozásánál figyelmet fordítottam a biztonságos adatátvitelre. Több ember segítségével adatokat gyűjtöttem a mobilalkalmazás segítségével, amiket gépi tanulási modellek tanítására használtam fel. A feladatot megnehezíti, hogy a különböző készüléktípusokba a gyártók más-más érzékelőfajtákat, azon belül is különböző fizikai jellemzőkkel bíró hardvereket építenek be. Az azonos típusú – például gyorsulásmérő –, de különböző specifikációjú szenzorok az esetleges kalibrációs hibák vagy mérési pontatlanságok miatt megegyező körülmények mellett is más értékeket mérhetnek. Ezen felül emberenként is lényegesen különböznek a vizsgált tevékenységek, ezért szükséges a nagy mennyiségű jó minőségű tanítóadat a modellek építésekor. Az okostelefon által mért adatok kliens-szerver alapú kiértékelésén felül megvizsgáltam a telefonon történő offline modellfuttatások lehetőségeit is.

Szerver oldalon a dolgozat megoldása során vizsgált osztályozási feladatra különböző mély neurális hálózatarchitektúrákat alkalmazok: a dolgozatban áttekintem a legegyszerűbb felépítésű előrecsatolt hálózatokat, a tulajdonságvektorok vizsgálatára a konvolúciós hálózatokat alkalmaztam, az időbeli összefüggéseket pedig visszacsatolt (rekurrens) hálózatokkal modelleztem GPU alapon. Eredményeimet részletesen dokumentáltam és objektív módszerekkel értékeltem a tanítóadatbázisról leválasztott tesztadatbázison. A mobilalkalmazásban is lehetővé tettem a betanított modellek valós életbeli használatát. A megoldást teszteltem szoftveres környezetben illetve valós felhasználókkal is. A tevékenységfelismerés eredményességét pontosság (precision), felidézés (recall) és F1 metrikákkal mértem.

2 Irodalomkutatás

Az irodalomkutatás során áttekintettem, hogy a munkám témájával kapcsolatban milyen korábbi megoldások érhetők el, és ezekben milyen módszerekkel, milyen eredményeket értek el. Az irodalomkutatás részeként áttekintettem a munkám során használt legfontosabb gépi tanulási eljárások elméleti hátterét is.

2.1 Korábbi megoldások

Elsőként egy 12 évvel ezelőtt közzétett kutatás eredményeit ismertetem, amit az Amerikai Egyesült Államok New Jersey-ben található Rutgers Egyetemén végeztek el [1]. A kutatás során megvizsgálták, hogy milyen lehetőségek rejlenek a gyorsulásmérők szenzoros értékei alapján történő tevékenységfelismerést megvalósító rendszerekben. Az adatok rögzítéséhez egy háromdimenziós gyorsulásmérőt használtak. A háromdimenziós gyorsulásmérő egy olyan szenzor, ami az x, y és z tengely szerinti gyorsulásra ad becsléseket. A cikkben a következő kérdésekre próbáltak meg válaszokat találni: Melyek a legjobb osztályozó algoritmusok a tevékenységek felismerésére? Melyek a mozgás azon jellemzői, amik fontosabbak / kevésbé fontosak? Mely tevékenységeket nehezebb felismerni a többinél?

A cikk szerzői az adatok gyűjtésekor a gyorsulásmérő szenzort az adatrögzítésben résztvevő emberek medencéjének tájékán rögzítették, miközben az emberek különböző tevékenységeket végeztek. Az adatokat két ember segítségével rögzítették. A gyorsulásmérő a mért adatokat egy Bluetooth-al összekapcsolt HP iPAQ eszközre küldte, ami eltárolta a mért adatokat, a későbbi adatvizsgálatok céljából. Az adatrögzítés 50 Hz-es mintavételi frekvenciával történt. A mérések során nyolc különböző tevékenységről végeztek adatrögzítéseket: állás, séta, futás, lépcsőzés felfelé, lépcsőzés lefelé, felülések, porszívózás és fogmosás.

Az adatok címkézése fél-automata módon történt. Az adatrögzítések során stopperórával mérték az egyes tevékenységek rögzítésének időtartamát, majd később rendelték hozzá a megfelelő címkét az adatpontokhoz.

A rögzített adatokat 256 minta méretű ablakok segítségével osztották fel úgy, hogy a szomszédos ablakokban 128 mintányi átfedés volt. Azért ilyen módon ablakoztak, mert az már az előzetes munkák során is sikeresnek bizonyult. Ilyen ablakmérettel minden

ablak 5,12 másodpercnyi időtartamnak felel meg. Az adatokat négyféle módszerrel dolgozták fel: átlag, normál eloszlás, energia, korreláció.

A feldolgozott adatokkal egyrészt a Weka programcsomagban elérhető következő tanulóalgoritmusokat tanították: Döntési táblák, Döntési fák, KNN (K-nearest neighbor – K legközelebbi szomszéd), SVM (Support Vector Machine – Szupport vektor gép), Naiv Bayes. Ezeken felül pedig több meta-szintű osztályozó módszert is használtak: boosting, bagging, többségi szavazás, stacking. Az itt említett algoritmusok bővebb ismertetése megtalálható a hivatkozott irodalomban. A tanítások hatékonyságát a keresztvalidáció módszerével értékelték.

A kutatómunkában a legjobban szereplő osztályozási módszer a többségi szavazás volt: az algoritmus több alap szintű osztályozó eredményei alapján hozza meg a végső döntést. Az adatfeldolgozási módszerek közül a jel energiájának vizsgálatát találták a legkevésbé hatékonynak. A vizsgált tevékenységek között különösen nehéz volt a fogmosást felismerni, valamint rosszul ment a felfelé és a lefelé történő lépcsőzés megkülönböztetése. A kutatók úgy vélik, hogy bizonyos tevékenységeket (például: állás, séta, futás, lépcsőzés felfelé, lépcsőzés lefelé, felülések, porszívózás) nagy pontossággal lehet felismerni a medencén rögzített szenzorokkal, míg a csak kéz és száj mozgásával járó (például: fogmosás) tevékenységeket nehéz felismerni.

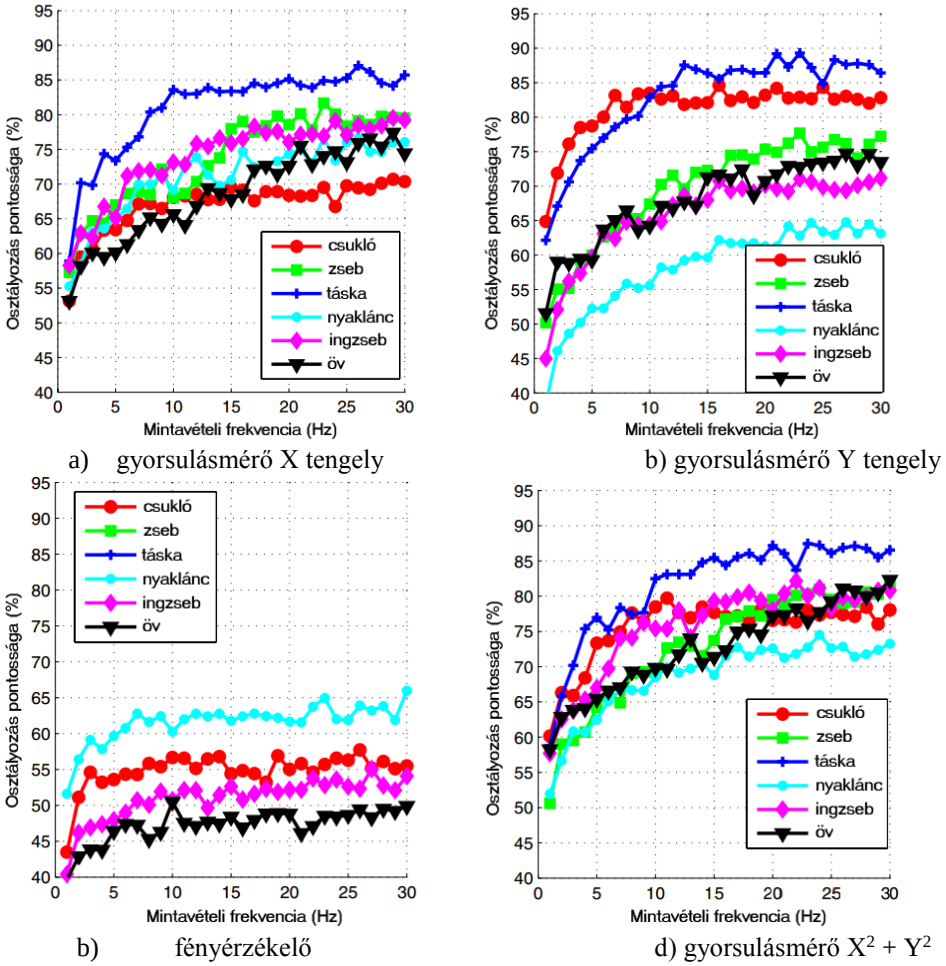
A következő kutatást három intézmény végezte: a pittsburgh-i Carnegie Mellon Egyetem, a Müncheni Műszaki Egyetem, és az Intel. A kutatásban megvizsgálták az osztályozó algoritmusokkal végzett tevékenység-felismerés hatékonyságát úgy, hogy több szenzort rögzítettek a kutatásban résztvevő emberekre [10]. Az érzékelőket a résztvevők bal csuklóján, övén, ingzsebében, jobb oldali nadrágzsebében, hátizsákján és nyakláncán helyezték el.

A méréseket végző eszközben egy kétirányú gyorsulásmérő, egy fényérzékelő, egy hőmérő és egy mikrofon található. Adatrögzítésre a gyorsulásmérő és a fényérzékelő adatait választották, a rögzített értékeket a mérésekre használt eszköz tárolta. A kutatás során hat tevékenység adatairól gyűjtöttek adatokat: ülés, állás, sétálás, lépcsőzés felfelé, lépcsőzés lefelé és futás. Az adatok rögzítését hat ember végezte, mindegyikük 45-50 percen keresztül. Az adatrögzítés 50 Hz-es mintavételi frekvenciával történt.

Az adatok feldolgozását és elemzését a Matlab és a Weka programcsomagokkal végezték. Az adatokat csak időtartományban vizsgálták, hogy elkerüljék a

frekvenciatartományba történő transzformálás számítási költségeit. Lineáris Diszkrimináns Analízis használatával megvizsgálták a rögzített adatokat és megállapították, hogy az ülés, állás és futás kategóriák jól elkülönülő klasztereket alkotnak, míg a sétálás, és a fel/le lépcsőzés rögzített jellemzői közel vannak egymáshoz. Az adatfeldolgozáshoz a jellemzők kiválasztását a Weka programcsomag korreláció alapú jellemző kiválasztás (Correlation based Feature Selection) funkcióját használták. Osztályozó algoritmusként a döntési fa osztályozót választották amiatt, hogy megfelelő egyensúlyt biztosított a felismerési pontosság és az alacsony számítási kapacitás szükséglete terén.

A kutatás során megvizsgálták, hogy mennyire befolyásolja a mintavételi frekvencia az osztályozási feladat pontosságát úgy, hogy az 50 Hz-en mintavételezett adatokat többféleképpen alulmintavételezték. A 2.1. ábra mutatja be a mintavételi frekvencia, különböző adatforrások és a felismerési pontosság közti összefüggéseket.



a) gyorsulásmérő X tengely

b) gyorsulásmérő Y tengely

b) fényérzékelő

d) gyorsulásmérő $X^2 + Y^2$

2.1. ábra -Felismerési pontosság különböző jellemző-vektorok és mintavételi frekvenciák használatával (forrás: [10] alapján)

A szerzők a tesztek eredményeinek kiértékelése alapján a 20 Hz-es mintavételi frekvenciát választották a tanítások során.

A kutatás során megvalósítottak egy olyan döntési fa alapú osztályozót, ami a csuklón rögzített szenzor adatai alapján volt tanítva, és magán a szenzor egységen fut. Ezzel az összeállítással tesztelték a megvalósított rendszert.

Egy, a texasi A&M egyetemen elvégzett kutatás során okostelefonnal történő adatok rögzítése alapján próbálták osztályozni az emberek tevékenységeit [11]. A cikkben olcsó, boltokban is kapható eszközöket használtak, és az okostelefonokba épített háromdimenziós gyorsulásmérő szenzor adatait rögzítették 50 Hz mintavételi frekvenciával. A szerzők öt féle tevékenység gyorsulásadatait rögzítették: sétálás, bicegés, kocogás, felfelé lépcsőzés, lefelé lépcsőzés. Az adatrögzítés során három ember segítségével gyűjtöttek méréseket. Az adatok rögzítése során az eddigiektől eltérően nem volt kikötve, hogy hol és milyen irányban kell elhelyezni az adatrögzítést végző eszközt, azt az adatrögzítő tetszőleges helyre tehette, például a nadrágzsebébe, kabátzsebbe, stb. A rögzített adatokat 256 mintából álló darabokra bontották, és minden ablakból 31 különböző jellemzővektort nyertek ki, ezek közt idő és frekvenciatartománybeli elemzések is szerepeltek. Az így feldolgozott adatok használatával négyféle osztályozót tanítottak: kvadratikus osztályozó, KNN, SVM és neurális hálózat. A kutatás során passzív és aktív tanulási módszerekkel is dolgoztak.

A passzív módszer során a címkézett adatokkal tanították a tanulóalgoritmusokat, és e módszer használatával választották ki a 31 jellemzővektor közül azokat, amelyeket később az aktív tanítások során is felhasználtak. A később használni kívánt jellemzők kiválasztásához a szekvenciális jellemző szelektálás (SFS) módszerét használták.

Az aktív tanulási módszerek használatával nem címkézett adatokkal tanították az algoritmusokat. Az aktív tanulás mögötti elmélet az, hogy a gépi tanulási technika jobb felismerési pontosságot tud elérni úgy, ha kevesebb felcímkézett adat áll a rendelkezésére, és ő választhatja ki, hogy mely adatokból tanul. A tanulási folyamatban szükséges egy külső „segítő” jelenléte, aki a tanulóalgoritmus kérésére felcímkézi a kért adatpontokat.

A cikk eredményei alapján a legjobb osztályozási pontosság 84,4% volt, és ezt az SVM osztályozóval érték el.

A következőnek áttekintett folyóiratcikket 2014-ben publikálták a fordhami egyetem kutatói (Department of Computer & Information Science). A kutatás célja egy olyan okostelefonon alapuló rendszer kifejlesztése volt, ami képes segíteni abban az embereknek, hogy megfelelő mennyiségű mozgást végezzenek a mindennapjaik során [2].

A szerzők megjegyzik, hogy a piacon már van több tevékenységkövetést is megvalósító termék (Fitbit, Nike+ Fuelband), viszont ezekhez az okostelefonon felül további hardverek megvásárlása is szükséges. A kutatók egy „Actitracker" nevű szolgáltatást fejlesztettek ki, melynek a napi tevékenységek azonosításához csak az okostelefonokba épített szenzorok szükségesek valamint a szoftvert is ingyenesen hozzáférhetővé teszik, így sokkal szélesebb közönség számára lesz elérhető, mint az előbb említett megoldások.

Az elkészített rendszer kliens-szerver architektúrájú, az okostelefonon a kliens alkalmazás végzi a gyorsulásmérő adatainak rögzítését (amikor a telefon nincs töltőn és a kijelző ki van kapcsolva), és beküldi a szervernek a rögzített adatokat. A kutatók rávilágítanak, hogy ha az adatok előfeldolgozását is az okostelefon végezné, és már az előfeldolgozott adatokat továbbítaná a szervernek, akkor a küldendő adat mennyisége jelentősen csökkenthető lenne (a cikk szerint a teljes adatméret 2,9%-ára), viszont a jelenlegi módszer előnye a kutatás időtartama alatt, hogy a szerver így el tudja tárolni a nyers szenzoradatokat.

A szerver az adatok fogadása után elvégzi a szükségesnek ítélt adattisztítási lépéseket és feldolgozza az adatokat 10 másodperces ablakokban (nincs átfedés). A feldolgozott adatokat ezután osztályozzák, hogy azokat a következő öt kategória valamelyikébe be tudják sorolni: sétálás, kocogás, lépcsőzés, állás, ülés/fekvés. Fontos szempont az okostelefon „rögzítése" az embereken, itt azt feltételezték, hogy a telefon a felhasználók első nadrágzsebében van. Használat közben megmérték azt is, hogy mennyivel csökkent az akkumulátor üzemideje, kb. 20-30% csökkenést dokumentáltak.

A kutatás során az adatok osztályozására a Random Forest osztályozóalgoritmust használták. A tanításokhoz a szerzők korábbi kutatásaik során gyűjtött adatokat használták, majd a felhasználók adatai és visszajelzései alapján javítottak a modellen. Az osztályozás eredményeit a felhasználók egy webes felületen tudták ellenőrizni és azokat értékelni. Az eredmények értékelését a felhasználók visszajelzései alapján és objektív alapon is értékelték. A felhasználóknak 1-től 5-ig terjedő skálán kellett értékelni a

szolgáltatást, különböző kérdések alapján. Az objektív értékeléshez a felhasználók egy része címkézett adatrögzítést is végzett, ezeken az adatokon a modell átlagosan 89%-os pontossággal operált.

A cikkben említettek szerint a szerzők valóban közzé tették az elkészített alkalmazást a Google Play Áruházban. Az alkalmazás regisztrációhoz kötött, a regisztrációs folyamat viszont saját tapasztalataim és az alkalmazásboltbeli visszajelzések alapján a többi érdeklődő szerint sem működik, így az alkalmazást nem lehet kipróbálni.

Egy olasz és spanyol egyetemek által közösen végzett kutatásban [12] egy olyan rendszert vázolnak fel, melyben okostelefonokon futtatható, alacsony hardverigényű gépi tanulási algoritmusok használatával történik a felhasználó aktuális tevékenységének a felismerése.

Az adatrögzítéseket egy Samsung Galaxy S2 okostelefonon végezték a kutatók, 30 résztvevő segítségével, a telefont a tesztalanyok derekára rögzítették. Az adatokat utólag címkézték úgy, hogy a rögzítés alatt videófelvételeket készítettek. A rögzített tevékenységek: állás, sétálás, fekvés, felfelé és lefelé lépcsőzés. A szenzorok közül a gyorsulásmérővel mért értékeket rögzítették 50Hz-es mintavételi frekvenciával. Az adatokból idő és frekvenciatartománybeli jellemzőket számítanak, amikből egy 17 jellemzőből álló tanítóvektort állítanak össze úgy, hogy egy időablak 2,56 másodpercnyi adatból áll, az ablakok 50%-os átfedésben vannak egymással. A tanításhoz és a tanítás ellenőrzéséhez 70% - 30 % arányban felosztották a tanítóadatbázist.

A szerzők ezután egy többosztályos HF-SVM (Hardware-Friendly SVM) modellt definiálnak, ezt a modellt tanították és vizsgálták a rögzített adatokkal, a modell különböző paraméterezései mellett. A modell eredményeit egy ún. konfúziós mátrixban (tévesztési mátrix) szemléltetik, összehasonlítva más SVM alapú megoldásokkal is (2.1. táblázat).

2.1. táblázat - Konfúziós mátrix, Multiclass SVM és Multiclass HF-SVM modellek felismerési pontosságának összehasonlítása (forrás: [12] alapján)

	MC-SVM							MC-HF-SVM k=8 bits						
Tevékenység	Séta	Lépcsőzés fel	Lépcsőzés le	Állás	Ülés	Fekvés	Fedés (%)	Séta	Lépcsőzés fel	Lépcsőzés le	Állás	Ülés	Fekvés	Fedés (%)
Séta	**109**	0	5	0	0	0	95,6	**109**	2	3	0	0	0	95,6
Lépcsőzés fel	1	**95**	40	0	0	0	69,8	1	**98**	37	0	0	0	72,1
Lépcsőzés le	15	9	**119**	0	0	0	83,2	15	14	**114**	0	0	0	79,7
Állás	0	5	0	**132**	5	0	93,0	0	5	0	**131**	6	0	92,2
Ülés	0	0	4	4	**108**	0	96,4	0	1	0	3	**108**	0	96,4
Fekvés	0	0	0	0	0	**142**	100	0	0	0	0	0	**142**	100
Pontosság (%)	87,2	87,2	72,6	97,1	95,5	100	**89,3**	87,2	81,7	74,0	97,8	94,7	100	**89,0**

A konfúziós mátrix sorai a valós, oszlopai pedig a becsült osztályt jelölik. A mátrix főátlójában a félkövérrel szedett értékek a helyesen osztályozott minták számát tartalmazzák. A konfúziós mátrixszal szemléletesen lehet ábrázolni az egyes kategóriákon visszamérhető osztályozás pontosságát ezért az eredmények ismertetésénél ebben a dolgozatban is használni fogom.

A cikk összefoglalásában pozitívan értékelik a munka eredményeit, melyek megerősítik, hogy a definiált eszközbarát SVM modell is megközelíti a hagyományos SVM modell hatékonyságát, miközben sikerült csökkenteni a modell számítási kapacitásigényét.

Egy 2017-ben a dél-koreai Kookmin egyetemen végzett kutatásban megvizsgálták a tevékenységfelismerés problémáját un. konvolúciós neurális hálózatok (bővebben: 2.2.3 - Konvolúciós neurális hálózatok) használatával [13].

Az adatok rögzítését öt végzős diák segítségével végezték, öt darab megegyező hardveres felépítésű Nexus 6P okostelefonnal. Az adatok rögzítéséhez saját alkalmazást írtak, amely az adatok rögzítéskori címkézését is lehetővé tette. Az adatrögzítő alkalmazás az okostelefon gyorsulásmérőjének értékeit (x, y és z tengely) rögzítette 1 Hz mintavételi frekvenciával (1 minta / másodperc). Három tevékenység rögzítését kérték a diákoktól, ezek a sétálás, futás és mozdulatlanság voltak. Összesen kb. 2,7 óra adatot rögzítettek.

A rögzített adatokat transzformációknak vetették alá, hogy csökkentsék az eszközök más-más elhelyezéséből adódó különbségeket. A három tengelyen mért adatokból egyetlen vektort készítettek a következő képlet szerint:

$$||a|| = \sqrt{x^2 + y^2 + z^2}.$$

A kutatásban a neurális hálózatot (az itt említett neurális hálózatokkal kapcsolatos fogalmakat a dolgozat későbbi részein bővebben kifejtem) a következőképp építették fel: a bemeneti réteg 10 vagy 20 időpillanatot kap meg, ami 10 illetve 20 másodpercet reprezentál. A bemeneti réteg után egyetlen konvolúciós réteg áll, ami 3, 4 és 5 méretű ablakokat vesz figyelembe, 128 db konvolúciós szűrőréteggel. Ezután következik egy max-pooling réteg, egy dropout majd a kimenet leképzését végző rétegek.

A konvolúciós hálózat eredményeinek vizsgálatához építettek ugyanerre a feladatra egy egyszerű Random Forest alapú modellt is. A konvolúciós neurális hálózat minden esetben felülmúlta a Random Forest modellt, a legjobb 92,71% pontosságú eredményt a három tevékenység vizsgálata során akkor érték el, amikor a hosszabb, 20 másodperces időablakokat vizsgálták.

A feldolgozott korábbi megoldásokban a modellek építése előtt a nyers adatok előfeldolgozási lépéseken (transzformációk) mentek keresztül. Több megoldásban az okostelefonon felül további eszközöket is kell használni, ami a felhasználók számára többletköltséget és kellemetlenséget is jelenthet. Munkám során a tanulóalgoritmusokat nyers adatokkal tanítottam, így a lényegkiemelést is a betanított modellek végzik, valamint kizárólag az okostelefonokba épített szenzorokra és egyéb adatforrásokra támaszkodom.

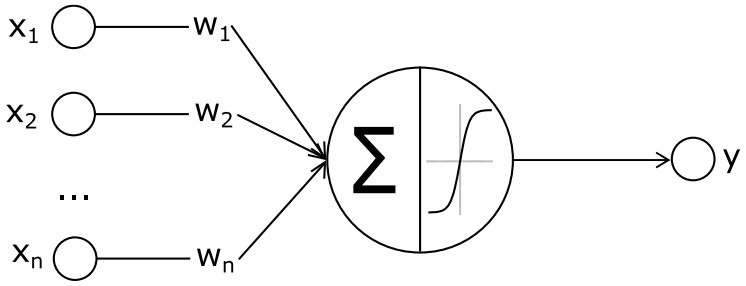
2.2 Mély neurális hálózatok elméleti alapjai

Az korábbi megoldásokról szóló fejezetben ismertetett irodalmak áttekintésén felül foglalkoztam a neurális hálózatok elméleti hátterével is [14], [5]. A dolgozatomnak nem célja az algoritmusok teljes és részletes bemutatása, hanem a működésükről ad egy általános leírást, ami szükséges lehet a dolgozat megértéséhez.

A neurális hálózatok olyan számítási feladatok modellezésére alkalmas rendszerek, amik képesek összetett összefüggéseket modellezni a hálózat bemenete, és a várt kimenet között. A neurális hálózatok számos feladat megoldásánál nemcsak

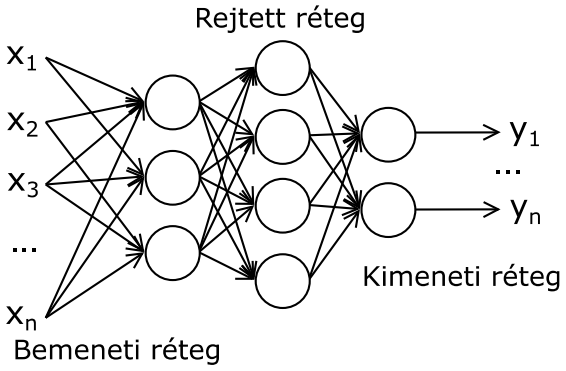
alkalmasnak, hanem jobbnak is bizonyulnak, mint a hagyományos algoritmikus számítási rendszerek. Ilyen feladatok például a különféle felismerési problémák, kezdve a viszonylag egyszerű nyomtatott számok és karakterek felismerésétől a jóval bonyolultabb kézírás-, kép- és egyéb alakzatfelismerésig [14]. További aktívan kutatott tématerület a gépi beszédfelismerés (Automatic Speech Recognition, ASR), ahol a legelterjedtebben használt rejtett Markov-modelleket (Hidden Markov Model, HMM) is leválthatják a neurális hálózatokon alapuló számítási módszerek [6]. A beszédfelismerés mellett a beszédszintézis (Text-To-Speech, TTS) kutatási területén is előtérbe kerültek a neurális hálózatok [8].

A neurális hálózat legkisebb alkotóegysége az elemi neuron (2.2. ábra), vagyis egy feldolgozó elem. A klasszikus elemi neuron egy több bemenetű, egy kimenetű egység, ami a bemenetek és a kimenet között valamilyen nemlineáris leképezést valósít meg. A neurális hálózat fontos tulajdonsága, hogy az egyes neuronok bemeneteinek súlyozott összegére hívja meg a nemlineáris aktivációs függvényt, ami a neuron kimeneti értékét szolgáltatja. A hálózat tanítása során ezeket a súlyokat kell úgy módosítani, hogy a kívánt kimenetet eredményezzék.



2.2. ábra - Elemi neuron

Egy neurális hálózat topológiáját egy irányított gráffal is megjeleníthetjük. A bemeneti neuronok a hálózat bemenetei, kimenetük a hálózat mélyebb rétegeiben található neuronok meghajtására szolgál. A rejtett réteg(ek) feladata, hogy a bemenő jeleket átalakítsa a kimenetnek megfelelő formára. A be és kimeneti rétegek között (2.3. ábra) tetszőleges számú rejtett réteg helyezkedhet el.



2.3. ábra – Előrecsatolt, teljes összeköttetésű neurális hálózat

A neurális hálózatokat a neuronokat összekötő rendszer fajtája alapján két nagy csoportba sorolhatjuk. Az egyik az előrecsatolt hálózatok, a másik pedig a visszacsatolt hálózatok (recurrent networks).

A gépi tanuló algoritmusok tanításánál három fő megközelítést ismertetek – főképp a neurális hálózatok szempontjából –, ezek az ellenőrzött tanulás (supervised learning), a megerősítéses tanulás (reinforcement learning), és a felügyelet nélküli tanulás (unsupervised learning).

Az ellenőrzött tanulási módszernél a tanítást összetartozó be- és kimeneti értékpárokkal – tanítómintákkal – végezzük. A hálózat feladata, hogy megtanulja a mintapontpárok segítségével a kívánt bemenet-kimenet leképezést. Mivel a folyamat végrehajtása során a kívánt kimenetek ismertek, a hálózat válasza összehasonlítható a kívánt válasszal. Az összehasonlítás eredményét felhasználhatjuk a hálózat olyan módosítására, hogy a hálózat tényleges kimenete és a kívánt kimenet közti eltérés minél inkább csökkenjen. Az ellenőrzött tanulási folyamat általában iteratívan, az ismeretek – mintapontok – fokozatos felhasználásával történik.

A megerősítéses tanulás során nem állnak rendelkezésünkre a kívánt kimenetek konkrét értékei, hanem csak annyit tudunk, hogy a modell válasza helyes-e vagy sem. Ez az információ a tanítás során arra elegendő, hogy megtudjuk: szükséges-e a modell módosítása, viszont a módosítás mértékének meghatározására már nem elegendő. A megerősítéses tanulás jól alkalmazható olyan esetekben ahol a modell egy bemenetre nem csupán egy jó választ adhat.

A felügyelet nélküli tanulásnál nem állnak a rendelkezésünkre az adott bemenetekhez a kívánt válaszok, a hálózatnak ezek nélkül kell valamilyen leképezést kialakítania a bemenet és a kimenet közt. A hálózatoknak azt kell megvizsgálniuk, hogy

van-e a bemeneti adatokban valami hasonlóság, van-e az adatok között korreláció, különválaszthatók-e az adatok csoportokra, osztályokra. Az ilyen módszerrel tanított hálózatok képesek önmaguk módosítására a kategorizálás sikeressége érdekében, ezért ezeket a hálózatokat önszervező hálózatoknak (self-organizing networks) is szokták nevezni.

A neurális hálózatok fontos tulajdonságai közé tartozik a tanulási képesség. A neurális hálózatok a tanulási folyamat során a hálózat súlyait javítani, adaptálni tudják a tanítás során felhasznált mintapontokkal [14]. A tanulás során a paraméterek módosításának célja, hogy az adott bemenetekre a kívánt válaszokat adja, hogy később kizárólag a bemeneti, a tanítás során nem felhasznált adatokra is helyes kimenetet adjon (regressziós és osztályozási problémák esetén is).

A neurális hálózatok tanulási képessége lehetővé teheti azt is, hogy egy már megfelelően betanított, jól működő rendszer a változó körülményekhez képes legyen igazodni, tehát a hálózat képességeit fejleszteni tudják.

Az előzőek mellett célszerű röviden megemlíteni a hálózat súlyainak beállítására használható hiba-visszaterjesztéses algoritmus (back-propagation) működésének hátterét is [15]. Az algoritmus az egyes tanítási iterációk (tanítóepochok) után a hálózat súlyfüggvényeinek módosításával igyekszik a modell hibáját csökkenteni.

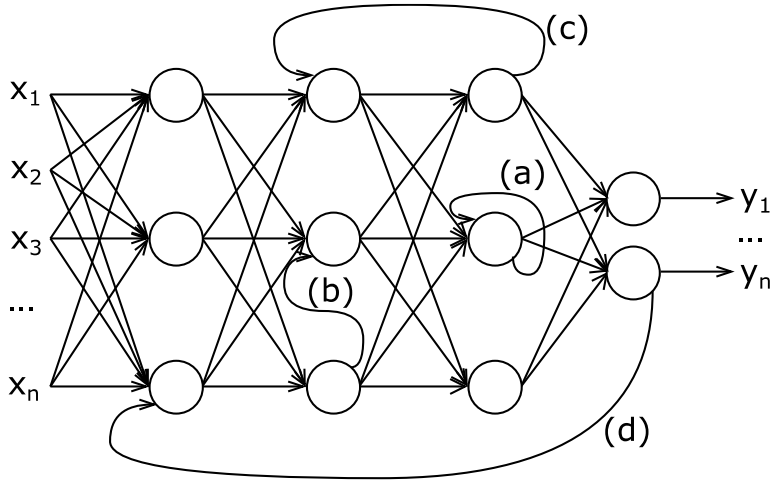
2.2.1 Előrecsatolt neurális hálózatok

Előrecsatoltnak nevezünk egy neurális hálózatot, ha a hálózat topológiáját leíró irányított gráf nem tartalmaz „kört”. Egy ilyen hálózatra mutat példát a 2.3. ábra. Az ábrán látható hálózat három aktív réteget tartalmaz, melyben az első réteg 3, a második (rejtett) réteg 4, a harmadik – jelen esetben a kimeneti – réteg pedig 2 processzáló elemet tartalmaz.

2.2.2 Visszacsatolt neurális hálózatok

A visszacsatolt hálózatok előnye az előrecsatolt hálózatokhoz képest, hogy nem csak a bemenetükre kerülő adatok és a kívánt kimenetek közti statikus leképezés

megtanulására képesek, hanem a hálózat válasza a régebbi bemeneti és/vagy kimeneti értékektől is függ ([14] 8. fejezet – Időfüggő (szekvenciális) hálók).



2.4. ábra - Visszacsatolt hálózat (forrás: [14] 1.9 ábra alapján)

A visszacsatolásoknak több fajtáját is megkülönböztetjük. A 2.4. ábra mutatja be az ismertetett visszacsatolásokat. Elemi visszacsatolásnak (2.4. ábra (a)) nevezzük azt a kapcsolatot, amikor egy réteg egy neuronjának kimenete közvetlenül az egyik saját bemenetére van csatlakoztatva. Laterális vagy rétegek közti kapcsolatnál (b) a valamelyik réteg neuronjainak kimenetei ugyanazon réteg bemeneteire kapcsolódnak (nem értve ide a neuron saját magára történő visszacsatolását). A rétegek közti visszacsatolások (c) több réteget tartalmazó hurkot hoznak létre a hálózatban. Globális visszacsatolás esetén (d) pedig a hálózat kimenetét csatoljuk vissza a bemenetére, így az éppen kiértékelésre kerülő minta az előző mintára adott kimenet értékétől is függ.

2.2.3 Konvolúciós neurális hálózatok

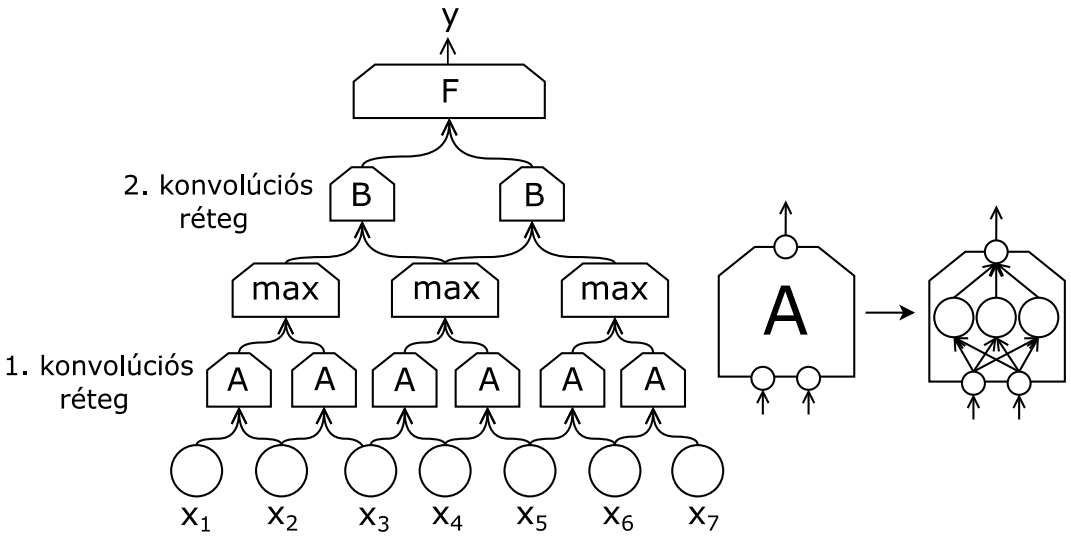
A konvolúciós neurális hálózatokat először a képfeldolgozási feladatok megoldásánál használták, konkrétan kézzel írott irányítószám felismerésénél [15], de a beszédfelismerésnél és idősorok vizsgálatánál is hasznosnak bizonyult [8] [7].

A konvolúciós neurális hálózatok olyan előrecsatolt hálózatok, amik három fő újítást vetnek be időbeli és térbeli összefüggések modellezése céljából. Ezek a helyi feldolgozó egységek (local receptive fields), megosztott súlyok (shared weights), időbeli vagy térbeli alulmintavételezés (temporal or spatial subsampling).

A helyi feldolgozó egységek (local receptive fields), használatával (2.5. ábra – „A”-val jelölt egységek) a neuronok alapvető jellemzőkiemelést tudnak elvégezni amiket

a magasabb szintű rétegek fel tudnak használni. A megosztott súlyokkal (az ábrán „A”-k súlyai megegyeznek) el lehet érni, hogy az adatokon (kép részlete, idősor részlete stb.) a feldolgozóegységek ugyanazt a műveletet hajtsák végre. A súlyok megosztásával csökkenthető a szabad paraméterek száma, ezáltal jobb általánosító képességet lehet elérni.

Gyakran előfordulhat olyan helyzet, hogy a modellezés szempontjából nem érdekes a jellemző pontos helye, amíg a többi jellemzőhöz képesti relatív helyzete megmarad. Emiatt bevezették az ún. max-pooling [16] rétegeket, amik a kinyert jellemzők közül a maximális értékűt tartják meg a feldolgozóegységek egy kis egymással szomszédos részhalmaza felett (2.5. ábra– „max” jelzésű egységek). A max-pooling rétegek megadják, hogy az egyes jellemzők jelen voltak-e az előző réteg egyes régióiban, de a pontos helyét nem.



2.5. ábra - Példa konvolúciós hálózatra (bal) és egy local receptive field kifejtése (jobb) (forrás: [17] alapján)

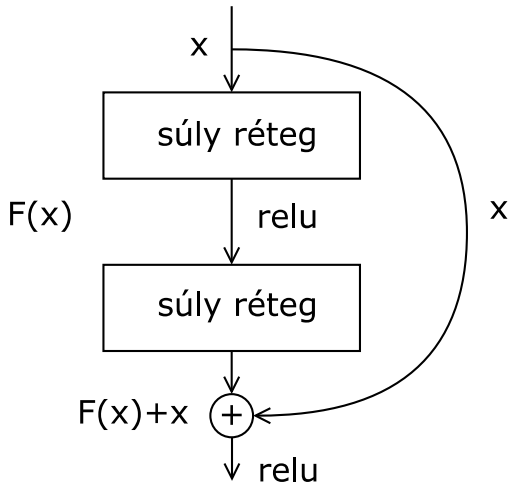
A 2.5. ábrán látható hálózat két konvolúciós réteget tartalmaz. Az ábra szerinti konvolúciós rétegek csak az előző réteg kimeneteinek 2 hosszú környezetét veszik figyelembe az ábra jobb áttekinthetősége érdekében, az adott feladattól függően viszont általában nagyobb környezetet célszerű figyelembe venni. A 2.5. ábra jobb oldali részén egy „A” jelű feldolgozó egység részletesebb megjelenítése látható. A „B”-vel jelölt egységek a második konvolúciós réteg feldolgozó egységei. Az ábrán „F”-el a teljes összeköttetésű (fully connected) előrecsatolt réteget jelöltem.

Összefoglalva egy konvolúciós réteg a bemenetére adott adatokat kisebb részekre bontva vizsgálja, azokon szűréseket és lényegkiemelést is végezhet. A konvolúciós réteghez kapcsolódhatnak további konvolúciós rétegek, visszacsatolt vagy teljes összeköttetésű rétegek is, amik a kívánt kimenet leképzését végzik. Több konvolúciós réteg esetén a rétegek egyes feldolgozóegységeinek bemeneteit az őt megelőző réteg kis szomszédságból álló egységeitől kapja (2.5. ábra – „B” jelzésű egységek).

2.2.4 Reziduális hálózatok

A rejtett rétegek számának növelésével nehezedhet a hálózatok tanítása az elenyésző gradiens (*vanishing gradient*) miatt [18]. Egy bizonyos pont után a teszthalmazon visszamérhető hiba növekedni kezdhet, viszont a vártakkal ellentétben ez nem feltétlenül a túltanulás hibája [19]. Vegyünk például egy kevésbé mély előtanított hálózatot, amihez hozzáveszünk további rétegeket. Amennyiben ezek a további rétegek mindössze egy identitásfüggvényt (a bemenettel megegyező kimenetet adnak) valósítanak meg, akkor a hálózat hibájának nem kellene többnek lennie, mint a kiegészítés előtt.

Ahelyett hogy abban bizakodnánk, hogy a legrosszabb esetben is az identitásfüggvényt fogják leképezni a „hozzávett” rétegek, explicit módon lehetővé tehetjük ezt úgy, hogy az eredeti $F(x)$ leképezés kimenetét kiegészítjük $F(x) + x$-re (2.6. ábra).



2.6. ábra - Reziduális blokk (forrás: [19] alapján)

Az ötlet mögött az a feltételezés, hogy a reziduális kapcsolatoknak köszönhetően a hibavisszaterjesztés során a gradiens a mélyebb rétegekbe is le tud jutni és így ezen rétegeket is hatékonyan tudjuk tanítani [20]. A megvalósítás során célszerű a rétegeket blokkokba szervezni úgy, hogy a következő blokknak átadjuk az előző blokk bemenetét is. Az architektúra hatékonysága empirikus módon is igazolt [19] [20] [21] [22].

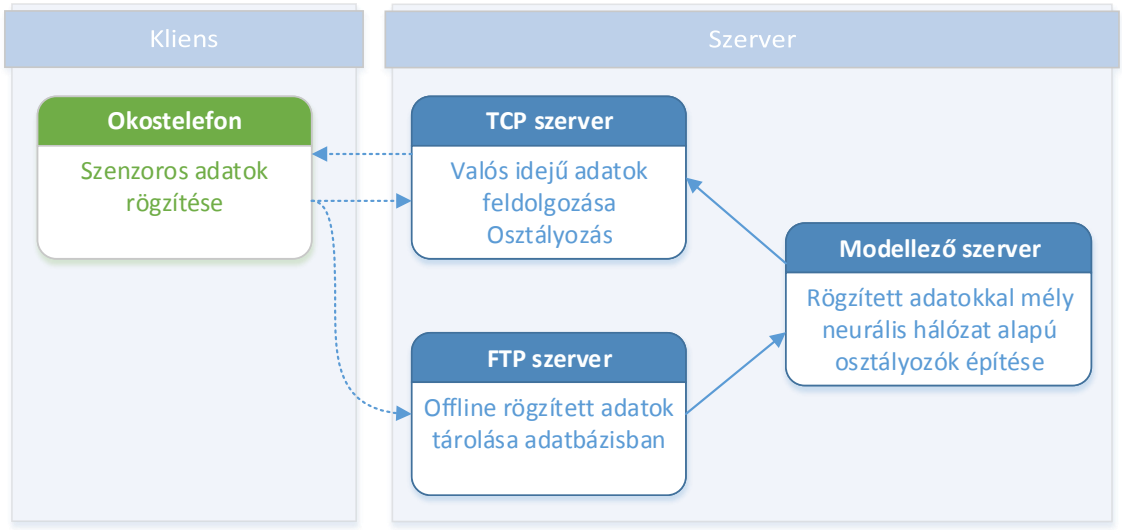
3 Rendszerterv

Jelen munka előzményeként, a BSc szakdolgozatom elkészítése során kialakítottam egy Google Android alapú adatrögzítő alkalmazást, amivel 14 tesztalany lépéseinek háromdimenziós gyorsulásadatait rögzítettem, miközben az adatrögzítést végző okostelefon az emberek nadrágzsebében volt [23]. A rögzített adatokat többféle módszer használatával feldolgoztam, majd a Random Forest, SVM, LDA és KNN tanulóalgoritmusokat tanítottam velük. Elkészítettem egy kísérleti mintarendszert, aminek célja az volt, hogy a felhasználót a járásának szenzoros adatai alapján megkülönböztesse más emberektől. Az elkészített mintarendszer a tanulóalgoritmusok és a feldolgozott adatok legjobb kombinációjából állt elő és a felhasználót átlagosan 94% pontossággal ismerte fel helyesen, míg a tévesen azonosított felhasználók aránya 1,09% volt. A mintarendszerben a gyorsulásadatokat autokorrelációval dolgoztam fel úgy, hogy minden ablak 10 másodpercnyi lépésadatot foglalt magába, a feldolgozott adatokkal pedig a KNN tanulóalgoritmust tanítottam. A szakdolgozatom elkészítésekor az adatok feldolgozását elsősorban Matlab-ban, a különböző modellek tanítását pedig Python-ban végeztem. A szakdolgozathoz elkészített adatrögzítő alkalmazás képes az okostelefonokba épített gyorsulásmérő szenzor értékeit rögzíteni, és a rögzített adatokat egy erre a célra regisztrált FTP szerverre feltölteni. Jelen munka előzményeként készített szakdolgozatom külön egészet alkot. A TDK dolgozat elkészítése során nem az volt a cél, hogy a korábbi munkámat kibővítsem, hanem most egy sokkal általánosabb célt fogalmazok meg, amit mély neurális hálózat alapú gépi tanuló algoritmusokkal vizsgálok.

Az előzetes munkáim valamint az irodalomkutatásban megismert korábbi megoldásokhoz képest az itt összeállított megoldások között lényeges különbség, hogy az adatokon nem végzek semmi féle transzformációt (előfeldolgozást) [1] [2] [10] [11] [12], a tanulóalgoritmusok a nyers szenzoradatokkal dolgoznak. A felhasználói élmény szempontjából további könnyebbség lehet, hogy a felvázolt megoldással nem szükséges az okostelefonon felül további eszközöket használni, ellentétben néhány korábbi megoldással [1] [10].

A tervezett rendszer célja egy olyan alkalmazáscsomag biztosítása, ami képes a felhasználó tevékenységeit nagy pontossággal felismerni. A tevékenységek felismerését a rendszer a gépi tanulás egyik aktívan kutatott technikájával, a jelenleg legnagyobb pontosságot és flexibilitást nyújtó mély neurális hálózatokon (Deep Neural Network,

DNN) alapuló modellekkel végzi [24] [25] [26]. Az általam kidolgozott rendszer kliens-szerver architektúra szerint épül fel, a rendszer vázlatos felépítése a 3.1. ábrán látható. Az okostelefonos kliens végzi a szenzoros adatok rögzítését és ezeket a rögzített adatokat „valós időben” és utólag – aktív internetkapcsolat hiányában – a rögzítés befejeztével is képes eljuttatni a szervernek.



3.1. ábra – A tervezett rendszer általános blokkdiagramja

A TCP szerver végzi az okostelefontól kapott adatok „valós idejű” feldolgozását és a tevékenységek osztályozását, míg az utólagos adatfeltöltést az FTP szerveren keresztül lehet elvégezni. A modellező szerver állítja elő a rögzített adatok osztályozásához használható mély neurális hálózatokon alapuló modelleket, amik az FTP szerverre feltöltött adatok alapján állnak össze.

3.1 Adatok rögzítése és előkészítése

Az adatrögzítés módszertanának kidolgozásakor fontos szempont volt a rögzítendő tevékenységek listájának meghatározása. Ennek meghatározásakor figyelembe vettem, hogy a feldolgozott irodalmakban milyen tevékenységek rögzítését részesítették előnyben, valamint azt is, hogy milyen tevékenységeket tudnék a lehető legegyszerűbben rögzíteni.

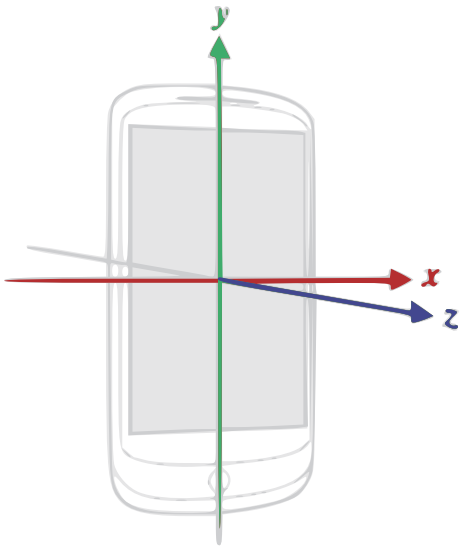
Az irodalomkutatásban részletesen ismertetett irodalmak közül több is vizsgálta [1], [2], [12] az állás, sétálás, lépcsőzés tevékenységeket így én is felvettem ezeket a rögzítendő tevékenységek listájára. Ezeken felül autóval, vonattal és metróval történő közlekedési tevékenységeket is kijelöltem adatrögzítésre. Mind az autózás, mind a

vonatozás tevékenység „leszármaztatható” az ülés tevékenységből, az adatok vizsgálata során így arra is lehetőség lesz, hogy kiderüljön: lehet-e különbséget tenni a két tevékenység adatainak jellemzői között. A végleges rögzítendő tevékenység-listám tehát a következő: állás, sétálás, lépcsőzés felfelé, lépcsőzés lefelé, autózás, vonatozás, metrózás. Az [1] irodalomban látott megoldással szemben célom hogy ne kelljen utólag felcímkézni a rögzített adatokat, hanem azt a felhasználó már rögzítéskor elvégezhesse.

3.1.1 Android alapú adatrögzítő alkalmazás

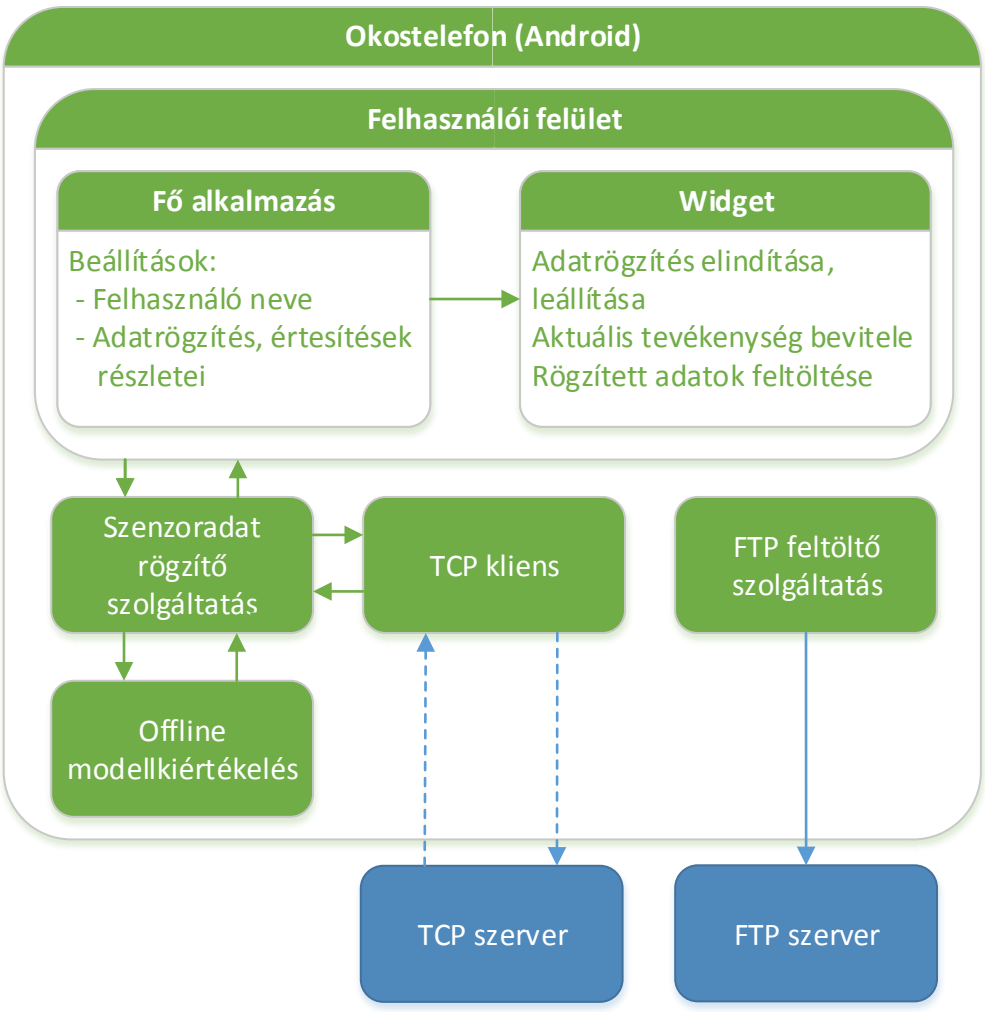
Az adatrögzítő alkalmazás tervezésekor a szakdolgozatom során megvalósított Google Android platformra fejlesztett applikációt vettem alapul. Az alkalmazás továbbfejlesztése során fontos szempont volt az egyszerű használhatóság, vagyis hogy az adatrögzítést végző felhasználónak a program használata ne jelentsen nehézséget valamint cél volt az is, hogy a mobil eszközök akkumulátor- és mobiladat használata ne növekedjen zavaró mértékben.

Az alkalmazás a mobil eszközökbe épített gyorsulásmérő, orientációs szenzor, fényérzékelő, közelségérzékelő és hőmérő által mért értékeket képes rögzíteni. A gyorsulásmérő és az orientációs szenzor a 3.2. ábrán látható három tengely mentén méri az adatokat. A szenzoros információkon felül további adatforrásokat is naplóz, konkrétan a pozíciós adatokat (GPS vagy GSM cellainformációk alapján), valamint a WiFi kapcsolat állapotát és az esetlegesen kapcsolódott hálózat nevét (SSID).



3.2. ábra - Android szenzor API által használt koordináta rendszer (forrás: [27] alapján)

Ahhoz, hogy az adatokat úgy tudja rögzíteni az alkalmazás, hogy már a rögzítés idején címkézni lehessen az egyes tevékenységeket, ki kellett egészíteni az alkalmazást egy olyan felülettel, ahol ez egyszerűen megtehető. Ezen felül célszerű volt, hogy ne csak egy előre beégetett tevékenység-listából lehessen választani, hanem hogy az alaplistához az adatrögzítést végző felhasználó egyéb, tetszőleges tevékenységeket is felvehessen. Lehetőséget biztosítok online és offline adatrögzítési módra is, valamint a rendszer valós felhasználása közbeni tesztelhetőségének érdekében kliens-szerver architektúrát dolgoztam ki a szenzoradatok beküldésére, azok feldolgozására és az eredmények okostelefonra történő visszaküldésére. A fejlesztés és az adatrögzítés során egy tanszéki Samsung Galaxy S4-es és a saját tulajdonban lévő Motorola Moto G készülékemet használtam. A rendszer blokkdiagramja és az egyes komponensek között zajló kommunikáció a 3.3. ábrán látható.



3.3. ábra - Adatrögzítő alkalmazás blokkdiagramja

Az előzőek során elkészített adatrögzítő alkalmazás Eclipse fejlesztői környezet alatt készült, azóta viszont az Android Studio lett a Google által hivatalosan támogatott környezet [28]. Emiatt átalakítottam a korábbi alkalmazásprojektet úgy, hogy az új fejlesztői környezettel lehessen elvégezni az okostelefonos fejlesztéseket. A fejlesztéseket 8-as verziójú Java fordítóval és az Android Studio aktuálisan legfrissebb verziójával végeztem (1.3-2.2).

3.1.1.1 Adatrögzítő alkalmazás osztályhierarchiája

A mobil eszközös fejlesztés kezdetén megterveztem, hogy milyen továbbfejlesztési lépések szükségesek az adatrögzítő alkalmazás megvalósításához. Ki kellett bővítenem a naplózott adatforrások körét, valamint a felhasználói felületet is átalakítottam úgy, hogy minél egyszerűbben tegye lehetővé címkézett adatok rögzítését. A szenzoradatokkal kapcsolatos feladatokat megnehezíti, hogy a különböző készüléktípusokba a gyártók más-más érzékelőfajtákat, azon belül is különböző fizikai jellemzőkkel bíró hardvereket építenek be. Az azonos típusú – például gyorsulásmérő –, de különböző specifikációjú szenzorok az esetleges kalibrációs hibák vagy mérési pontatlanságok miatt is megegyező körülmények mellett más értékeket mérhetnek, ezért fontos szempont volt a több felhasználóval történő adatrögzítések végzésekor, hogy az egyes készüléktípusokat is rögzítse az alkalmazás.

A tervezés során fontos volt az online történő adatátviteli módszerek biztonságossá tétele valamint az adatformalom minimalizálása. Az online adatrögzítések során TCP/IP feletti titkosított SSL kapcsolaton keresztül történő adatátvitelt választottam, kényelmi opcióként pedig offline adatrögzítési lehetőséget is elérhetővé tettem. Offline esetben az okostelefon memóriakártyájára rögzíti az alkalmazás a mért értékeket és azt az adatrögzítés végeztével lehet egy FTP szerverre feltölteni. Az offline módszer akkor lehet hasznos, ha az adatrögzítés közben a felhasználónak nincs internetkapcsolata vagy nem akarja elhasználni a korlátozott mobilinternet keretét.

Az ismertetett osztályok teljeskörű leírása a korábban elkészített szakdolgozatomban szerepel [23], itt a jelen munka elkészítése során tervezett és kivitelezett főbb fejlesztéseket ismertetem osztályok szerinti bontásban.

SensorService: Az osztály egy Androidos szolgáltatást (Service) valósít meg, és feladata, hogy az okostelefon egy háttérfolyamataként elvégezze az adatok rögzítését. Az

adatrögzítések során több feldolgozott irodalomhoz hasonlóan [1] [10] [11] [12] 50 Hz mintavételi frekvenciát használok. Az újonnan felvett adatforrások: hőmérő, közelségérzékelő, fényérzékelő, WiFi kapcsolat információi, lokációs adatok. Az osztály felelőssége továbbá az is, hogy a widget-en keresztül visszajelezze a felhasználónak az aktuálisan rögzített tevékenységet. A kliens-szerver architectúra megvalósításához az osztály elvégzi a rögzített adatok JSON formátumra alakítását is, és ha online adatbeküldési módban van az alkalmazás akkor a TCP kliensen keresztül elküldi az adatokat. Ez az osztály kezdeményezi a TCP kapcsolat kiépítését és a kapcsolat megszakadása esetén is ez az osztály kísérli meg a kapcsolat újbóli felépítését. Amennyiben offline modellkiértékelést használunk, ez az osztály hívja meg a lokális modell függvényeit.

SettingsFragment: Az osztály felelőssége az alkalmazással kapcsolatos beállítások kezelése. Itt valósítom meg azt a funkciót, hogy naplózza az alkalmazás az egyes készüléktípusokat. Ezt a feladatot úgy oldom meg, hogy az adatrögzítést végző felhasználó nevének változása esetén rögzítem az eszköz típusát, ami a későbbiek során a rögzített adatokkal együtt feltöltésre kerül az adatok gyűjtésére használt FTP szerverre. Amennyiben online adatbeküldés történik, akkor a kommunikáció kezdetén elküldött bejelentkező információkkal együtt küldöm el a készülékkel kapcsolatos adatokat. Az osztály kezeli a szabadon változtatható tevékenységlistát is, ezért további feladata, hogy a lista megváltozása esetén a felhasználó által létrehozott widget-eket értesítse a változásról.

WidgetOnOff: Ennek az osztálynak a feladata, hogy megvalósítsa a címkézett adatrögzítés elindítását és leállítását. Ezen felül ez az osztály valósítja meg a tevékenységkategóriák dinamikus megjelenítését is. Alapértelmezés szerint egy sorban három tevékenységhez rendelek megérinthető gombokat, amennyiben több tevékenységet kell megjeleníteni, ezeket mindig újabb sorokba teszem. Fontos a felhasználó felé történő visszajelzés is, az aktuálisan rögzített tevékenységet zöld színnel emelem ki.

CSVWriter: Az osztály felelőssége, hogy a SensorService által rögzített adatokat egy megfelelően strukturált CSV fájlba írja. Az osztály lehetőség szerint az okostelefonokba helyezett SD kártyára írja az adatokat, ennek hiányában pedig az eszközök belső tárhelyére. Az osztályt az új adatforrások kezelésével kellett kiegészíteni.

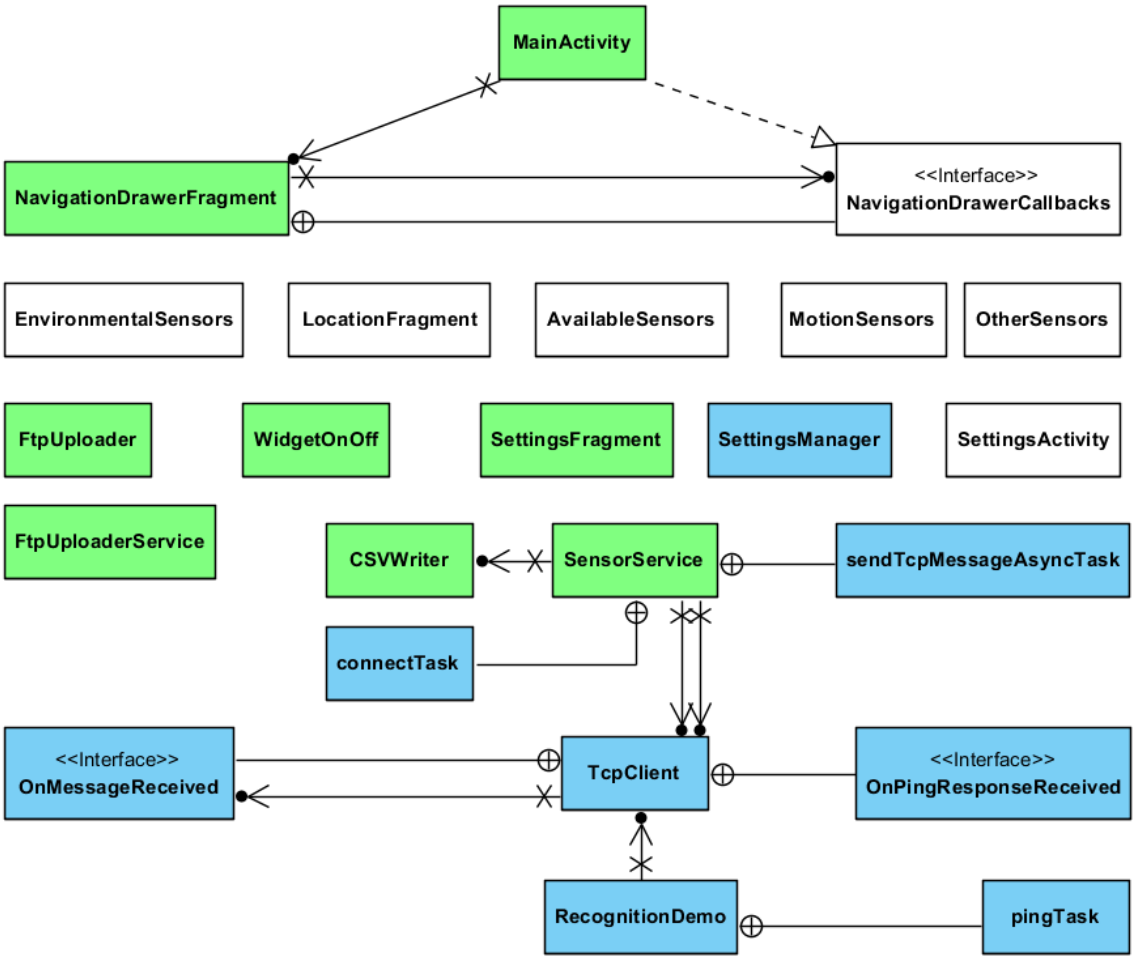
Az osztály által gyűjtött adatok kerülnek bele a valós időben beküldött JSON üzenetekbe is.

FtpUploaderService: Ez az osztály egy Androidos szolgáltatást (Service) valósít meg. Feladata, hogy a SensorService osztály által rögzített adatokat feltöltse egy előre konfigurált FTP szerverre. A sikeres feltöltésről a widget tájékoztatja a felhasználót.

TcpClient: Az osztály feladata a TCP kliens kezelésével járó metódusok megvalósítása. Az osztály statikus változókban tárolja a szerver elérhetőségeit (IP cím és portszám), valamint ez az osztály végzi az SSL kapcsolat kezelését is. Az osztály valósítja meg a specifikált üzenetküldési és fogadási protokollt valamit az adatok tömörítését is.

RecognitionDemo: Az osztály egy Androidos Fragment osztályt valósít meg. Feladata a tevékenységfelismerés szolgáltatás elindítása és megállítása, valamint a felismert tevékenységekkel kapcsolatos statisztikák megjelenítésének kezelése.

Az alkalmazás általános osztálydiagramja a 3.4. ábrán látható, kék színnel a teljesen új osztályokat jelöltem; zölddel a már meglévő, de módosított; fehérrel pedig a módosítás nélküli osztályokat.



3.4. ábra – Androidos alkalmazás osztálydiagram

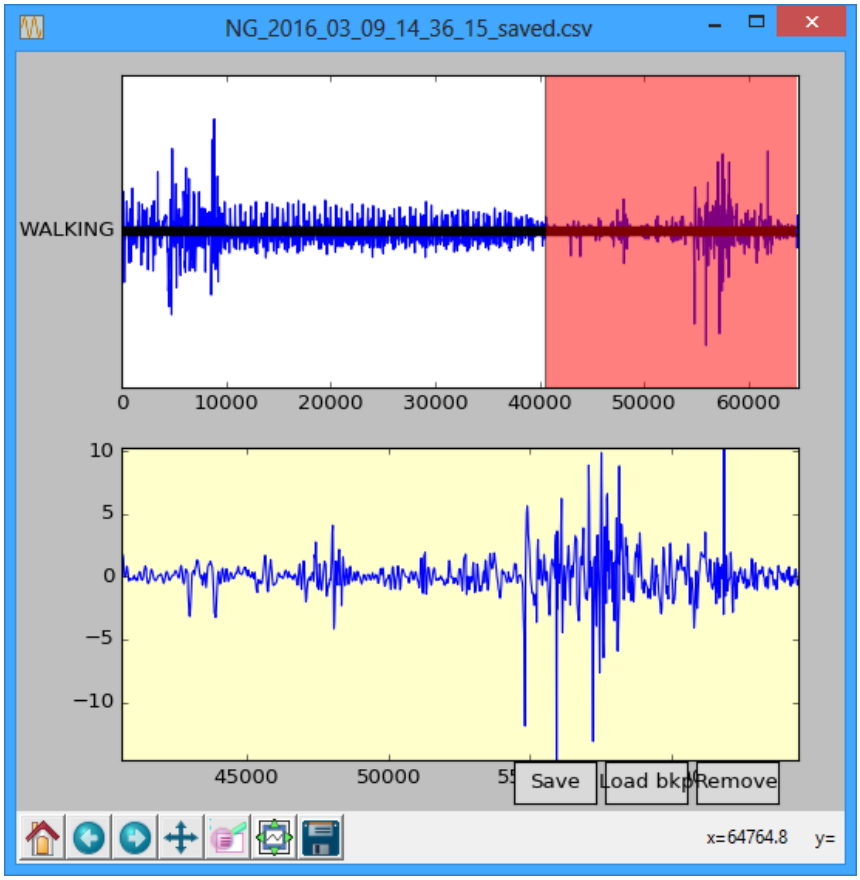
3.1.2 Adatbázisok

A rögzített adatokból tanító adatbázisokat hozok létre. Ezekben tárolom az adatrögzítések során gyűjtött nyers adatokat. Minden tanítóadatpont mellett eltárolom a tanítóvektorhoz tartozó szöveges címkét is. Az adatokat a rögzítés módja miatt elsődlegesen sok különálló CSV fájlban tárolom, de a felhasználás módja miatt feldolgozom az adatokat egy közös adatbázisba, amit a tanítások során használok fel.

3.1.3 Adatfájlok szerkesztésére szolgáló alkalmazás

Az adatok rögzítése során többször előfordult, hogy az adatrögzítést végző személy elfelejtette leállítani a rögzítést, így sok rosszul felcímkézett adat is keletkezett. Ennek a problémának a megoldására elkészítettem egy olyan Python alapú alkalmazást, amivel egyesével meg lehet vizsgálni a rögzített és feltöltött adatfájlokat és a szemmel láthatóan rosszul címkézett adatokat el lehet távolítani. Ez akkor is hasznos, amikor az adatokat rögzítő ember jelzi, hogy mikor felejtette el leállítani a rögzítést. Az alkalmazás

felhasználói felületének elemeit az EasyGui [29] Python csomag használatával állítottam össze.



3.5. ábra - Rögzített adatfájl szerkesztése

A 3.5. ábrán az elkészített alkalmazás látható futás közben. Az alkalmazás elindítása után egy általános tallózó ablak jelenik meg, ahol ki lehet választani a szerkeszteni kívánt adatfájlt. A fájl kiválasztása után megjelenítem a gyorsulásmérővel rögzített adatsort, valamint megjelölöm az adatsorhoz rendelt címkeértékeket is. Amennyiben eltávolítandó adatokat tartalmaz a fájl, ezeket a felső ábrán az egérrel ki lehet jelölni (pirossal jelezve), majd kizárólag a kijelölt szakasz adatait megjeleníti az alkalmazás a képernyő alsó részén.

Amennyiben valóban el kívánjuk távolítani ezt a részt, a „Remove” gombra kell kattintani. Az adatok szerkesztése után a „Save” gombbal lehet véglegesíteni a módosításokat. Az adatfájl módosítása esetén az eredeti fájlról biztonsági mentést is készít a program, ami bármikor visszatölthető.

3.2 Kliens-szerver architektúra

Az adatok valós időben történő rögzítéséhez valamint a rendszer éles használatban történő teszteléséhez kliens-szerver architektúrát dolgoztam ki. A tervezés során fontos volt a kommunikációs protokoll megbízhatósága, könnyű bővíthetősége, a biztonságos adatátvitel valamint az adatforgalom minimalizálása. A megbízhatóság az idősor jellegű adatok rögzítése miatt volt fontos, a pontos mérésekhez nem engedhetők meg hiányzó adatok. A kutatás közben gyakran kerülhetünk olyan helyzetbe, hogy új adatok átvitele válik szükségessé, ezt a kommunikációs protokoll egyszerű bővíthetőségével lehet megvalósítani. A biztonságos adatátvitel manapság alapvető fontosságú mivel személyes adatokkal dolgozunk. A lehető legkisebb adatforgalom az adatrögzítést és a tesztelést végző felhasználók érdekeit szolgálja, hogy a korlátozott adatforgalmú mobilinternet kapcsolatukat minél kevésbé vegye igénybe a szolgáltatás.

3.2.1 Kommunikációs protokoll

A kommunikáció alapjának TCP/IP kapcsolatot választottam. A TCP (Transmission Control Protocol – átvitelvezérlő protokoll) protokoll megbízható, sorrendhelyes bájtfolyamot biztosít a hálózati végpontok között [30]. A TCP dinamikusan alkalmazkodik az összekapcsolt hálózatok tulajdonságaihoz és nagymértékben ellenáll a meghibásodásokkal szemben.

A tervezés során a biztonságos adatátvitel megvalósíthatóságához az SSL (Secure Sockets Layer) alapú titkosítást választottam. A TCP feletti biztonságos adatátvitel megvalósítására dolgozták ki az SSL protokollt, így könnyen összeilleszthető a tervezés korábbi fázisában választott hálózati protokollal. Az SSL többek között lehetőséget nyújt kölcsönös hitelesítésre a kliens és a szerver között, titkosított kommunikációra és az adatok sértetlenségének biztosítására.

A könnyű bővíthetőség megvalósítására az üzeneteket JSON (JavaScript Object Notation) string-eket használok. A JSON [31] egy kisméretű, emberek által is könnyen írható és olvasható szöveges alapú összetett adatok tárolására is használható protokoll. A JSON string-ek felépítése programozási nyelv független, így könnyebb átjárást biztosít a különböző programnyelvekre épülő kliens és szerver között. Bár a szöveg alapú számátvitel nagyobb méretű adatcsomagokkal jár a bináris számábrázolási módokhoz képest, a kutatás e pontján a bővíthetőségi szempontokat vettem fontosabbnak, az adatmennyiség csökkentésére más módszereket használtam fel. Az üzenetek protokollját

úgy alakítom, hogy a későbbiekben a szövegesről bináris alapú számábrázolásra történő módosításokat könnyen el lehessen végezni.

Mivel az adatok szöveges formában kerülnek továbbításra, a hálózaton keresztül küldött adatok mennyiségének csökkentésére a zlib [32] veszteségmentes tömörítő könyvtárcsomagot választottam, de lehetőséget biztosítok tömörítetlen adatok továbbítására is amiatt, mert a rövid üzeneteket nem mindig éri meg tömöríteni.

A következőkben ismertetem a tervezett kommunikációs protokoll részleteit. A protokoll üzenetek kezelésére szolgáló metódusokat a kliensben és a szerverben is meg kell valósítani. A TCP üzeneteken alapuló protokollokban tapasztalataim szerint célszerű az üzenetek elején megadni az átvinni kívánt üzenet hosszát, így mindig tudjuk, hogy még mennyi adatot kell kiolvasnunk a byte folyamból. A tervezett protokollban minden üzenet a következőképpen épül fel:

```
<4 karakter hosszú vezérlési parancs> <üzenet hossza: 32 bites int> <üzenet bájtjai>
```

A vezérlési parancs jelenleg arra szolgál, hogy tudassa a fogadó féllel, hogy tömörített vagy tömörítetlen adatot küld-e. A bináris számábrázolási módra történő áttérés esetén itt lehet megadni az éppen küldött adat fajtáját. A vezérlési parancsok a következők lehetnek:

- `"UCJS"` – UnCompressedJSon – Tömörítetlen JSON esetén
- `"COJS"` – COmpressedJSon – Tömörített JSON esetén.

Az üzenet hossza adja meg az átviendő üzenet bájtjainak számát egy 32 bites big-endian integer formájában.

A következőkben a kommunikáció során használt JSON üzenetek formátumát ismertetem. A JSON protokoll tervezésénél különös figyelmet szántam a könnyű bővíthetőség támogatására, figyelemmel tartva azt, hogy ne okozzon hibát az, ha egy üzenet a várton felül további adatokat is tartalmaz.

Minden üzenet egy JSON objektumból áll, melynek gyökérelemei között lennie kell egy `"messageType"` kulcsú string-nek, mely tartalmazza az üzenet típusát, az objektum további eleme(i) tartalmazhatják az átvinni kívánt adatokat.

Az mért szenzoros adatok beküldésének formája a következő: az üzenet típusa (message type) `"sensorData"`, a JSON objektum gyökerében egy `"sensorData"` kulcs

alatt kell lennie egy JSON tömbnek ami egy vagy több JSON objektumot tartalmaz. Ezekben a JSON objektumokban egy-egy időegységhez tartozó adatok találhatók. A szolgáltatás minden ilyen módon fogadott adatot fájlba rögzít. Példa egy szenzoradatokat tartalmazó üzenetre:

```
{
  "messageType":"sensorData",
  "sensorData":[
    {
      "TIME_ms":"1476686178922"
      "ACC_X":"-2.63602757",
      "ACC_Y":"4.42476034",
      "ACC_Z":"6.55868769",
      ...
    },
    {
      "TIME_ms":"1476686178962",
      ...
    },
    ...
  ]
}
```

A felismerési eredmények visszaküldéséhez a JSON objektumban az üzenet típusa `"recognitionResult"`. Az üzenetben `"recognitionResult"` kulcs alatt kell lennie a felismerési eredményeket tartalmazó JSON tömbnek. Példa egy felismerési eredményeket tartalmazó üzenetre:

```
{
  "messageType":"recognitionResult",
  "recognitionResult":{
    "results":[
      "BIKE",
      "BIKE",
      "BIKE"
    ]
  }
}
```

A szerver lehetőséget biztosít a szolgáltatás működésének ellenőrzéséhez is. Ehhez egy `"ping"` típusú üzenetet kell küldeni a szervernek:

```
{
  "messageType":"ping"
}
```

A szerver a `"ping"` típusú üzenetre egy `"pong"` típusú üzenettel válaszol. A JSON objektum gyökerében az üzenet típusát leíró string mellett egy `"pongData"` nevű további információkat hordozó kulcsot is tartalmazhat. Példa egy ilyen üzenetre, ami egy „Service OK!” szövegű állapotinformációt is tartalmaz:

```
{
  "pongData":{
    "message":"Service OK!"
  },
  "messageType":"pong"
}
```

Az adatok rögzítéséhez szükséges bizonyos felhasználói adatok ismerete, ezeket egy `"loginData"` típusú üzenetben kell beküldeni a szolgáltatáshoz történő kapcsolódás után. Ebben az üzenetben szerepel többek között az adatrögzítést végző személy neve, a rögzítés során használt okostelefon gyártója, típusa, Android verziója, nyelve és az alkalmazás verziószáma is. Példa egy bejelentkező üzenetre:

```
{
  "messageType":"loginData",
  "loginData":{
    "app_version_code":9,
    "device_manufacturer":"motorola",
    "device_model":"XT1072",
    "android_api":23,
    "locale":"en_GB",
    "pref_recorder_name":"cb",
    "pref_wifi_auto_upload":false,
    "pref_timedelay":"5",
    "pref_location_passive":false,
    "pref_activities":"Walking;Car;Stairs;Running;Standing;Sitting",
    "pref_location":true,
    ...
  }
}
```

3.2.2 Kliens és szerver

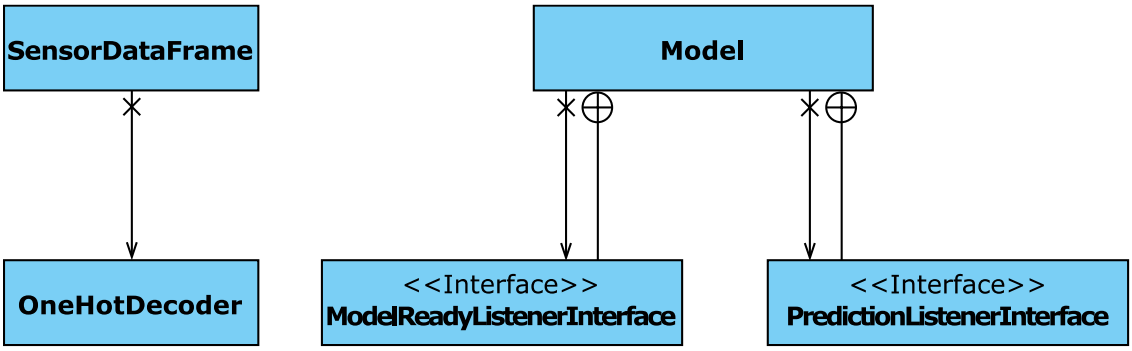
A szerver oldali szolgáltatást Python nyelvben valósítottam meg, mert az adatok feldolgozása, a tanulóalgoritmusok tanítása és a modellek futtatása mind Python környezetben történik. A szolgáltatáshoz az Androidos kliens SSL kapcsolattal tud kapcsolódni. A kliens a sikeres kapcsolódás után küldheti a szervernek a mért értékeket, ami fájlba menti a kapott adatokat, valamit felismerési eredményeket is visszaküldhet a kliensnek.

3.3 Modellek futtatása okostelefonon (offline)

Ahhoz hogy a modelleket internetkapcsolat nélkül is lehessen futtatni megvizsgáltam az okostelefonon való modellfuttatások lehetőségeit. Munkám ezen szakaszát külön alfejezetben ismertetem, mert a tervezés és megvalósítás során arra

törekedtem, hogy a modellek futtatására szolgáló kódok önállóan – az adatrögzítő és demonstrációs alkalmazástól függetlenül – is használhatók legyenek esetlegesen más alkalmazásokba történő beépíthetőség érdekében.

A megvalósítandó feladatnak három osztályt terveztem, ezek áttekintő osztálydiagramja a 3.6. ábrán látható. A `Model` osztály magát a neurális hálózat modellt valósítja meg. Ennek az osztálynak lehet átadni a kiértékelni kívánt értékeket, ami alapján visszakapunk egy predikciót (becsült tevékenységformát). Az osztálynak képesnek kell lennie a szerveroldalon betanított modellek felépítésének és súlyainak fájlból való beolvasására. Az osztály két interfészt biztosít az őt használó osztályok számára, amikkel értesülhetnek a modell sikeres betöltéséről (`ModelReadyListenerInterface`) és a futtatások eredményéről (`PredictionListenerInterface`).



3.6. ábra - Offline modellfuttatás osztálydiagram

A `SensorDataFrame` osztály feladata az egyenként érkező minták tárolása, amíg a kiértékelési ablakmérethez szükséges adatmennyiség össze nem gyűlik. További feladata a tárolt értékek standardizálása (bővebben 4.5.1.4 - Standardizálás) is.

A `OneHotDecoder` osztály feladata a modell eredményeit visszaalakítani a kívánt osztálycímkékre (itt a felismert tevékenység szöveges címkéje, erről bővebben: 4.4 - Adatbázisok).

A modell felépítésén és a modellhez tartozó súlyokon felül képesnek kell lennie a rendszernek további adatok beolvasására, ezek lehetnek például a standardizáláshoz szükséges értékek, a felhasználni kívánt szenzorok listája vagy akár az osztálycímkék szövegei, stb. Ezen értékek tárolásához is egy JSON struktúrát dolgoztam ki, aminek gyökerében egyetlen JSON objektum áll. Az objektumon belül kulcs-érték párok képében

JSON tömbök, egész vagy lebegőpontos számok szerepelhetnek. Példa egy ilyen JSON struktúrára:


```
{
  "scaler_std":[
    2.4522879563492768,
    1.59203171197179,
    2.4822302623543813,
    1.4750598345759482,
    18.19105269612947,
    9.032898854672846
  ],
  "scaler_mean":[
    2.114430940298025,
    -0.0003125096671385598,
    -0.00031806758130055636,
    0.0003902589508391045,
    6.16147590264085,
    2.17578659857129
  ],
  "labelbinarizer":[
    "BIKE",
    "CAR",
    "RUNNING",
    "STAIRS",
    "STANDING",
    "WALKING"
  ],
  "sensors":[
    "ACC_X",
    "ACC_Y",
    "ACC_Z",
    "ORI_X",
    "ORI_Y",
    "ORI_Z"
  ],
  "noverlap":170,
  "n_DatapointsInTimestep":200
}
```


3.4 Mély tanuló architektúrák

Ebben a fejezetben ismertetem a munkám során használt konkrét mély neurális hálózatokon alapuló tanulóalgoritmusok architektúráit. A dolgozat elkészítésekor Python-ban végeztem az adatok feldolgozását, a modellek tanítását és azok tesztelését is. Munkám kezdetén több különböző neurális hálózatokat megvalósító osztálykönyvtárat is kipróbáltam: Neurolab [33], Pylearn2 [34], Keras [35]. Az utóbbi kettő a Python-os Theano [36] nevű osztálykönyvtárra épül, vagyis képes a kódok GPU-n történő futtatására is, a Keras csomaghoz ezen felül a TensorFlow [37] keretrendszert is lehet

használni mely a PC-s környezet mellett mobil (Android és iOS) és beágyazott rendszereken is futtatható. A GPU alapú programozás előnyei közé tartozik, hogy a párhuzamos kódok futtatását nagyobb hatékonysággal végezheti, mint a CPU, így nagyságrendekkel is meggyorsíthatja a számításigényes műveleteket amellett, hogy olcsóbb erőforrás a CPU-nál.

Ahhoz hogy megismerkedjek a neurális hálózatok gyakorlati alkalmazásával, megvizsgáltam az ún. XOR problémát és a szinusz és koszinusz függvények közelítését. Az XOR vagyis a kizáró vagy művelet nem lineárisan szeparálható, ezért a korai neurális hálózatok nem tudták megtanulni ezt a műveletet. Később azonban a hiba-visszaterjesztéses (back-propagation) algoritmusok megjelenésével sikerült megoldani a feladatot. Az XOR probléma tanulmányozása után áttértem a szinusz és koszinusz függvények approximációjára. A neurális hálózatok képesek csaknem tetszőleges nemlineáris leképezések tetszőleges pontosságú approximációjára, vagyis, a neurális hálózatok univerzális approximátornak tekinthetők [14].

A kipróbált osztálykönyvtárak közül a Keras-t választottam a további munkák elvégzésére. A Neurolab nem futtataható GPU-n, a Keras előnye a Pylearn2 megoldásához képest, hogy sokkal „közelibb” az elmélethez a programozása, illetve a PyLearn2 fejlesztését időközben leállították.

A munkám megoldásakor a Python 3.5-ös és a Keras 2.0.8-as verzióját használtam.

3.4.1 Előrecsatolt neurális hálózatok

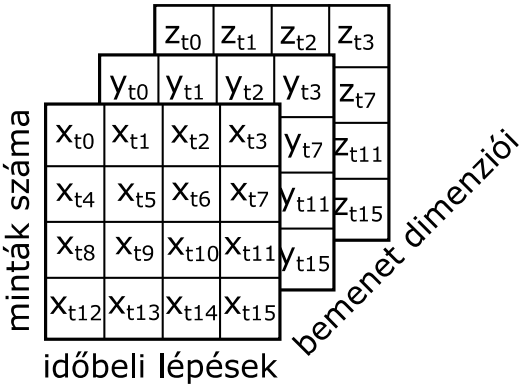
A feladatom megoldása során elsődlegesen nyers idősor jellegű adatokkal tanítom a neurális hálózatokat. Az ilyen adatsorokat célszerű valamilyen előfeldolgozás után átadni a teljes összeköttetésű előrecsatolt hálózatoknak. Ezt a feladatot a visszacsatolt és a konvolúciós hálózatok segítségével valósítom meg, a kívánt kimenet – ebben az esetben az osztálycímkék – leképzését pedig teljes összeköttetésű hálózatokkal végzem.

3.4.2 Visszacsatolt neurális hálózatok

A klasszikus visszacsatolt hálózatok hátránya, hogy hosszú távú összefüggéseket nem, vagy csak nehezen tudnak megtanulni valamint a hiba visszaterjesztéses algoritmus futtatása során a többszörös visszacsatolások miatt a túl kicsi illetve a túl nagy gradiensek „eltűnhetnek”, illetve „felrobbanhatnak” (*vanishing-exploding* gradients [38]).

Az LSTM (Long Short-Term Memory) [39] alapú visszacsatolt hálózatok előnye, hogy kiküszöböli a gradiensek problémáját, valamint további előnye, hogy hosszabb távú összefüggések megtanulására is képes.

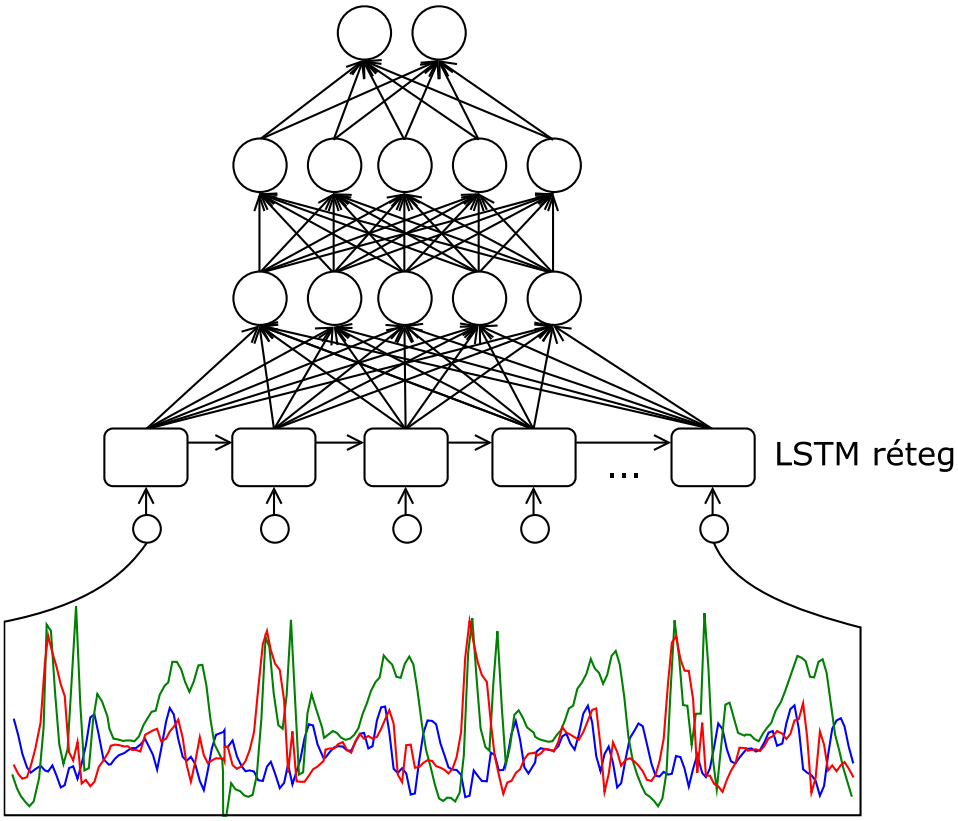
Az LSTM hálózatot megvalósító kód a Keras könyvtár része, így ezt használva végeztem a saját adatokkal való tanításokat. A Kerasban a visszacsatolt hálózatok tanításához (és teszteléséhez is) háromdimenziós tömbökben kell megadni az adatokat, az egyes időablakok szerinti felbontásban. A tömb méretének és alakjának a következő formát kell követnie: (minták száma, időbeli lépések, bemenet dimenziószáma), a felépítés bővebben a 3.7. ábrán látható.



3.7. ábra - Adatokat tartalmazó mátrix összeállítása a Keras LSTM modulja számára

Az LSTM hálózatok tanítása során többdimenziós idősor jellegű adatokat kellett a 3.7. ábrán látható formátumra alakítani. Az átalakítás során lényeges szempont volt, hogy mennyi egymás után következő adatpontot fogjak össze egy időablakba. A mérések során megvizsgáltam, hogy milyen hatással van az időablak hossza a felismerés pontosságára.

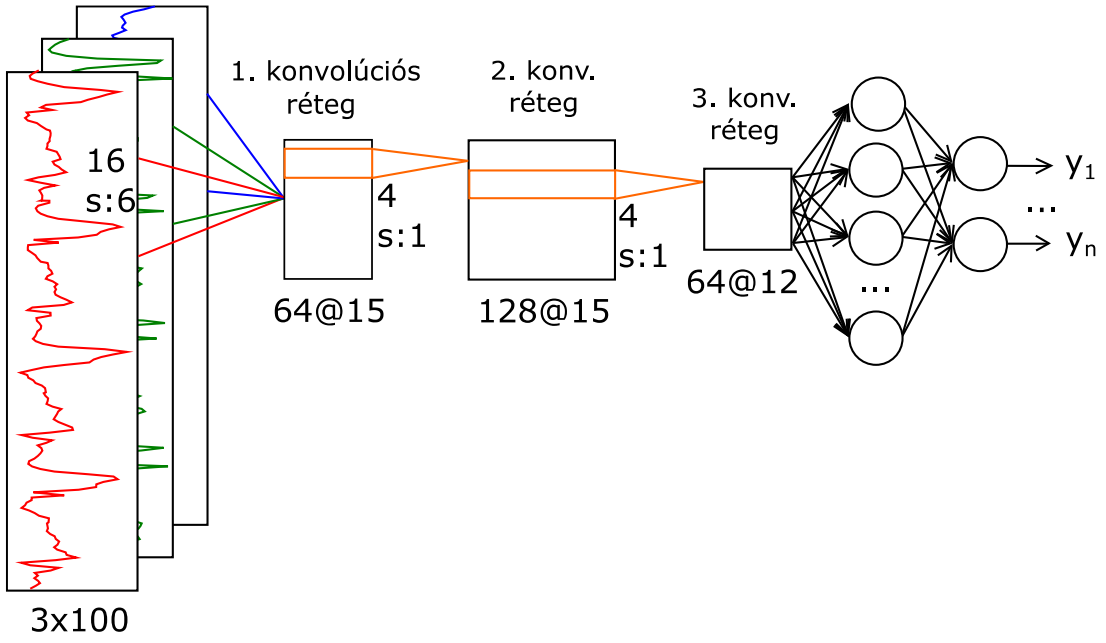
A hálózatokat úgy építettem fel, hogy az egy időegységnyi nyers adatot egy LSTM cellákból álló réteg bemenetére adom. Az LSTM rétegre egy teljes összeköttetésű előrecsatolt hálózatot építettem, aminek fő feladata az osztálycímkék leképzése (3.8. ábra).



3.8. ábra - Egy időegységnyi nyers adat feldolgozása LSTM hálózattal

3.4.3 Konvolúciós neurális hálózatok

A Keras egydimenziós konvolúciós hálózatainak használatakor hasonlóképpen kell összeállítani a bemeneti mátrixot, mint az LSTM hálózat esetében, vagyis itt is időegységnyi nyers adatot adok a hálózat bemenetére. Az első konvolúciós réteg az időegységnyi adat részleteit vizsgálja, azokból többféle szűrő segítségével alacsonyszintű lényegkiemeléseket végez. Az első konvolúciós réteg kimenetére további konvolúciós rétegeket is építettem. Itt megvizsgáltam a modellek pontosságát többféle konvolúciós rétegszámmal és hiperparaméter értékekkel is. A konvolúciós rétegek fölé teljes összeköttetésű előrecsatolt hálózatot építettem, ami a kimeneti osztálycímkék előállításáért felel.

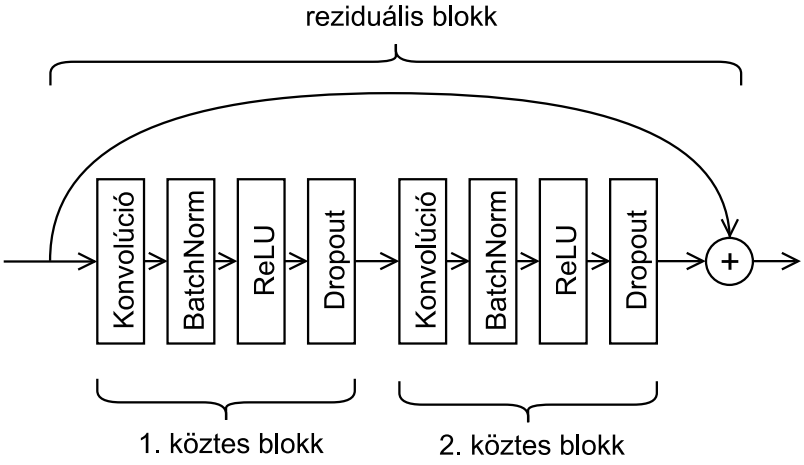


3.9. ábra - Példa architektúra egy időegységnyi nyers adat feldolgozására konvolúciós hálózattal. A@B jelentése: A darab B dimenziós szűrő, C; és s:D jelentése: C hosszúságú környezet, D eltolással.

A 3.9. ábrán egy időegységnyi adat feldolgozását végző hálózat látható. A bemenő adatok 100 darab 3 dimenziós mintapontokból állnak (ábra bal oldala). A bemeneti adatok egy részét vizsgálja az első konvolúciós réteg (64 darab 15 dimenziós szűrő, 4 hosszúságú környezet, 1 eltolással), ami további konvolúciós rétegekkel is össze van kötve. A kimenet leképzését az ábra jobb oldalán látható teljes összeköttetésű előrecsatolt hálózat végzi.

3.4.4 Reziduális hálózatok

A reziduális hálózatok használatakor a reziduális összeköttetéseket és az azok által „átugrott” rétegeket reziduális blokkoknak nevezzük. Nevezzük továbbá az „átugrott” rétegeket köztes rétegeknek. A köztes rétegeket a következőképp választottam meg: konvolúciós réteg, batch normalizációs réteg, aktivációs réteg (ReLU aktivációs föggvénnyel) és dropout réteg (a batch normalizációról és a dropout rétegről bővebben a 4.5.1 fejezetben). Nevezzünk egy ilyen rétegsorozatot egy köztes blokknak. Minden konvolúciós blokkot úgy állítottam össze, hogy az két köztes blokkot tartalmazzon, erre látható egy példa a 3.10. ábrán. Mivel a feladat megoldásakor idősor jellegű adatokkal dolgoztam, minden konvolúciós réteg egydimenziós konvolúciót valósít meg.



3.10. ábra - Reziduális blokk felépítése

Az ilyen reziduális blokkokból felépített hálózat hiperparamétereit [9] alapján úgy választottam meg, hogy a konvolúciós rétegek szűrőinek száma mindig $f * k$ ahol f a kezdeti szűrőszám, k pedig 1-ről indul és minden negyedik reziduális blokknál egyel nő. Az előző építőelemeken felül minden második reziduális blokk után egy MaxPooling réteget is alkalmaztam. A konvolúciós ablakméretet, a kezdeti szűrőméretet és a reziduális blokkok számát a hiperparaméterek optimalizálása során határoztam meg szem előtt tartva a minél jobb osztályozási pontosságot. A korábbiakhoz hasonlóan a kimenet leképzését egy teljes összeköttetésű előrecsatolt hálózat végzi.

4 Megvalósítás

A fejezet első részében ismertetem a megvalósított adatrögzítő alkalmazás implementálásának lényeges részeit. Következőnek összefoglalom a tanítások során használt tanítóadatbázisok felépítését, majd a tanulóalgoritmusok gyakorlati használatának részleteit.

4.1 Okostelefon alapú adatrögzítő alkalmazás

A munka kezdetén továbbfejlesztettem a szakdolgozatom során a Google Android platformra készített adatrögzítő alkalmazást úgy, hogy azzal különböző tevékenységek szenzoradatait lehessen rögzíteni. A már korábban elkészített alkalmazás képes az okostelefon gyorsulásmérőjének adatainak rögzítésére 50 Hz-es mintavételi frekvenciával. Az alkalmazás a rögzített adatokat először az okostelefon SD kártyájára (annak hiányában a belső tárhelyre) rögzíti, majd egy erre a célra regisztrált FTP szerverre tölti fel, CSV fájlok formájában. Az adatrögzítést egy kezdőképernyőre tehető widget segítségével lehet elindítani és megállítani, valamint az adatok feltöltését is a widget-tel lehet kezdeményezni. Az előbbi módosításokon felül még megvalósítottam, hogy olyan egyéb szenzorokat és adatforrásokat is rögzítsek, amikkel az irodalomkutatás során is találkoztam.

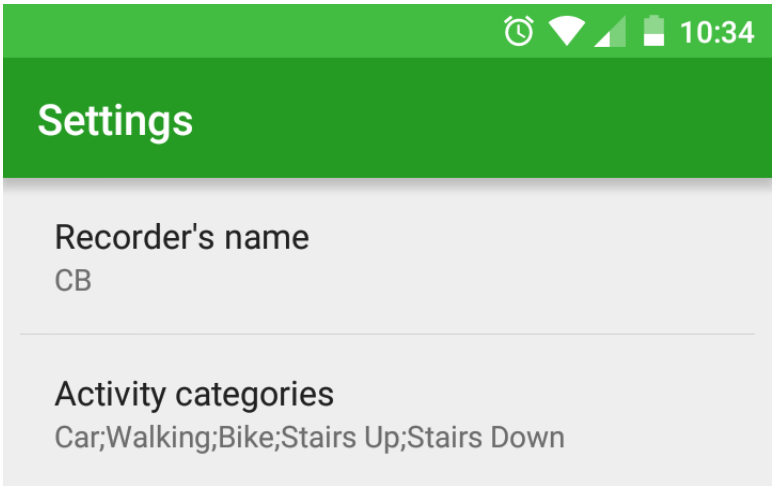
Az alkalmazást úgy fejlesztettem tovább, hogy az adatrögzítést végző felhasználó képes legyen az adatrögzítés idején felcímkézni a rögzített tevékenységeket, így elkerülve az [1] irodalomban látott utólagos címkézés szükségességét. Ezen felül megvalósítottam azt is, hogy a tevékenységek listájára a felhasználó tetszőleges számú új tevékenységet is felvehessen, valamint módosíthassa az alapértelmezetten beállított tevékenység-listát is.

Az alkalmazás minimális Android API szintje 11 (Android 3.0), mert innentől támogatja a rendszer a szenzorok mintavételi idejének pontos beállításának lehetőségét.

4.1.1 Felhasználói felület az adatrögzítéshez

A tevékenységek rögzítésének lehetőségét célszerű volt a korábbiak során már sikeresen használt widget-es felülethez hasonlóan megvalósítani. Ahhoz, hogy a tevékenységek listája bővíthető legyen, a választható opciókat az alkalmazás beállításai közt mentem el. A beállítások osztályt (*SharedPreferences*) és az azt megjelenítő képernyőt már korábban megvalósítottam, így egy új menüpontot kellett felvennem. A

tevékenységek listáját egy *EditTextPreference* beállítási opcióval oldottam meg. Ez az elem szövegesen tárolja a beállításokat, itt lehet felsorolni a tevékenységeket, egymást pontosvesszőkkel elválasztva. A beállítások képernyő a 4.1. ábrán látható. Az alapbeállításként megjelenő tevékenységek a következők: Car;Walking;Bike;Stairs Up;Stairs Down ezek tetszőlegesen megváltoztathatók.



4.1. ábra - Az adatrögzítést végző személy nevének és a tevékenységek listájának beállítása

Az elmentett tevékenység listát ezután meg kellett jeleníteni az alkalmazás widget-én. A widget-en az adatrögzítés leállítására és az adatok FTP szerverre történő feltöltésének elindítására szolgáló gombokon kívül megjelenítem a rögzíthető tevékenységcímkéket is. Mivel ezek a címkék futásidő alatt változhatnak, a felületet dinamikusan bővíthetőre készítettem, a megfelelő gombokat is a widget példányosítása után kellett futásidőben létrehozni (*RemoteViews*), és elhelyezni őket a felhasználói felület megfelelő pontjain.

A megvalósítás e pontján olyan akadályba ütköztem, hogy nehézkes volt a dinamikusan létrehozott gombok eseményeinek a kezelése. A gondot úgy sikerült megoldani, hogy amikor létrehozok egy új gombot, annak az eseménykezelőjének az *Intent*-jének az *action* attribútumában eltárolom az aktuális tevékenység nevét, és később ezzel azonosítom az eseményeket. A tevékenységekhez tartozó gombok futásidőben történő elhelyezéséhez kiegészítettem a meglévő képernyő elrendezést (layout) egy üres tárolóelemmel, amit soronként töltök fel a gombokkal úgy, hogy egy sorban legfeljebb három gomb legyen. A gombok hozzáadásáért felelős függvény a widget-et megvalósító osztály *onUpdate* függvényében kerül meghívásra. Ez azért fontos, mert a tevékenységek

listáját tartalmazó beállítás módosítása után kérvényezem a widget frissítését, hogy a módosítás után rögtön a legfrissebb tevékenységek közül lehessen választani.

A widget-et úgy alakítottam ki, hogy az egyes tevékenységeket megérintve automatikusan elinduljon az adatrögzítés. Ez egy kényelmi funkció, hogy ne kelljen külön elindítani a rögzítést, valamint kiküszöböli a címkézetlen adatok rögzíthetőségét. Az aktuális tevékenységet a *SensorService* osztály *currentActivity* nevű statikus változójában tárolom. Ennek a változónak az értékét módosítja a widget, és innen olvassa ki a rögzítendő tevékenységet az adatrögzítést végző szolgáltatás. A widget ezen felül színkiemeléssel jelzi az éppen rögzítésre kerülő tevékenység nevét (4.2. ábra).

4.2. ábra - Az elkészített widget a sétálás (walking) tevékenység rögzítése közben

Ahhoz, hogy a gyorsulásmérő szenzor értékein kívül más érzékelők méréseit is rögzítse az alkalmazás, a *SensorService* nevű osztályt kellett kiegészítenem. Az alkalmazás a kiegészítés után képes az okostelefonokba beépített fényérzékelő, közelségérzékelő és a hőmérő szenzor adatainak rögzítésére. A szenzoros értékeken kívül további lehetséges adatforrások információit is rögzítem. Ezek a pozíciós adatok (GPS vagy cellainformációk alapján), a WiFi hálózat állapota, és a hálózat neve, amihez az okostelefon kapcsolódva van, és a kijelző aktuális állapota vagyis, hogy be vagy ki van-e kapcsolva a kijelző.

4.1.2 Valós idejű kapcsolat

A tervezett valós idejű szolgáltatáshoz való kapcsolódáshoz egy TCP/IP felett működő SSL klienst kellett megvalósítani, ami képes kapcsolódni a szerverhez, annak továbbítani a rögzített szenzoros adatokat valamint feldolgozni a szervertől érkező felismerési eredményeket. A kliensnek ezeken felül követnie kell a korábban specifikált kommunikációs protokollt.

A TCP kliens kapcsolódási, üzenetküldési-fogadási feladatait valamint az üzenetekhez kapcsolódó protokollok kezelését egy Java alapú osztállyal oldottam meg. A TCP kliens feladatai közé tartozik az üzenetek küldése és azok fogadása. Míg az üzenetek küldésének idejét az okostelefonon futó alkalmazás választja meg, az üzenetek fogadására minden időpillanatban készen kell állni. Az Android rendszer nagyon érzékeny a UI (User Interface – Felhasználói felület) szálon hosszasan futó kódokra, ezért a TCP kliens kezelését több háttérszálon (background thread) kellett megoldani. A TCP üzenetek fogadásához egy állandóan figyelő szálat kell megvalósítani, de az üzenetek elküldése is hosszadalmas folyamat lehet az üzenetek hosszának és a hálózati kapcsolat átviteli sebességének függvényében. Ezeknek a feladatoknak a háttérben történő végrehajtásához az Android specifikus AsyncTask [40] osztályt használtam fel. Az AsyncTask osztállyal hosszan futó műveleteket végezhetünk a háttérben, melynek eredményét egyszerűen hozhatjuk a felhasználói felületet kezelő szál tudtára.

A beérkező üzenetek kezelését egy – a TCP kliens életciklusa során – folyamatosan futó AsyncTask-on belül oldottam meg. A kimenő üzeneteket szintén AsyncTask-ok használatával implementáltam viszont ezeket az üzeneteket nem egy folyamatosan futó háttérszál kezelte, hanem minden üzenetnél egy új osztálypéldány jön létre, ami az üzenet sikeres elküldése után leáll.

Az üzenetek tömörítése a kliensalkalmazásban volt a legfontosabb mivel itt gyűlik össze a feldolgozandó szenzoros adat, amit el kell juttatni a feldolgozást végző szervernek. A tömörítő eljárás hatékonyabban dolgozik nagyobb adatmennyiségekkel, ezért az adatokat nem azonnal továbbítja a kliens, hanem csak akkor, ha összegyűlt egy másodpercnyi. Bár ez egy másodperc késleltetést visz a rendszerbe, a megoldandó tevékenységfelismerési feladat nem olyan időzítéskritikus, hogy ezt ne engedhessük meg, viszont a kommunikáció során felhasznált kisebb adatmennyiség sok felhasználónak fontos szempont lehet. Méréseim alapján az itt használt tömörítési eljárással 68%-al csökkent az átvitt adatok mértéke.

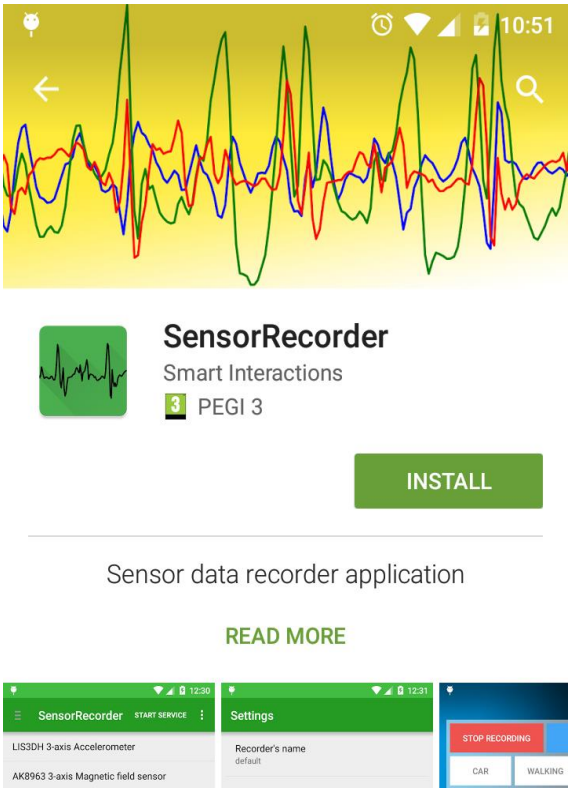
4.1.3 Felismert tevékenységek tárolása

Az alkalmazásban lehetővé tettem a felismert tevékenységekről való statisztikai adatok gyűjtését. Az információk tárolását egy SQL adatbázis használatával biztosítottam

[41]. Az adatbázisnak egyetlen táblája van, aminek minden rekordja egy-egy naphoz tárolja az aznap felismert tevékenységek típusát és mennyiségét.

4.1.4 Alkalmazás közzététele és frissítések kezelése

Az alkalmazást a több felhasználót is érintő kibővített adatrögzítések során elérhetővé tettem a Google Play Áruházban is. Az áruházban történő publikálás előnye, hogy egyszerűbben eloszthető az adatrögzítést végző emberek közt az alkalmazás, valamint a frissítés is a Play Áruházban már megszokott módon elvégezhető. A programot béta állapotban tettem közzé, így nem érhető el nyilvánosan, csak a tesztelésre történő jelentkezés után (ehhez Google Fiók szükséges). A tesztre viszont a megfelelő link[1] birtokában már bárki egyszerűen jelentkezhet egyetlen gombnyomással és ezután már megjelenik és telepíthető az Áruházból az alkalmazás (4.3. ábra). A telepítés menetét és az alkalmazás használatára vonatkozó utasításokat összefoglaltam egy használati útmutatóban, ezt elérhetővé tettem a felhasználók számára. Az alkalmazás fejlesztése közben 17 frissítést tettem közzé a Play Áruházon keresztül.



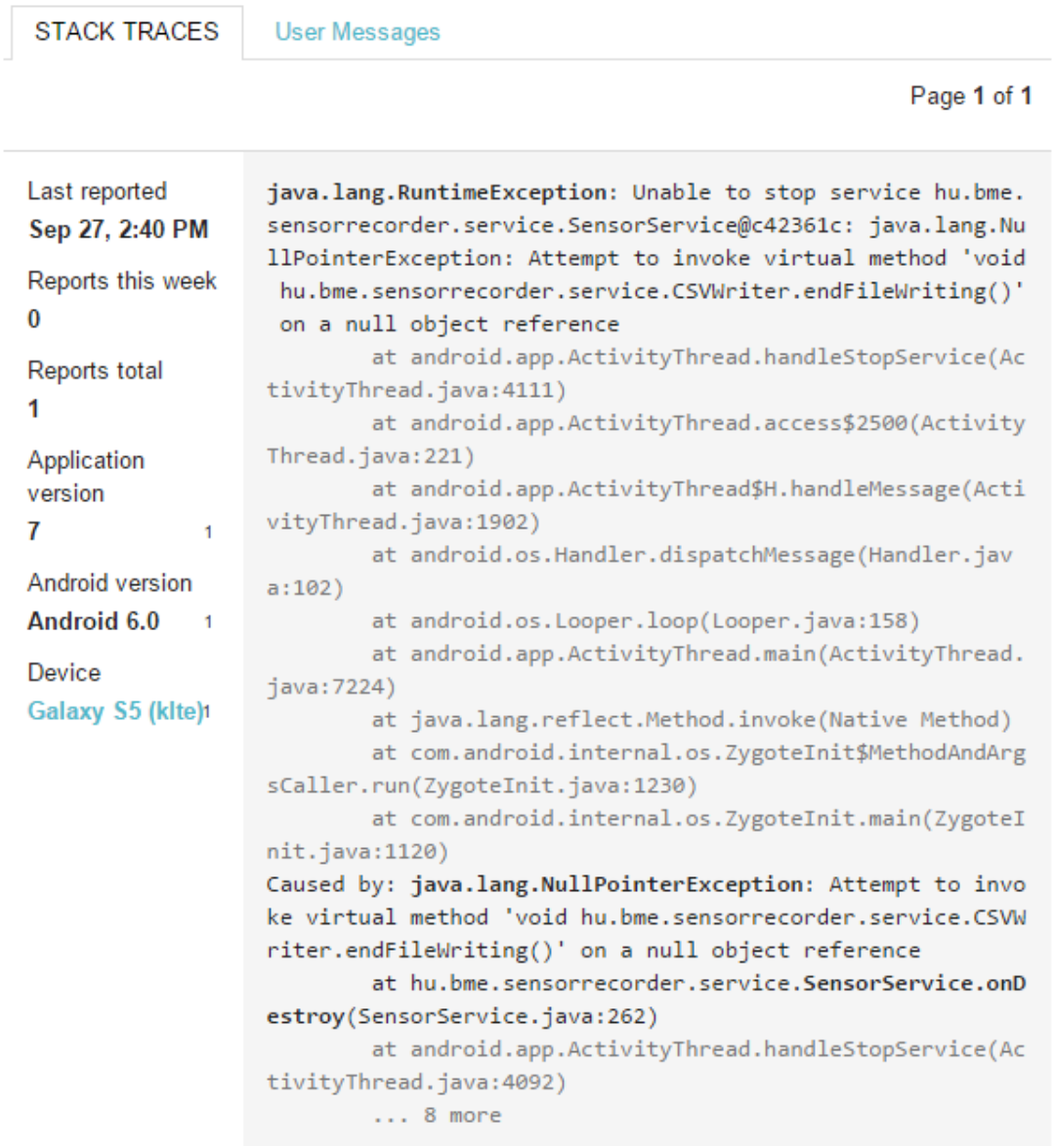
4.3. ábra -SensorRecorder alkalmazás a Google Play Áruházban

[1] https://play.google.com/apps/testing/hu.ca0phr.sensorrecorder

Az elkészült alkalmazás összesen 7532 sorból áll (a szakdolgozatomhoz képest 4390 új kódsor), a forráskódok 183 kB tárhelyet (a szakdolgozatomhoz képest 56 kB-al több) foglalnak.

4.1.5 Tesztelés

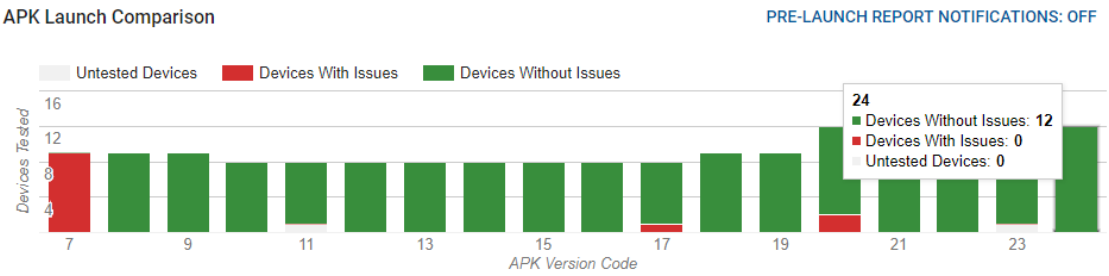
Az alkalmazást az adatrögzítést végző felhasználók a rögzítés módja miatt aktívan tesztelték, a hibákat a Google Play Áruház hibabeküldő rendszerén keresztül jelentették be. Az ilyen módon bejelentett hibákról a Google fejlesztői felületén (Developer Console) keresztül kapok értesítéseket. Ezen a felületen láthatók a hiba keletkezésének részletei, az eldobott kivételek típusa és a kivételt dobó sor száma, vagyis a részletes stacktrace adatok hasonlóan a fejlesztői környezetekben megszokottakhoz, ami nagy segítség volt a hibák feltárásában (4.4. ábra).



4.4. ábra – Hibabejelentés megjelenítése az Android Developer Console-ban

Az alkalmazást több szoftveres úton is teszteltem, először az Android Studio UI/Application Exerciser Monkey tesztjének használatával [42]. Ezt a tesztet emulátoron lehet futtatni és az alkalmazáson ál-véletlen (pseudo random) felhasználói felületi eseményeket (kattintás, érintés, gesztus és rendszerszintű események) hajt végre. Ilyen módon stressz tesztnek lehet kitenni az alkalmazást véletlenszerű, de megismételhető módon.

A második szoftveres teszthez a Google Play Áruházba való alkalmazások feltöltésére létrehozott rendszert használtam [43]. Ez a rendszer (Developer Console) minden feltöltött alkalmazásverziót letesztel (4.5. ábra) az előző Monkey teszthez hasonlóan, viszont ezek a tesztek valós okostelefonokon futnak. Ezek az eszközök a legtöbb használatban lévő Android verzión elvégzik a teszteket (jelenleg 4.4 – 8.0), és több gyártót és nyelvet is megvizsgálnak. Alkalmazáshiba esetén elérhető a fejlesztőnek a teljes stacktrace valamint egy videó is a hiba keletkezésének körülményeiről. Ezt az eszközt különösen hasznosnak találtam a fejlesztés során mivel olyan hibákat is javítani tudtam, amik a fejlesztéshez használt eszközökön nem jelentkeztek.



4.5. ábra - Google Developer Console - Összefoglaló ábra a tesztek eredményeiről

A fenti módszerekkel feltárt hibák javításait frissítéseken keresztül juttattam el a felhasználóknak.

4.2 Szerver megvalósítása

A szerver feladatai közé tartozik, hogy állandóan várakozzon bejövő kapcsolatokra, a kapcsolódott kliensektől folyamatosan fogadja és dolgozza fel a kapott adatokat, valamint futtassa le a beállított modellen az adatokat, aminek eredményét pedig küldje vissza a kliens részére. A kapott adatokat a feldolgozás során fájlba is rögzíti.

A szervert egy több szálon futó Python alkalmazás képében valósítottam meg, mert az adatok feldolgozása, a tanulóalgoritmusok tanítása és a modellek futtatása mind

Python környezetben történik. Minden újonnan kapcsolódott kliens feldolgozása egy új szálon történik, ezzel biztosítom, hogy egyszerre több kliens kiszolgálása is párhuzamosan történhessen.

A szerver megvalósítja a 3.2.1. fejezetben ismertetett üzenetküldési protokollt. A szerver egy tanszéki Windows 8.1-es számítógépen fut, amiben egy Cuda képes NVIDIA GeForce GT 640M LE GPU található, így a modellek futtatása is GPU-n történik. A számítógépre van továbbítva egy a BME hálózatába tartozó nyilvános IP cím megfelelő portja, így külső hálózatokból is elérhető.

4.3 Modellek okostelefonon történő kiértékelése

A Rendszerterv fejezethez hasonlóan a modellek offline kiértékelésének módját a megvalósítás ismertetésénél is különválasztottam a fő Androidos alkalmazástól. A részfeladat megvalósítása során több megközelítést is körbejártam. Első megközelítésben a TensorFlow által is ajánlott natív megoldást [44] vizsgáltam meg. A natív kódok futtatása főképp nagy számításigényű műveleteknél lehet előnyös [45]. A TensorFlow által biztosított fejlesztőeszközök Linuxra és MacOS-re érhetők el, ilyen konfigurációjú eszközök viszont nem álltak rendelkezésemre mobilfejlesztési célokra, ezért ezt a megközelítést egy Ubuntu 16.04. operációs rendszert futtató virtuális gépen próbáltam ki. A megoldás működött, viszont – részben a virtuális környezet miatt – egy egyszerű alkalmazás lefordítása is 20-30 percet vett igénybe, ami a mintaalkalmazás fejlesztése során jelentősen lassította az előrehaladást.

A második megközelítés a KerasJS [46] JavaScript alapú osztálykönyvtár volt, ami böngészőkben teszi lehetővé a TensorFlow backend-et használó Keras alapú modellek futtatását GPU támogatással. A JavaScript alapú könyvtár miatt a modellek futtatása egy HTML oldalon történik az alkalmazás egy háttérszálában. Az Android (Java) és a HTML oldal közti kommunikálást kellett megoldani, amire az Android rendszer a `JavascriptInterface` [47] megoldást kínálja. A módszerrel sikerült megvalósítani a tervezett `Model` osztályt.

A 3.3 fejezetben ismertetett tervek alapján megvalósítottam a `SensorDataFrame` és `OneHotDecoder` osztályokat is. A `SensorDataFrame` egy olyan interfészt kínál, amin

keresztül a mintaalkalmazásban futó `SensorService` nevű szenzoradatokat figyelő szolgáltatás el tudja juttatni a `Model` osztály egy példányának a mért értékeket. A kiértékelendő modell kimenetét pedig a `OneHotDecoder` segítségével egyszerű szöveges formában adja vissza a mintaalkalmazásnak.

4.3.1 Tesztelés

Az offline modellfuttatás teszteléséhez Unit (modul) teszteket alkalmaztam. A Unit tesztelés során a szoftver kisebb alkotóelemeit tudjuk hatékonyan tesztelni úgy, hogy az egyes modulokhoz teszteseteket rendelünk. Minden tesztesetnél definiáljuk a tesztelni kívánt bemeneti értékeket vagy eseményeket és az elvárt kimeneteket.

Az implementált tesztesetek részletesen megvizsgálják a fejezetben érintett modulok függvényeit. A `SensorDataFrame` osztály teszteléséhez ellenőrzöm a minták felvételét és tárolását, valamint azok lekérését az igény szerinti átlapolások kezelésével együtt. Az osztály feladata a standardizálás elvégzése is, így ezt is ellenőrzöm több teszteset használatával.

A `Model` osztály tesztelésénél validálom a modell felépítésének és súlyainak helyes betöltését valamint azt, hogy adott bemenetekre ad-e becsléseket. A becslések eredményeinek címkévé alakításával tesztelem a `OneHotDecorder` funkcióit.

4.4 Adatbázisok

A tanító adatbázis felépítése előtt minden rögzített adatfájlt átvizsgáltam a 3.1.3-as fejezetben ismertetett alkalmazással. Ezután a rögzített nyers adatokat a Pandas [48] Python csomag használatával dolgoztam fel és az adatokat egy fájlban tároltam.

Az adatok előfeldolgozásakor egy kisebb akadályba ütköztem, ami az Android rendszer szenzorkezeléséből fakad. Bizonyos adatforrások nem egyenlő időközönként jelzik az eseményeket, amikre az alkalmazások feliratkozhatnak, hanem csak a mért értékek megváltozása esetén. A gyorsulásmérő és az orientációs szenzor képes volt az 50Hz-es mintavételi frekvencia kiszolgálására, viszont például a fényérzékelő, lokációs szenzor és a WiFi kapcsolat adatai bizonyos esetekben sokkal ritkábban változnak, emiatt a rögzített adatfájlokban sok hiányzó adatpont szerepelt. Ezt a problémát úgy oldottam

meg, hogy a hiányzó adatokat mindig a legutóbbi nem-hiányzó adatponttal pótoltam. Ez a gyakorlatban azt jelenti, hogy ha például a telefont zsebre teszem, akkor 0-t (sötét) jelez vissza a fényérzékelő. Ez az adat viszont a következő zsebből kivételig nem változik, itt hiányzó értékek jelennek meg az adatokban. Ezeket helyettesítem a legelső 0-t (sötétet) jelző adatponttal mindaddig, amíg ez meg nem változik (kivettem a telefont a zsebemből, újra világos van).

A rögzített és feldolgozott adatok mennyisége és kategória szerinti eloszlása a 4.1. táblázatban látható. Legtöbb adat az autózás, sétálás és biciklizés tevékenységekből van, legkevesebb pedig a lépcsőzésből.

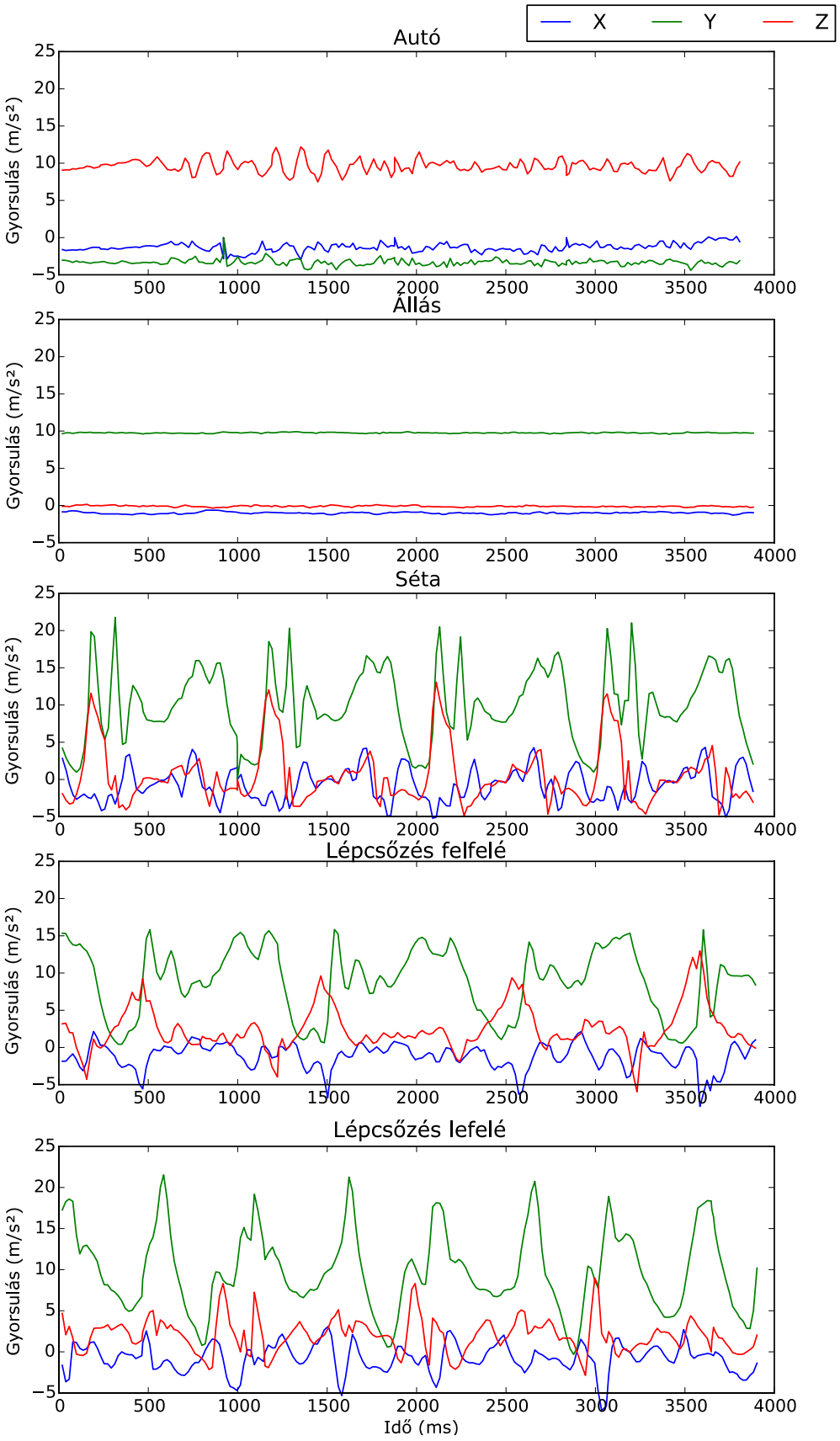
4.1. táblázat - Rögzített tevékenységadatok mennyisége kategóriánként

	Bicikli	Autó	Futás	Lépcsőzés	Állás	Séta	Összesen
Idő (óra)	9,28	30	7,15	0,25	1,55	14,75	62,98

Összesen ~63 órányi adatot gyűjtöttem 11 embertől. A feldolgozott adatok 2 GB tárterületet foglalnak.

A neurális hálózatok tanításához a szöveges tanítócímkéket átalakítottam one-hot encoding használatával. A one-hot encoding minden tanítócímkéhez egy olyan vektort rendel aminek annyi eleme van ahány osztálycímke. A vektor egyetlen eleme „1” értékű, ez jelzi, hogy melyik osztályba tartozik az adott minta, a többi elem pedig „0”. Az ilyen módon előállított vektorok visszaalakíthatók szöveges formára.

A 4.6. ábrán az autózás, egy helyben állás, sétálás, lefelé lépcsőzés és felfelé lépcsőzés gyorsulásadatainak ábrázolása látható. A függőleges tengelyen a gyorsulás, a vízszintes tengelyen pedig az idő (ms). van megjelenítve.



4.6. ábra - Autó, állás, sétálás, lépcsőzés fel, lépcsőzés le tevékenységek gyorsulásértékeinek összehasonlítása. A vízszintes tengelyen az idő van megjelenítve, helyszűke miatt ezt csak az alsó ábrán tüntetem fel.

4.5 Mély neurális hálózatok gyakorlati megvalósítása

A rendszert ellenőrzött tanulási módszerrel tanítottam. A tanítás célja, hogy a tanítandó rendszer paramétereit olyan módon válasszuk meg vagy alakítsuk ki, hogy a betanított rendszer képes legyen a tanulás során látott mintapontok alapján új, eddig nem látott mintákra is megfelelő választ adni.

A létrehozott tanítóadatbázist felosztottam tanító-, teszt-, és validációs adatsorokra. A tanítás alatt a tanítóadatok segítségével tanulja meg a rendszer a kívánt leképezés becslését. A validációs adathalmazt a modell tanítóepochok között történő kiértékeléséhez használtam. A tanítás lefutása után pedig a tesztadathalmazon mértem le a modell pontosságát a következő metrikák használatával: pontosság (precision), felidézés (recall), F1.

A tanító-, teszt-, és validációs adatsorokat rendre 70%, 20% és 10% arányban osztottam fel. A tanítások során megvizsgáltam azt az esetet is, hogy egy kiválasztott ember adatsorát eltávolítottam a tanítóhalmazból és utána az ő adatsoraival teszteltem a betanított modellt.

A tanítások futtatásához a BME TMIT Smartlab laborcsoportjának egyik Ubuntu 16.04 operációs rendszert futtató Intel Core i7-4790 @ 3.60GHz CPU-t , 32 GB RAM-ot és egy nVidia GTX 1080 Ti GPU-t tartalmazó szerverét használtam.

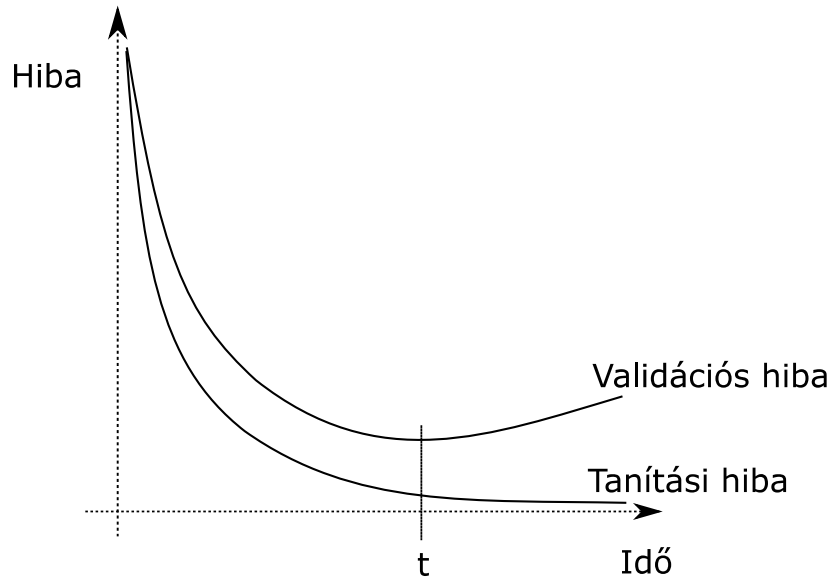
4.5.1 Tanítások során használt technikák

A gépi tanulás során a tanuló rendszer a tanulás folyamata közben látott mintapontokból nyert ismeretekből javítja a teljesítőképességét (nagyobb pontosság, kisebb hiba, stb.). A minél optimálisabb tanítások futtatásához több technika alkalmazható [14] [49]. A pontos modellek készítésének egyik fontos szempontja, hogy elkerüljük a túltanulást. Túltanulásról akkor beszélünk, ha a modell válaszai túlzottan illeszkednek a tanítóminták által megszabott leképezéshez, vagyis a modell általánosító képessége romlik.

4.5.1.1 Early Stopping

A neurális hálózatok tanításakor általában egy előre meghatározott számú tanítóepochon keresztül tanítják a hálózatot. Az Early Stopping módszer lényege, hogy amennyiben a tanítás során bizonyos számú tanítóepoch után nem javul tovább a validációs hiba akkor automatikusan leáll a tanítás. Amíg a hiba mértéke a tanító

adatsoron és a validációs adatsoron visszamérve egyaránt csökken, addig nagy valószínűséggel nem lép fel a túltanulás jelensége. Viszont amikor a tanító hiba tovább csökken, a validációs hiba viszont nem (esetleg nől) akkor célszerű leállítani a tanítást (4.7. ábra). A tanítás leállítására az optimális időpontot a 4.7. ábrán t-vel jelöltem. Ennek az időpontnak a megtalálása nem minden esetben egyértelmű, de az optimálishoz a lehető leközelibb leállítás esetén is sokat javulhat a modell pontossága. A módszer további előnye, hogy a tanítás megfelelő időpontban történő leállításával időt spórolhatunk.



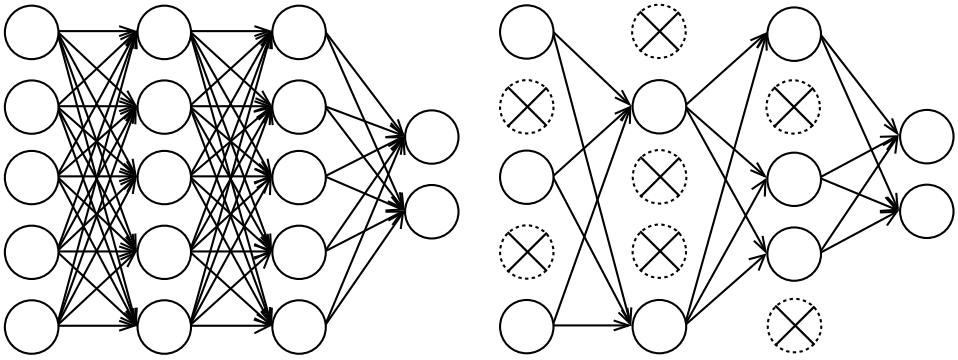
4.7. ábra - Tanító és validációs adatokon visszamért hiba mértékének összehasonlítása a tanítás során

A Keras-ban az Early Stopping módszerrel párhuzamosan célszerű a modell súlyait minden olyan tanítóepoch után elmenteni (Model Checkpoint) amikor javult a validációs hiba. A módszerrel alapértelmezés szerint figyelhetjük a modell pontosságát (osztályozási feladat esetén) vagy a hiba mértékét, de egyéb visszamérési metrikák használata is implementálható. A tesztelés során ezek a súlyok visszatölthetők, így biztosítva hogy a lehető legoptimálisabb modellt használjuk.

4.5.1.2 Dropout

A dropout egy olyan módszer, amivel a neurális hálózatok pontosságát lehet javítani a túltanulás mértékének csökkentésével [50]. A módszer lényege, hogy a neurális hálózat tanítása során minden egyes tanítóepoch végrehajtása után a belső rétegek között

a neuronok közti összeköttetések egy részét (tipikusan 20-50%) véletlenszerűen eldobjuk (4.8. ábra).



4.8. ábra – Dropout módszer; bal oldal: összeköttetések eldobása előtt, jobb oldal: összeköttetések eldobása után (forrás: [50] alapján)

Ennek hatása, hogy az azonos rétegbeli neuronok kevésbé tudnak egymás segítségére hagyatkozni, önállósodásra vannak kényszerítve. A módszer nagy előnye, hogy egyszerűen implementálható és együtt tud működni a hiba-visszaterjesztéses alapú tanítási módszerekkel.

4.5.1.3 Minibatch tanulás

A tanítóepochok során az adatokat többféleképpen lehet a hálózat súlyainak módosítására felhasználni. Az egyik ilyen eset során a teljes tanítóadatbázist használhatjuk fel [49]. Ez az eset sokszor lassú, nehéz a folyamat párhuzamosítása. Egy másik eset lehet az, ha minden mintát egyesével használunk fel. A harmadik megközelítés az úgynevezett mini-batch tanítás, itt a tanítópontokat véletlenszerűen felosztják bizonyos mintaszámú részekre és ezekbe a kisebb részekbe (batch) tartozó mintákat használják fel egyidejűleg a hálózat tanítására úgy, hogy minden tanítóepochban minden minta felhasználásra kerül. A tanítások során a minibatch módszert használtam.

4.5.1.4 Standardizálás

A neurális hálózatok az elméleti háttér alapján képesek közelíteni a nagy értékkészletű függvényeket is, de az alkalmazott nemlineáris aktivációs függvény értékkészletének korlátai miatt célszerű standardizálni a bemeneti adatokat. Standardizált adatok használatával a tanítások során a hiba általában gyorsabban csökken, ha a

bemeneti változók átlaga nullához közeli. A folyamat során a bemeneti változók értékkészletét célszerű úgy skálázni, hogy az átlaguk 0, a szórásuk pedig 1 legyen.

4.5.1.5 Batch normalizálás réteg

A batch normalizálás réteg segítségével a hálózat bizonyos pontjain standardizálhatjuk az adatokat. A bemeneti értékek standardizálásával már 0 átlagúra és 1 szórásúra alakítottuk az adatokat, viszont ahogy ezek az értékek a hálózaton végighaladnak, változásokon esnek keresztül, ami kedvezőtlenül befolyásolhatja őket a tanítás szempontjából. A batch normalizálás réteggel minibatchenként standardizálhatjuk az előző réteg kimeneteit [51].

4.5.1.6 Tanítóminták összekeverése

A neurális hálózatok leggyorsabban a váratlan mintákból tanulnak, ezért célszerű minden tanítási iteráció során a hálózat számára újdonságtartalmat hordozó mintákkal tanítani. Az újdonságtartalom vizsgálata nem egyszerű feladat, viszont ha az egymás után következő tanítóminták más-más osztályba tartoznak, akkor a minták nagy valószínűséggel különböző információkat fognak tartalmazni.

4.5.1.7 Súlyok kezdeti értéke

A hálózat kezdeti súlyainak értéke jelentősen befolyásolhatja a tanulási folyamatot, mert a kezdeti hiba mértékét az indulási súlyvektor határozza meg [14]. A súlyokat véletlenszerűen választjuk, de úgy, hogy a nemlineáris aktivációs függvény és a bemeneti változók értékkészletei összemérhetők legyenek. A véletlen kezdeti értékek a szimmetriák elkerülését biztosíthatják, vagyis azt, hogy az egyes neuronok különböző leképezéseket valósítsanak meg.

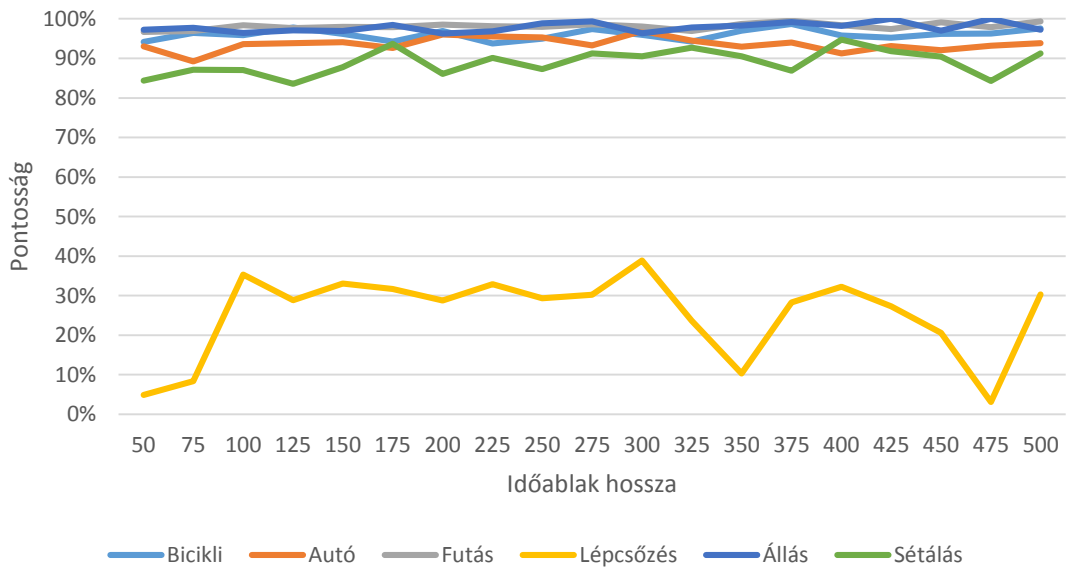
A munka során elkészített Python kódok 2818 kódsorból állnak és 98 kB tárhelyet foglalnak.

5 Eredmények

A tevékenységadatok elemzése során kiderült, hogy némely hasonló tevékenységeket nehezen tud megkülönböztetni a rendszer (például lépcsőzés felfelé, lefelé, hasonlóan az [1] irodalomban látottakhoz), viszont az ilyen és ehhez hasonló tevékenységek címkéinek összevonásával javítani tudtam a felismerési pontosságon.

A szenzoradatokat is megvizsgáltam, olyan szempontból, hogy melyek lehetnek azon adatforrások, amik rontanak a felismerési pontosságon. Ehhez írtam egy olyan Python scriptet, ami minden iterációban „kihagy" egy adatforrást a tanítóvektorból, végigfuttatja a tanítást és kiszámolja a pontosságot. Ezt minden adatforrásra elvégzi, majd kiválasztja a legjobb pontosságot elérő adathalmazt, és újrakezdi a ciklust – viszont már egy adatforrással kisebb halmazon.

A tanítások során olyan méréseket is végeztem ahol azt elemeztem, hogy az egy időablakba összefogott adatok mennyiségének változása milyen hatással van a felismerés pontosságára. Egy ilyen mérés eredményét ábrázoltam az 5.1. ábrán. A mérések alapján látható, hogy az időablak méretének a tevékenységek nagy részénél nem volt számottevő hatása a felismerés eredményessége szempontjából. Jelentős különbségeket a lépcsőzés kategóriánál tapasztaltam, ennek a tevékenységnek a felismerési pontossága elmarad a többi vizsgált osztályétól.

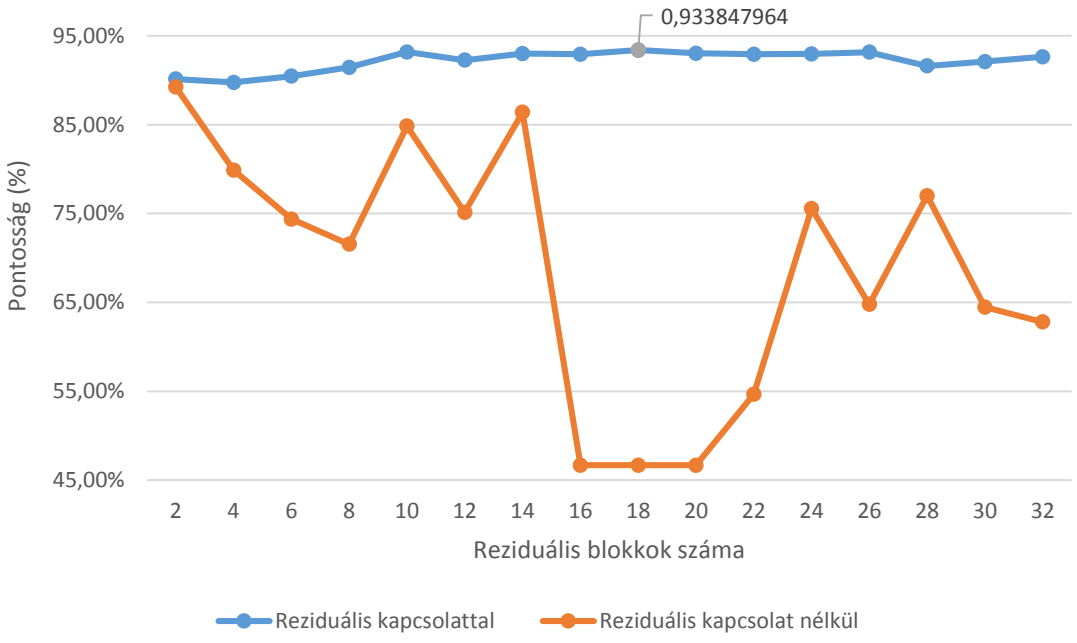


5.1. ábra - Tevékenységkategóriák felismerési pontosságának változása az vizsgált időablak hosszának függvényében

A tanítások futtatásakor azt is szem előtt tartottam, hogy ha egy kategóriából lényegesen több adat van, mint a többiből (autózás), akkor abból annyi adatot célszerű elhagyni, hogy a többi kategóriát is figyelembe véve kiegyenlített mintaszám legyen az egyes osztályokban.

A feladat megoldása során a megvizsgált mély neurális hálózatarchitektúrák közül legjobb pontosságúak a reziduális hálózatok voltak. A reziduális hálózatok hiperparamétereinek tanulmányozása során megvizsgáltam a kezdeti konvolúciós szűrők méretének, a konvolúciós ablak hosszának, a reziduális blokkok számának és a reziduális kapcsolatok meglétének hatását a rendszer pontosságára. E két utóbbi vizsgálat eredményeit részletesebben is ismertetem.

A már említett két hiperparaméter elemzésekor a reziduális blokkok számát 2 és 32 között változtattam, egyszer a reziduális kapcsolat megtartásával, egyszer pedig nélküle. Az így kapott hálózatokat tanítottam, a többi hiperparaméter a vizsgálatok során változatlan volt. A tanítások eredményei az 5.2. ábrán láthatók.



5.2. ábra - Reziduális kapcsolat hatása a blokkok számának változtatása mellett

Megfigyelhető, hogy a reziduális kapcsolatokat is tartalmazó hálózatok minden kipróbált reziduális blokkméret esetén jobb pontosságot értek el a reziduális kapcsolat nélküli párjukhoz képest. A legjobb eredményt 18 reziduális blokk használatával kaptam itt 93,38% volt a pontosság.

A dolgoztat elkészítése alatt futtatott és dokumentált tanítások összesen 350 különböző összeállításból állnak (LSTM, konvolúciós és reziduális architektúrák). A tanítások és azok kiértékelései mintegy 500 órán keresztül futottak.

5.1 A kísérleti rendszer eredményeinek ismertetése

Az elkészített kísérleti rendszer eredményeinek ismertetéséhez kiválasztottam a mérések során legjobb hatékonysággal működő összeállítást. A legjobb összeállításban a reziduális hálózat 200 egymás utáni mintát vizsgál (50 Hz mintavételi frekvencia mellett ez 4 másodperc) úgy, hogy az egymást követő mintáknál 170 mintányi átfedés van. A reziduális hálózatban 18 reziduális blokk van, ami 36 konvolúciós réteget jelent. A konvolúciós rétegek kezdeti szűrőszáma 16, a konvolúciós ablakméret pedig 4 hosszú a teljes hálózatban. Az összeállításban összesen 414614 db tanítható paraméter található. A tanítást 500 tanítóepochal végeztem, a batch méret 512 minta volt. Hibaszámító függvénynek a „categorical crossentropy”-t használtam, a betanított kategóriák: autó, sétálás, biciklizés, futás, lépcsőzés és az egy helyben állás.

A tanítás eredményességét az 5.1. táblázatban látható konfúziós mátrixban ábrázolom, a tevékenységek felismerésének átlagos pontossága 93,4% volt (súlyozott átlag). A legrosszabb felismerési pontosságú tevékenység a lépcsőzés. Ennek az oka az lehet, hogy a tevékenység nagyon hasonlít a sétálásra, sok lépcsőzés kategóriába tartozó tesztpontot sorol a modell a sétálás alá, valamint ebből a tevékenységből van a legkevesebb adat.

5.1. táblázat - Legjobb összeállítás tesztelésének eredménye konfúziós mátrixban

	Becsült						
Valós	*Bicikli*	*Autó*	*Futás*	*Lépcsőzés*	*Állás*	*Sétálás*	Összesen
Bicikli	**10972(90%)**	201	0	2	42	516	11733
Autó	423	**33764(98%)**	4	0	98	1807	36096
Futás	35	27	**8861(99%)**	0	6	112	9041
Lépcsőzés	4	21	1	**40 (60%)**	0	254	320
Állás	37	127	4	0	**1672(82%)**	50	1890
Sétálás	702	383	5	25	230	**16912(86%)**	18257
Összesen	12173	34523	8875	67	2048	19651	77337

A konfúziós mátrix oszlopai a valós, sorai pedig a becsült osztályt jelölik. A mátrix főátlójában szereplő értékek a helyesen osztályozott minták számát jelölik.

5.2 Demonstrációs mintaalkalmazás

A legjobb modell okostelefonos környezetben való tesztelésének megkönnyítéséhez implementáltam egy olyan nézetet ahol napi bontásban egy kördiagramon [52] lehet megjeleníteni az adott napon felismert tevékenységek arányát (5.3. ábra). A nézet a tevékenységfelismerési statisztikákhoz használt SQL adatbázis adatait jeleníti meg.

5.3. ábra - Felismerési statisztikák megjelenítése

A felismerési szolgáltatás indításának és leállításának megkönnyítéséhez az erre szolgáló gombokat is elhelyeztem és ezt a nézetet állítottam be az alkalmazás kezdőképernyőjének. Az alkalmazás letölthető[2] a Google Play áruházból.

[2] https://play.google.com/apps/testing/hu.ca0phr.sensorrecorder

6 Összefoglalás

Munkám során az irodalomkutatás elvégzése után tanulmányoztam a neurális hálózatok elméleti hátterét és többféle mintamérést elvégeztem, amivel gyakorlati tapasztalatokat szereztem a neurális hálózatok tanításával és képességeivel kapcsolatban. A dolgozat bevezetőjében kitűzött célokat a következők szerint valósítottam meg:

1. Módosítottam a korábban elkészített adatrögzítő alkalmazást és kidolgoztam az adatrögzítésekhez szükséges módszertant.
2. A módszertant követve 11 ember segítségével címkézett adatrögzítéseket végeztem és az így kapott adatokkal új referencia adatbázist hoztam létre.
3. A korábbi tevékenységfelismerést végző rendszerekhez képest új megközelítést alkalmaztam az adatok feldolgozása során, a mély neurális hálózatok használatával. A rendelkezésemre álló adatok segítségével modelleket építettem, amelyek közül a legjobb összeállítás 93,4% pontossággal meg tudta határozni a felhasználó által végzett tevékenységet. Eredményeimet részletesen dokumentáltam és objektív módszerekkel értékeltem ki.
4. Céljaim szerint TCP-IP alapú kliens-szerver architektúrát alakítottam ki az adatok rögzítéséhez valamint az elkészült modellek online futtatásához.
5. A kliens-szerver architektúrán felül lehetővé tettem a betanított modellek okostelefonon való offline futtatását is.
6. A tanítások során kapott legjobb modellt felhasználva implementáltam és publikáltam egy demonstrációs mintaalkalmazást, ami lehetővé teszi a modellek valós életbeli tesztelését.

Az irodalomkutatás során általam feltárt korábbi munkákban jobb eredményt nem találtam, de hasonló feltételek mellett végzett vizsgálatokat sem ismerek, ezért nem láttam értelmét az összehasonlításnak (más tevékenységek, más algoritmusok, más tanítóadatok, más adatforrások stb.). A [13] irodalomban még leginkább hasonló feltételek mellett, viszont csak három tevékenységkategóriával (sétálás, futás és mozdulatlanság) 92,71% átlagos pontosságot értek el.

A munkám során kialakított tevékenységfelismerő rendszer felhasználható például az ember-gép interakciók kontextusfüggő segítésére. A jelenleg elérhető módszerekhez képest előny lehet az egyedi tevékenységek betanításának lehetősége is. A szakirodalomban tevékenységfelismerésen túl is nagy lehetőségeket látnak a szenzoros adatok mély neurális hálózatokkal történő elemzésében, például az orvosi területen való felhasználására is, viszont a további felhasználási lehetőségek pontos vizsgálatához az adott területen történő átfogó adatrögzítések és további kutatások szükségesek.

Irodalomjegyzék

[1] N. Ravi, N. Dandekar, P. Mysore és M. L. Littman, „Activity Recognition from Accelerometer Data," *AAAI Vol. 5,* pp. 1541-1546, 2005.

[2] G. M. Weiss, J. W. Lockhart, T. T. Pulickal, P. T. McHugh, I. H. Ronan és J. L. Timko, „Actitracker: a smartphone-based activity recognition system for improving health and well-being," *SIGKDD Exploration Newsletter,* p. 9, 2014.

[3] S. Arora, V. Venkataraman, S. Donohue, K. M. Biglan, E. R. Dorsey és M. A. Little, „High accuracy discrimination of Parkinson's disease participants from healthy controls using smartphones," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 3641-3664, 2014.

[4] T.-V. How, J. Chee, E. Wan és A. Mihailidis, „Mywalk: a mobile app for gait asymmetry rehabilitation in the community," *7th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth),* pp. 73-76, 2013.

[5] Y. Bengio, „Learning deep architectures for AI.," *Foundations and trends in Machine Learning,* pp. 1-47, 2009.

[6] A. Graves, A.-r. Mohamed és G. Hinton, „Speech Recognition with Deep Recurrent Neural Networks," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 6645-6649, 2013.

[7] Y. LeCun és Y. Bengio, „Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks,* p. 10, 1995.

[8] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior és K. Kavukcuoglu, „Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499,* p. 15, 2016.

[9] P. Rajpurkar, A. Hannun, M. Haghpanahi, C. Bourn és A. Y. Ng, „Cardiologist-level arrhythmia detection with convolutional neural networks," *arXiv preprint arXiv:1707.01836,* p. 9, 2017.

[10] U. Maurer, A. Smailagic, D. P. Siewiorek és M. Deisher, „Activity recognition and monitoring using multiple sensors on different body positions," *IEEE proceedings on the International Workshop on Wearable and Implantable Body Sensor Networks,* p. 4, 2006.

[11] A. Rasekh, C.-A. Chen és Y. Lu, „Human activity recognition using smartphone," *arXiv preprint arXiv:1401.8212,* p. 7, 2014.

[12] D. Anguita, A. Ghio, L. Oneto, X. Parra és J. L. Reyes-Ortiz, „Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," *Ambient assisted living and home care,* pp. 216-223, 2012.

[13] S.-M. Lee, S. M. Yoon és H. Cho, „Human activity recognition from accelerometer data using Convolutional Neural Network," *IEEE International Conference on Big Data and Smart Computing (BigComp),* pp. 131-134, 2017.

[14] M. Altrichter, G. Horváth, B. Pataki, G. Strausz, G. Takács és J. Valyon, „Neurális hálózatok," Panem, Budapest, 2006.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard és L. D. Jackel, „Backpropagation applied to handwritten zip code recognition," *Neural computation,* pp. 541-551, 1989.

[16] D. Scherer, A. Müller és S. Behnke, „Evaluation of pooling operations in convolutional architectures for object recognition," *Artificial Neural Networks,* pp. 92-101, 2010.

[17] C. Olah, „Conv Nets: A Modular Perspective," 2014. [Online]. Elérhető: http://colah.github.io/posts/2014-07-Conv-Nets-Modular/. [Hozzáférés dátuma: 18 05 2016].

[18] X. Glorot és Y. Bengio, „Understanding the difficulty of training deep feedforward neural networks," *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics,* p. 8, 2010.

[19] K. He, X. Zhang, S. Ren és J. Sun, „Deep residual learning for image recognition," *Proceedings of the IEEE conference on computer vision and pattern recognition,* p. 2016, 770-778.

[20] S. Xie, R. Girshick, P. Dollár, Z. Tu és K. He, „Aggregated residual transformations for deep neural networks," *arXiv preprint arXiv:1611.05431,* p. 10, 2016.

[21] C. Szegedy, S. Ioffe, V. Vanhoucke és A. A. Alemi, „Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *AAAI,* pp. 4278-4284, 2017.

[22] S. Zagoruyko és N. Komodakis, „Wide residual networks," *arXiv preprint arXiv:1605.07146,* p. 15, 2017.

[23] B. Czeba, „Statisztikai modellekkel segített ember-gép interakció mobil eszközökön," 2014, p. 56.

[24] L. Deng, G. Hinton és B. Kingsbury, „New types of deep neural network learning for speech recognition and related applications: An overview," *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* pp. 8599-8603, 2013.

[25] E. B. Lenselink, N. Dijke, B. Bongers, G. Papadatos, H. W. Vlijmen, W. Kowalczyk, A. P. IJzerman és G. J. Westen, „Beyond the hype: deep neural networks outperform established methods using a ChEMBL bioactivity benchmark set," *Journal of Cheminformatics,* p. 14, 2017.

[26] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger és P. Simard, „Comparison of classifier methods: a case study in handwritten digit recognition," *Proceedings of the 12th IAPR International conference on Pattern Recognition*, pp. 77-82, 1994.

[27] „Sensors Overview | Android Developers," [Online]. Elérhető: https://developer.android.com/guide/topics/sensors/sensors_overview.html. [Hozzáférés dátuma: 18 05 2016].

[28] „Android Studo," [Online]. Elérhető: https://developer.android.com/studio/index.html. [Hozzáférés dátuma: 19 05 2016].

[29] R. Lugg, S. Ferg, A. Zawadzki és H. Jens, „easygui for Python," [Online]. Elérhető: https://github.com/robertlugg/easygui. [Hozzáférés dátuma: 12 05 2016].

[30] A. S. Tanenbaum és D. J. Wetherall, Számítógép-hálózatok, Panem Könyvek, 2013.

[31] T. Bray, *The javascript object notation (json) data interchange format*, RFC 7159, 2014.

[32] P. Deutsch és J.-L. Gailly, *ZLIB Compressed Data Format Specification version 3.3*, RFC 1950, 1996.

[33] „Neurolab 0.3.5 documentation," [Online]. Elérhető: https://pythonhosted.org/neurolab/. [Hozzáférés dátuma: 21 05 2016].

[34] I. J. Goodfellow, D. Warde-Farley, P. Lamblin, V. Dumoulin, M. Mirza, R. Pascanu, J. Bergstra, F. Bastien és Y. Bengio, „Pylearn2: a machine learning research library," *arXiv preprint arXiv:1308.4214*, p. 9, 2013.

[35] F. Chollet, „Keras," GitHub, https://github.com/fchollet/keras, 2015.

[36] T. D. Team, „Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, 2016.

[37] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean és M. Devin, „Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, p. 19, 2016.

[38] C. Olah, „Understanding LSTM Networks," 2015. [Online]. Elérhető: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Hozzáférés dátuma: 18 05 2015].

[39] S. Hochreiter és J. Schmidhuber, „Long short-term memory," *Neural computation*, pp. 1735-1780, 1997.

[40] „AsyncTask | Android Developers,” [Online]. Elérhető: https://developer.android.com/reference/android/os/AsyncTask.html. [Hozzáférés dátuma: 7 10 2016].

[41] „Saving Data in SQL Databases | Android Developers,” [Online]. Elérhető: https://developer.android.com/training/basics/data-storage/databases.html. [Hozzáférés dátuma: 03 10 2017].

[42] „UI/Application Exerciser Monkey | Android Studio,” [Online]. Elérhető: https://developer.android.com/studio/test/monkey.html. [Hozzáférés dátuma: 14 10 2016].

[43] „Use pre-launch reports to identify issues,” [Online]. Elérhető: https://support.google.com/googleplay/android-developer/answer/7002270?hl=en. [Hozzáférés dátuma: 20 10 2017].

[44] „TensorFlow Mobile,” [Online]. Elérhető: https://www.tensorflow.org/mobile/. [Hozzáférés dátuma: 14 09 2017].

[45] J. K. Lee és J. Y. Lee, „Android programming techniques for improving performance,” *International Conference on Awareness Science and Technology (iCAST),* pp. 386-389, 2011.

[46] L. Chen, „transcranial/keras-js Run Keras models in the browser, with GPU support,” [Online]. Elérhető: https://github.com/transcranial/keras-js. [Hozzáférés dátuma: 14 05 2017].

[47] „Building Web Apps in WebView,” [Online]. Elérhető: https://developer.android.com/guide/webapps/webview.html. [Hozzáférés dátuma: 02. 08. 2017.].

[48] W. McKinney, „Data Structures for Statistical Computing in Python,” *Proceedings of the 9th Python in Science Conference,* pp. 51-56, 2010.

[49] Y. LeCun, L. Bottu, G. Orr és K.-R. Müller, „Efficient backprop,” *Neural networks: Tricks of the trade,* pp. 9-48, 2012.

[50] G. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever és R. Salakhutdinov, „Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *The Journal of Machine Learning Research,* pp. 1929-1958, 2014.

[51] S. Ioffe és C. Szegedy, „Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *International Conference on Machine Learning,* pp. 448-456, 2015.

[52] P. Jahoda, „MPAndroidChart: A powerful Android chart view / graph view library, supporting line- bar- pie- radar- bubble- and candlestick charts as well as scaling, dragging and animations.,” [Online]. Elérhető: https://github.com/PhilJay/MPAndroidChart. [Hozzáférés dátuma: 22 10 2017].

Rövidítésjegyzék

ASR: Automatic Speech Recognition – Automatikus beszédfelismerés

TTS: Text-To-Speech – Gépi beszédszintézis

CPU: Central Processing Unit – Központi feldolgozóegység, Processzor

CSV: Comma-separated Values – Vesszővel elválasztott értékek

DNN: Deep Neural Network – Mély neurális hálózat

CNN: Convolutional Neural Network – Konvolúciós neurálos hálózat

RNN: Recurrent Neural Network – Visszacsatolt neurális hálózat

FFN: Feed Forward Network – Előrecsatolt hálózat

GPU: Graphics Processing Unit – Grafikai processzor

HMM: Hidden Markov Model – Rejtett Markov model

KNN: K-nearest neighbor – K legközelebbi szomszéd

LDA: Linear Discriminant Analysis – Lineáris diszkrimináns analízis

LSTM: Long Short-Term Memory – Hosszú rövidtávú memória

SVM: Support Vector Machine: Szupport vektor gép