SZAKDOLGOZAT

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Kulcsár Dániel Patrik**
Törzskönyvi száma: T/006276/FI12904/N
Neptun kódja: YKNJXY

A dolgozat címe:

Navigációs alkalmazás és neurális hálózat alapú táblafelismerő rendszer fejlesztése
Developing a navigation application and road sign recognition system based on neural network

Intézményi konzulens: Kovács András
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés specializáció

A feladat

Napjainkban az autóval közlekedők száma egyre nagyobb, ezzel együtt a közlekedéshez kapcsolódó mobilalkalmazások száma is növekszik. A piacon már elérhetőek közösségi navigációs alkalmazások (pl. Waze), azonban olyan még nem létezik, ami a jelzőtáblákat képes lenne jelezni a térképen. A hallgató feladata egy olyan rendszer megalkotása, ami szerveroldali és kliensoldali alkalmazásból áll. A kliensoldal az Android operációs rendszerre megírt navigációs applikáció legyen egy táblarögzítő üzemmóddal. Ennek az üzemmódnak a segítségével az alkalmazás legyen képes továbbítani a szerver felé a megfelelő koordinátákat és néhány képkockát másodpercenként, amit a telefon kamerája készít el. A szerveroldalon történjen meg a képek szegmentálása, majd neurális hálózat segítségével történjen meg az adott képkockán látható közlekedési jelzőtábla felismerése. A szerver a neurális hálózat válaszát (tábla típusa) és annak koordinátáját rögzítse az adatbázisban. A hallgató készítsen továbbá egy olyan webes felületet, ahol lehetőség van manuálisan is rögzíteni táblákat a térképen. A hallgató végezze el a szükséges teszteket az elkészült rendszeren és dokumentálja azt.

A dolgozatnak tartalmaznia kell:

- a szakirodalom részletes áttekintését és értékelését,
- a megoldandó feladat pontos leírását és részletes elemzését,
- a meglévő rendszerek elemzését,
- az alkalmazni kívánt technológiák elemzését, döntések indoklását,
- a rendszertervet,
- a megvalósítás pontos leírását,
- a tesztelési tervet, teszteredményeket, a fejlesztés során felmerült problémák elemzését,
- az elkészült rendszer felhasználási és továbbfejlesztési lehetőségeit.

ÓBUDAI EGYETEM Neumann János Informatikai Kar Ph

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

Budapest, 2022.05.14.

.....

hallgató aláírása

OE-NIK Hallgató neve: **Kulcsár Dániel Patrik**

2021 Hallgató törzskönyvi száma: **T/006276/FI12904/N**

Navigációs alkalmazás és neurális hálózat alapú táblafelismerő rendszer fejlesztése

Developing a navigation application and road sign recognition system based on neural network

Tartalom

1. BEVEZETÉS 10
2. A SZAKDOLGOZATBAN MEGVALÓSÍTANDÓ RENDSZER 12
2.1. A PROBLÉMA FELVETÉSE 12
2.2. A PROBLÉMA FONTOSSÁGA 12
3. NAVIGÁCIÓS RENDSZEREK A MINDENNAPOKBAN 14
3.1. GOOGLE MAPS 14
3.2. APPLE MAPS 15
3.3. WAZE 15
3.4. NAVIGÁCIÓS ALKALMAZÁSOK ÖSSZEGZÉSE 16
4. HASONLÓ RENDSZEREK 18
4.1. TÁBLAFELISMERŐ RENDSZEREK 18
4.1.1. Road-Sign Detection and Recognition Based on Support Vector Machines 20
4.1.2. Faster R-CNN for Small Traffic Sign Detection 23
4.1.3. A YOLO Based Approach for Traffic Sign Detection 25
4.1.4. HASONLÓ RENDSZEREK ÉRTÉKELÉSE 28
5. NEURÁLIS HÁLÓZATOK 30
5.1. KONVOLÚCIÓS NEURÁLIS HÁLÓZATOK 30
5.2. OBJEKTUM DETEKTÁLÁS ÉS KLASSZIFIKÁLÁS KONVOLÚCIÓS NEURÁLIS HÁLÓZAT SEGÍTSÉGÉVEL 31
5.2.1. R-CNN, FAST R-CNN, FASTER R-CNN 31
5.2.2. YOLO: YOU ONLY LOOK ONCE 32
6. A RENDSZER SPECIFIKÁCIÓJA 33
6.1. VÁZLATOS SPECIFIKÁCIÓ 34
7. RENDSZERTERV 36
7.1. ADATBÁZISTERV 37
7.2. A RENDSZER KOMMUNIKÁCIÓJA 38
8. IMPLEMENTÁLÁS 41
8.1. A SZERVEROLDAL MEGVALÓSÍTÁSA 41
8.1.1. TÁBLA HOZZÁADÁSA WEBEN KERESZTÜL 41
8.1.2. ÖSSZES TÁBLA LEKÉRDEZÉSE WEBEN KERESZTÜL 42
8.1.3. KÖZELI PONTOK LEKÉRDEZÉSE ANDROID KLIENSEN 43
8.1.4. KÉPFELTÖLTÉS ANDROID KLIENS HASZNÁLATÁVAL 43

8.1.5. FELTÖLTÖTT KÉP SZEGMENTÁLÁSA 43
8.1.6. A SZEGMENTÁLT KÉP KLASSZIFIKÁLÁSA 44
8.2. ANDROID OPERÁCIÓS RENDSZERREL RENDELKEZŐ KLIENSOLDAL MEGVALÓSÍTÁSA 48
8.2.1. NAVIGÁCIÓ MEGVALÓSÍTÁSA MAPBOX SDK SEGÍTSÉGÉVEL 48
8.2.2. KÉPKÜLDÉS MEGVALÓSÍTÁSA 50
8.3. WEBKLIENS MEGVALÓSÍTÁSA 51
9. TESZTELÉS 52
9.1. ANDROID KLIENS FUNKCIÓINAK TESZTELÉSE 52
9.3. A SZERVEROLDAL ÉS A NEURÁLIS HÁLÓZATOK TESZTELÉSE 55
11. EREDMÉNYEK ÉRTÉKELÉSE 61
11.2. SZERVEROLDAL EREDMÉNYEI 61
12. TOVÁBBFEJLESZTÉS LEHETŐSÉGEI 63
12.1. KLIENSOLDALAK 63
12.2. SZERVEROLDAL 63
13. ÖSSZEFOGLALÓ 64
14. SUMMARY 65
15. IRODALOMJEGYZÉK 66
16. ÁBRAJEGYZÉK 68

1. BEVEZETÉS

Az autóval való közlekedéshez hozzátartozik az, hogy az autó vezetőjének meg kell választania azt az útvonalat, amelyen el szeretne jutni az általa választott célállomásra. Abban az esetben, ha a vezető már ismeri az utat, akkor könnyű dolga van, hiszen külső segítség igénybevétele nélkül is el fog tudni jutni a kívánt helyre. Más esetekben, azonban szükség van valamilyen segítségre. A kétezres éveket megelőzően ez a segítség a papíralapú térkép volt.

Az autógyártók tudták, hogy a sofőröknek jóval kényelmesebb lenne valami olyan segédberendezés, ami akár vezetés közben is tudja mutatni az útvonalat, illetve azt, hogy hol, melyik irányba kell fordulni. Ilyen rendszereken már a 80-as évek első felében dolgoztak, végül a 90-es évek második felében terjedt el az a hozzáállás az autógyártóknál, hogy bizonyos modelljeikben opcionális extraként kérhető volt egyfajta navigációs rendszer. Az ezutáni lépcsőfok a hordozható GPS (Global Positioning System) volt, amit egyrészt azért hoztak létre, hogy a régebbi autókkal rendelkező emberek is tudjanak navigációs rendszer használni, illetve azért, mert a beépített GPS egy viszonylag drága extra volt akkoriban, így sokan nem kérték az autóikba. Napjainkban már a hordozható GPS is háttérbe szorult köszönhetően az okostelefonok elterjedésének. Az autóvezetők jelentős része már csak az okostelefonnal történő navigálást választja még akkor is, amikor nincs igazán rá szüksége, mert ismeri az adott útvonalat, azonban a mai navigációs alkalmazások már sokszor olyan plusz információkat (pl. forgalmi adatok, útépítések, útlezárások stb.) tudnak megosztani a vezetővel, ami minden esetben hasznos lehet.

Az első autóba beépített navigációs rendszert a Honda hozta létre 1981-ben. A később megjelent megoldásokkal ellentétben ez a rendszer még nem GPS műholdakat használt a pozícionálásra. A Honda által megalkotott rendszer egy héliumgáz giroszkóp segítségével jutott gyorsulási és sebesség adatokhoz, ami alapján el tudta helyezni az autót a térképen. [1]

Elsőként a General Motors által fejlesztett Oldsmobile 88-ban volt beépített GPS-alapú navigációs rendszer 1995-ben, majd ezek után jelent meg más nagy gyártónál is mint például a BMW-nél 2001-ben vagy az Audinál 2002-ben. [2]

A hordozható GPS-en több cég is dolgozott párhuzamosan, mivel hatalmas térnyerése lett ennek a technológiának a 2000-es évek elején. Ezen cégek közül az egyik legismertebb a TomTom. Ez a cég hozta létre a TomTom Go-t 2004-ben, ami igazából megteremtette a hordozható navigációs eszközök kategóriáját.

A hordozható GPS korszakát egy viszonylag rövid időszak követte. A gyártók úgy gondolták, hogy a nyomógombos telefonok is képesek lesznek már azt a szolgáltatást nyújtani a megfelelő alkalmazással, amit egy hordozható GPS, ezért például a Nokia megállapodást kötött a Navteq vállalattal, hogy készítsenek saját navigációs applikációt

a telefonjaikra, ez lett végül a Nokia Maps. Már a nyomógombos telefonokon is használható megoldások léteztek, azonban ez az ötlet az okostelefonoknál lett igazán kiforrott, hiszen ott már a hardver is megfelelő volt ahhoz, hogy egy akadásmentes, gyors felhasználói élményt nyújtson az alkalmazás a felhasználóinak. [4]

A következő nagy változás az okostelefonok elterjedésével történt meg, amikor ezeket az eszközöket kezdték el használni az emberek navigáció céljából. Ez a formája a navigációnak azért lett nagyon sikeres, mert nem volt szükség egy plusz eszközre ahhoz, hogy navigáljanak az emberek, illetve azért is, mert számos navigációs alkalmazás közül válogathatnak a felhasználók és ezek jelentős része teljesen ingyenesen használható. Igyekeztek a szoftvergyártók valami extrát beletenni saját alkalmazásaikba, például a Google a saját térképes applikációjába megvalósította azt, hogy ne csak az autósoknak legyen érdemes használni az alkalmazást, ezért ebben külön van lehetőség a gyalogos útvonalterv mellett tömegközlekedési útvonaltervet is készíteni.

Az autósok körében a napjaink egyik leggyakrabban használt navigációs alkalmazás a Waze. Rengeteg hasznos funkciója van az applikációnak köztük például az automata forgalom feltorlódás, illetve annak megszűnésének észlelése, a sebességkorlátok, útlezárások, elterelések jelzése. [5]

2. A SZAKDOLGOZATBAN MEGVALÓSÍTANDÓ RENDSZER

Ebben a fejezetben a szakdolgozatom alapjául szolgáló problémát és annak fontosságát fogom bemutatni. A következő két alfejezetben leírtakkal az a célom, hogy rávilágítsak az általam megvalósítani kívánt rendszer létjogosultságára.

2.1. A PROBLÉMA FELVETÉSE

Léteznek már autógyártóknak olyan modelljei, melyek rendelkeznek éjjellátó kamerával és abban az esetben, ha sötétben valaki átmegy a zebrán az autó előtt, legyen szó egy kivilágítatlan járműről vagy egy gyalogosról, akkor jelez a sofőrnek, illetve adott esetben automatikusan lefékez, ez csökkenti az éjszaka történő ilyen típusú balesetek számát. [6] A rendszer megbízhatóan működik, azonban egyrészt ez a megoldás nagyon drága, ezért a napjainkban közlekedő autók még csak elenyésző része rendelkezik ezzel a technológiával, illetve abban az esetben, ha az autó vezetőjének a figyelmét valami eltereli és nem veszi észre az átkelőhelyet jelző táblát, akkor ez a rendszer sem jelez, hogy átkelő következik. Emellett a rendszer mellett léteznek olyan megoldások is, amik kifejezetten a gyorshajtás megakadályozására jöttek létre. A rendszer figyeli a sebességkorlátozó táblákat az út szélén és beállítástól függően jelzi a sofőrnek, ha túllépte azt vagy adott esetben vissza is lassítja az autót arra a sebességre, ami a megengedett az adott útszakaszon.

Az olyan balesetek megelőzésére, amelyek azért történnek meg, mert az autót vezető személy nem figyelt vagy nem vett észre egy jelzőtáblát, megoldás lehetne egy olyan navigációs alkalmazás, amely képes lenne jelezni az adott útvonalon elhelyezett táblákat, így abban az esetben, ha a vezető éppen másra figyel, akkor is időben értesülne arról, hogy a következő sarkon például egy stop tábla lesz, ezért időben kezdje meg a lassítást.

Az alapötlet negatív oldala, hogy a táblákat kézzel kellene rögzíteni a térképen, azonban a neurális hálók adta lehetőségeket kihasználva a manuális rögzítés mellett lehetőség lehetne automatikus táblarögzítésre is. Az automatikus rögzítéshez a felhasználónak csak annyi dolga lenne, hogy úgy helyezze el a telefont az autója szélvédője előtt, hogy annak kamerája lássa az utat és az út szélén álló táblákat.

2.2. A PROBLÉMA FONTOSSÁGA

A forgalomban való részvétel hatalmas felelősséggel jár, amivel minden autóvezető feltételezhetően tisztában van, azonban sajnos egy apró figyelmetlenség is hatalmas bajt okozhat az utakon. A Központi Statisztikai Hivatal (továbbiakban: KSH) közúti közlekedési balesetekkel foglalkozó statisztikájából kiderül, hogy 2019-ről 2020-ra a balesetek száma csökkenő tendenciát mutat, azonban a 2020-as számok így is magasak. A Közlekedéstudományi Intézet (továbbiakban: KTI) ezzel a témával foglalkozó cikkéből kiderül, hogy nagyon sok gyalogos és kerékpáros szenvedett balesetet a 2011-2018 közötti időszakban. A statisztikából feltételezhető, hogy függetlenül attól, hogy a másik fél egy biciklis, egy gyalogos vagy egy másik autóvezető, a balesetek egy része

elkerülhető, megelőzhető lett volna, ha a vétkes fél körültekintőbben közlekedik és nagyobb figyelmet fordít az út mellett elhelyezett közlekedési táblákra. [7][8]

Szakdolgozatomban ezért egy olyan rendszert szeretnék létrehozni, ami az alapvető navigációs funkciók mellett képes jelezni a felhasználója felé az útvonalon megjelenő jelzőtáblákat.

A rendszer, amit létre fogok hozni azért lehetne jó megoldás erre a problémára, mert az okostelefonok már annyira elterjedtek napjainkban, hogy nem kéne egyik felhasználónak sem plusz beruházást tennie azért, hogy kihasználhassa az alkalmazás adta lehetőségeket és így nagyobb esély lenne a figyelmetlenségek miatt megtörtént balesetek megelőzésére, csökkentésére.

3. NAVIGÁCIÓS RENDSZEREK A MINDENNAPOKBAN

Az okostelefon az egyik legsűrűbben használt eszköz navigáció céljára. Az alkalmazásboltokban rengeteg alternatíva közül válogathatnak a felhasználók térképalkalmazásokból. A három legnépszerűbb ilyen típusú alkalmazás a Google Maps, az Apple Maps és a Waze. Ebben a fejezetben az előbb említett alkalmazásokról fogom ismertetni a főbb tudnivalókat.

3.1. GOOGLE MAPS

A Google saját térképszolgáltatását 2005 februárjában indította el Google Maps néven. A szoftver megjelenéskor még csak az útvonaltervezést és egyéb alapfunkciókat tartalmazott. 2007-ben került bele az élő forgalomfigyelés, ami a mai napig az egyik leghasznosabb funkciója a szolgáltatásnak. A Google Maps használata 2007-től kezdett növekedni igazán, ugyanis ekkor jelent meg az első mobilra készült verzió az akkori Blackberry telefonokra, majd 2009-ben Android 2.0-át futtatott eszközökre. Az alkalmazás 2009-től vált valós alternatívává az autós navigációk között ugyanis itt kapott egy nagyszabású átdolgozást az egész applikáció, ekkor már nem csak mutatta, hanem mondani is képes volt, hogy melyik lehetőségnél kell lefordulni, illetve figyelembe vette útvonaltervezésnél az aktuális forgalmi információkat. 2015-ben már offline módon is tudott navigálni az alkalmazás abban az esetben, ha az adott térképrészlet le volt mentve a telefonunkra. [9-13]

A Google Maps esetében lehet érezni, hogy nagy hangsúlyt fektettek arra, hogy az autósok számára is ez legyen az elsőszámú navigáció, hiszen például beépítették a sávjelzést is az alkalmazásukba, ami jelzi a sávokat, illetve azt, hogy a megjelenített sávok közül melyikbe kell majd haladnia a felhasználónak vagy a parkoló autó funkció, ami megmutatja, hogy hol parkolt le a felhasználó az autójával. Ezek a funkciók kifejezetten az autósoknak készültek. Az autósok mellett, azonban ez az alkalmazás a gyalogosoknak és a bicikliseknek is szól, ugyanis gyalogos útvonaltervezés esetén a tömegközlekedési eszközök menetrendjével számítja ki a várható utazási időt, biciklis útvonaltervezés esetén pedig az alkalmazás törekszik arra, hogy a kijelölt bicikliutakon keresztül tervezze meg az utat a felhasználónak.

Az alkalmazás, tehát nagyobb körnek készült, nem csak az autósokat akarja kiszolgálni. Képes felhasználni valós idejű adatokat a navigálás során, de sok esetben nem veszi figyelembe a forgalmi viszonyokat. A Google Maps a navigáláson kívül a különböző éttermláncok, boltok oldalait is képes megjeleníteni (rajtuk a nyitvatartási órákkal, véleményekkel stb.), ezzel elősegítve a felhasználó döntését azzal kapcsolatban, hogy milyen úticélt válasszon.

3.2. APPLE MAPS

Az Apple 2012 őszén mutatta be saját navigációs alkalmazását, az Apple Mapset. Nem titkolt céljuk az volt, hogy azok a felhasználók, akik az általuk gyártott eszközökkel rendelkeztek ne a konkurencia térképszolgáltatását használják.

Az Apple ugyanazt szerette volna elérni, amit a Google, hogy mindenkinek szóljon az alkalmazás, legyen szó autósról, gyalogosról, biciklisről. Az alkalmazás első éveiben minden téren le volt maradva a konkurenciához képest.

Az útvonaltervezés funkció használatakor többnyire hosszabb útvonalakat tervezett az Apple alkalmazása szemben a Google szoftverével. Az Apple Maps első éveiben csak gyalogos, illetve autós navigálásra lehetett használni, ugyanis a tömegközlekedési eszközöket is számításba vevő útvonaltervezés csak 2015 után került be az alkalmazásba. Az alkalmazás a különböző helyek oldalait és értékeléseit ugyanúgy képes megjeleníteni, mint a Google-s megfelelője, tehát ebben nincs eltérés, azonban az adatok származásában van. A Google Maps-en megjelenő adatok, legyen az értékelés, képek az adott helyről, ezek mind saját adatok, hiszen van lehetőség az alkalmazáson belül arra, hogy a felhasználó leírja véleményét az adott helyről, akár fotókkal illusztrálva. Az Apple esetében ez nincs így, ezért ők a TripAdvisor-on található adatokat jeleníti meg, ezért fordulhat elő az is, hogy a két térképalkalmazásban ugyanannak a helynek az értékelése eltérő.

Az Apple Mapsről is elmondható az, hogy jóval nagyobb körnek készült, mint csak az autós felhasználók, illetve az is, hogy sok olyan funkció van benne, ami hasznos, azonban egy csak autós felhasználásra túl sok.

3.3. WAZE

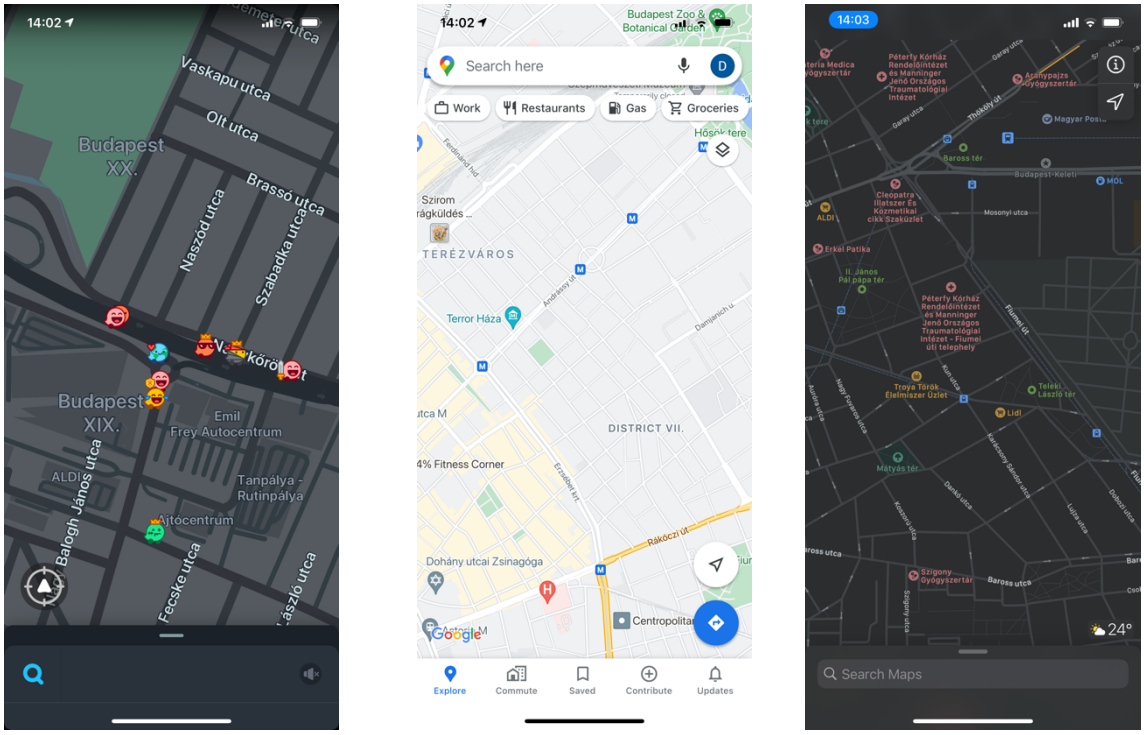
A Waze mint cég 2006-ban jött létre, abból a megfontolásból, hogy egy olyan navigációs alkalmazást fognak létrehozni, amihez hasonló még nem volt a piacon addig. Az alap elképzelés az volt, hogy egy közösségi navigációs alkalmazást fejlesztenek, ami megkönnyíti a forgalomfigyelést, a torlódások, dugók észlelését. A céget 2013-ban felvásárolta a Google, mivel ők is látták, hogy nagy lehetőség van a közösségi navigáció fejlesztésében.[14]

Az alkalmazásba a korábbi rendszerek bemutatása során említett funkciók bele is kerültek. A torlódások, forgalmidugók jelzése többnyire automatikusan történik, mivel az alkalmazás használatával beleegyezünk abba, hogy felhasználja a felhasználó sebességét, illetve pozícióját annak érdekében, hogy az alkalmazás például torlódást jelezhessen a többi felhasználó számára. Tehát abban az esetben, ha több Waze felhasználó is, egymáshoz viszonylag közel, egy főúton csak lépésben tud haladni, akkor az alkalmazás ezt úgy kezeli, hogy torlódás történt. Valahány forgalmi szituációt manuálisan lehet jelezni az alkalmazásban, sőt sokszor, ha torlódást érzékel az alkalmazás megerősítést

kér, hogy valóban torlódás van vagy egyéb ok miatt haladunk lassabban a maximálisan megengedett sebességnél. [15]

Az alkalmazás központi részét képezi a közösség, ha az emberek nem használják az alkalmazást, akkor a főbb funkciók haszontalanná válnak, ennek következtében a Waze nem azok közé az applikációk közé tartozik, amelyeket lehet offline módban használni. Az alkalmazás jelzi a felhasználó felé, hogy milyen sebességgel közlekedik és jelzést ad, ha a megengedett sebességet meghaladta, emellett ez az alkalmazás is jelzi a térképen azt, hogy hol parkolta le a felhasználó az autóját.

Az előző két navigációs szoftverrel ellentétben a Waze-nél érezhető, hogy kifejezetten az autósokra fokuszálva jött létre az alkalmazás, tehát gyalogosok vagy a tömegközlekedést használók számára nem ez a megfelelő alkalmazás navigálás céljából.



3.1. ábra: A három legnépszerűbb térképalkalmazás kezelőfelületei
(balról jobbra: Waze, Google Maps, Apple Maps)

3.4. NAVIGÁCIÓS ALKALMAZÁSOK ÖSSZEGZÉSE

Az előző fejezetekből kiolvasható, hogy a Waze-zel ellentétben a Google Maps és az Apple Maps sem csak a navigálásra lett létrehozva. Ez a két applikáció olyan plusz funkciókkal rendelkezik, amiknek köszönhetően a felhasználó akkor is megnyitja és használja a programot, amikor még nincs is szüksége navigálásra. Konkrét ilyen példa lehet, amikor a felhasználó éttermet keres, ezért megnyitja a térképalkalmazást és az applikáció által felkínált éttermek közül választ egyet az értékelések alapján.

A 3.1. ábrán az általam vizsgált alkalmazások grafikus felülete látható. A Waze és az Apple Maps hasonlóan letisztult kinézettel rendelkezik. Attól függetlenül, hogy a Google Maps ugyanazzal a célcsoporttal rendelkezik többnyire, mint az Apple Maps, a kinézet eléggé eltérő. A Google Maps esetében a felhasználónak egyértelműen jelezve, hogy van lehetősége éttermet, benzinkutat, boltot keresni, a konkurencia esetében ezeknek a funkcióknak az eléréséhez minimum egy ujjmozdulat szükséges. A Waze esetében a letisztultság azért érthető és szükséges, mivel csak autósok használják az alkalmazást, sokan közölük csak a forgalomfigyelésre, tehát célállomás megjelölése nélkül használják az alkalmazást és esetenként zavaró lenne a felhasználó számára, ha minden funkciógomb ki lenne helyezve a kezdőképernyőre.

A 3.1. táblázatban látható az általam választott navigációs rendszerek összehasonlítása néhány szempont alapján. A Google Maps jóval több havi felhasználóval rendelkezik, mint versenytársai. [16] Ez köszönhető annak, hogy az emberek 2005 óta ismerhetik ezt a térképszolgáltatást, a sok jól működő plusz funkciónak, annak, hogy a gyalogosokan,

bicikliseknek, autósoknak egyaránt készül ez az alkalmazás, illetve a platformfüggetlenségnek.

	Google Maps	**Apple Maps**	**Waze**
Platform	**Android, iOS**	**iOS**	**Android, iOS**
Felhasználók száma (millió fő/hónap)	**154**	**23,3**	**25,6**
Célcsoport	**gyalogos, biciklis, autós**	**gyalogos, biciklis, autós**	**autós**
Kiadás éve (telefonra)	**2007**	**2012**	**2006**

3.1. táblázat: Az általam vizsgált térképalkalmazások összehasonlítása néhány szempont alapján

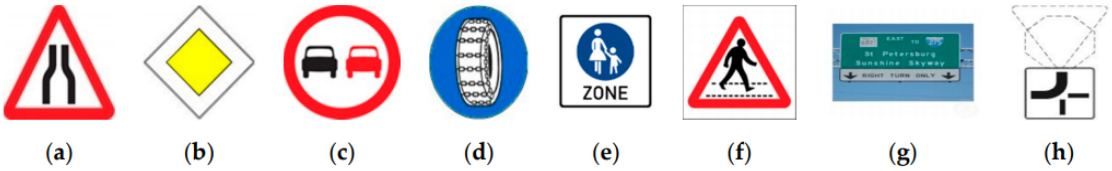
4. HASONLÓ RENDSZEREK

Ebben a fejezetben az általam választott hasonló rendszereket fogom bemutatni. A rendszerek kiválasztása során ügyeltem arra, hogy legyen olyan megoldás, ami a táblafelismerés feladatát hagyományos módon oldja meg, illetve olyan is, ami ugyanezt egy modern megközelítéssel valósítja meg.

4.1. TÁBLAFELISMERŐ RENDSZEREK

Az autógyártók célja hosszú évek óta az, hogy egyre inkább megkönnyítsék a vezetők dolgát a különböző autóba szerelt segédrendszerekkel. Ez egyaránt hivatott a vezető kényelmét és biztonságát szolgálni. Ilyen rendszer például az adaptív tempomat, a sávtartó elektronika vagy a táblafelismerő rendszer.

A napjainkban gyártott autók jelentős részében már az alapfelszereltség része a sebességkorlátozó táblákat jelző rendszer. A táblák különböző jelentéseket hordoznak magukkal, ezeket a jelentéseket három tényező összessége határozza meg: a tábla színe, a tábla alakja és a táblán látható piktogram. A bécsi közúti közlekedési egyezménynek köszönhetően a közlekedési táblák többnyire megegyeznek az egyezményt aláírt országokban. Ebben az egyezményben 8 csoportba kategorizálták a KRESZ táblákat, amely csoportokat a 4.1. ábra hivatott bemutatni. [17]



4.1. ábra: Veszélyt jelző tábla (a), Elsőbbséget szabályozó jelzőtábla (b), Tilalmi jelzőtábla (c), Utasítást adó jelzőtábla (d), Különleges szabályt jelző tábla (e), Tájékoztató jelzőtábla (f), Útbaigazító és utaló jelzőtábla (g), Kiegészítő jelzőtábla (h)

A táblafelismerés mint feladat tipikusan abba a csoportba tartozik, amit hagyományos módszerekkel, illetve napjainkban közkedvelt, modern módszerekkel is meg lehet valósítani. A hagyományos módszerrel létrehozott rendszerek esetenként pontosabbak lehetnek modern társaival összehasonlítva, azonban a különböző változásokat (például látásviszonyok) a modern módszerrel létrehozott rendszerek viselik jobban. Hagyományos módszerek esetében a táblák legszembetűnőbb tulajdonságaival lehet dolgozni.

A hagyományos módszerrel működő táblafelismerő rendszerek működésének három fázisa van: az észlelés, a követés (tracking), és az osztályozás/besorolás. Az észlelés fázisában a bemeneti képen kell a rendszernek megtalálnia azt, hogy az adott tábla hol helyezkedik el, tehát a ROI-t (Region Of Interest) kell meghatároznia, a besorolás fázisában pedig azt kell megadnia, hogy az általa ismert táblák alapján melyikre hasonlít a leginkább, melyikkel egyezik meg. [18]

A jelzőtáblák színei és alakjai nagyon jellemzőek, így azok elkülöníthetőek az egyéb, a táblafelismeréshez nem szükséges objektumoktól. Leginkább ennek a tulajdonságának köszönhető az, hogy hatékonyan lehet használni a gépi látás adta lehetőségeket az ilyen típusú feladatokon. Az észlelést három féle módszerrel lehet megoldani színalapú, alakzatalapú módszerrel vagy hibrid (szín- és alakzatalapú) észleléssel. [18]

A színalapú módszer alkalmazása azért tud hatékonyan működni, mert a jelzőtáblák tervezésekor a láthatóság miatt szándékosan kontrasztos színeket alkalmaztak az alkotók, hogy nagyon jól kivehetőek legyenek. Az alapból kontrasztos színek mellett, nagy segítség a megfelelő színtér megválasztása (RGB, HSV, HSI stb.). A módszer előnyei közé sorolható, hogy viszonylag alacsony a számítási igénye, legnagyobb hátránya, viszont az, hogy nagyon sok minden befolyásolhatja a detektálás sikerességét. Ilyen tényezők lehetnek például a táblának a távolsága, az időjárási körülmények, a napszak, a látásiviszonyok és számos egyéb tényező. [18]

A táblák színe mellett azok alakja is kellőképp jellemző, így az is egy lehetőség, hogy ez alapján történjen a szegmentálás, az alakzatalapú módszer ezt használja ki. Ez a módszer nem igényli azt, hogy színes képeket használjunk, így szürkeárnyalatos képekkel is használható. Ahhoz, hogy meg lehessen állapítani egy tábla alakját először annak körvonalait szükséges megtalálni. Ennek a módszernek az előnye, hogy a detektálást nem befolyásolják a fényviszonyok, hátránya azonban az, hogy nagyon erőforrásigényes, illetve elmosódott vagy valamennyire eltakart táblákat nehezebben ismer fel. [18]

A hibrid módszer az a megközelítés, amikor a színalapú és alakzatalapú módszereket közösen felhasználva jön létre a táblafelismerő rendszer, ennek legfőbb oka az, hogy még pontosabb legyen a jelzőtábla felismerés. Egyes esetekben ezt a módszert úgy alkalmazták, hogy a színalapú módszerrel szegmentálták a képet, majd a szegmentált képen történt az alakzatfelismerés, mivel azonban az RGB színtér érzékeny a különböző fényviszonyokra/visszaverődésekre ezért azt átkonvertálták HSI színtérbe.

A napjainkban eléggé elterjedt módszer, hogy valamilyen neurális hálózatot használunk fel a táblák detektálására és klasszifikálására egyaránt. A hardverek, illetve a neurális hálózatok fejlődésének köszönhetően, mára már léteznek akár valós időben felhasználható hálózatarchitektúrák, amik kifejezetten a különböző objektumok detektálására és klasszifikálására jöttek létre. Ilyenek például a YOLO, RCNN (Fast-RCNN, Faster-RCNN), SSD.

A hagyományos és a modern módszer akár egyszerre is felhasználható, abban az esetben, ha a tábla szegmentálását valamilyen hagyományos módszerrel oldjuk meg, majd a szegmentálás eredményét továbbadjuk egy konvolúciós neurális hálózatnak. Ezen hálózatok felhasználhatják akár a VGG16, InceptionV3 vagy a ResNet50 architektúrák egyikét.

4.1.1. Road-Sign Detection and Recognition Based on Support Vector Machines

A tanulmány 2007 júniusában született, melynek kutatói a táblafelismerés problémáját SVM (Support Vector Machine) segítségével oldották meg.

Ebben a tanulmányban a kutatók egy hibrid módszert használtak, tehát a táblák színe mellett a táblák alakját vizsgálva történik meg a detektálás, majd a klasszifikálás. A színalapú módszer egyik nagy hátránya, hogy a látási viszonyok nagyon befolyásolják a szegmentáció sikerességét, éppen ezért javasolt a megfelelő színtér megválasztása. Ebben az esetben a kutatók az RGB (Red, Green, Blue) színteret a HSI (Hue, Saturation, Intensity) színtérre cserélték. A HSI színtér valóban jó választásnak bizonyult a projekten dolgozók tesztjei alapján, azonban fehér táblák esetén nem tűnt használható megoldásnak, mivel ezek esetében a színárnyalat és a szaturáció komponensek nem tartalmaztak elegendő információt a szegmentáláshoz. Annak érdekében, hogy a fehér táblákkal is tudjanak dolgozni a színtelen dekompozíció (achromatic decomposition) módszere mellett döntöttek. [19]

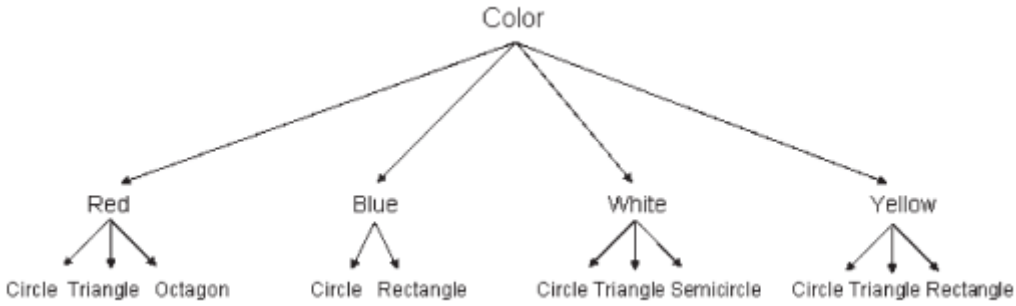
A képre a szegmentálást követően felkerülnek a maszkok, amiknek az a dolga, hogy a képen már csak a valóban hasznos pixelek maradjanak meg, tehát optimális esetben a maszkok hatására a képen csak egy tábla vagy több tábla látszik, a háttér pedig kap valamilyen semleges színt, általában ez a szín a fekete. Optimális eset azonban nagyon ritkán adódik, sokszor olyan pixelek is maradnak a képen elsötétítetlenül, amelyek nem tartoznak semmilyen táblához. Ennek kiküszöbölésére határszámokat (threshold) határoztak meg a projekt kutatói arra vonatkozóan, hogy mekkora objektumok azok, amik számukra hasznos információt hordoznak. Ezzel a döntéssel lecsökkentették annak a lehetőségét, hogy a képen a klasszifikáció szempontjából irreleváns pixelek elsötétítelenül maradhassanak. A táblát ezután a lépés után már be lehet sorolni a megfelelő színcsoportba. [20]

Az így kapott képen már meghatározható a BoI (Blobs of Interest), tehát az a hely, ahol az adott tábla elhelyezkedik. Abból adódóan, hogy a rendszer alakzat klasszifikációért felelős része érzékeny a táblák tájolására, ezért mielőtt a tábla idekerülne a megfelelő referencia pozícióba lesz mozgatva, így elérték, hogy a rendszer működését nem befolyásolja a táblák elhelyezkedése. Miután megtörtént a referencia pozícióba állítás a táblákat alakzat szerint osztályokba kell sorolni, amihez SVM-et (Support Vector Machine: Szupport Vektor Gép) használtak. [21]

Az SVM eredetileg bináris osztályozási problémák megoldására lett létrehozva, azonban van lehetőség többosztályú klasszifikációra is felhasználni a rendszert. Ebben a projektben one-versus-all elvű, vagyis „egy mindenki ellen” SVM-et használtak a fejlesztők, amely többosztályú klasszifikáció problémáját visszavezeti a bináris klasszifikáció problémájára. Mindez úgy van megvalósítva, hogy minden osztályhoz, jelen esetben táblához létrehozza a két osztályt. Az egyik osztályban csak egy féle tábla van, nevezzük ezt főtáblának, a másik osztályban az összes többi tábla szerepel, legyenek

ezek a melléktáblák. Ennek a megközelítésnek egyik nagy hátránya a főtábla és a melléktáblák mintáinak számbeli különbségével kapcsolatos. Ez a kiegyensúlyozatlanság a klasszifikáció eredményességét is befolyásolhatja. Másik megvalósítási módja az SVM-es több osztályú klasszifikációnak a one-versus-one, vagyis az „egy-az-egy” ellen elv. Ez az elv szintén bináris klasszifikációs problémára van visszavezetve, azonban itt egy táblacsoport egy másik táblacsoporttal van összevetve, tehát itt a kiegyensúlyozatlanság mint probléma kevésbé jöhet elő. [19]

A tréning fázis során a rendszer osztályoként átlagosan 50 darab képet kapott. Az osztályba besorolás, tehát úgy lett megoldva ebben a rendszerben, hogy először a tábla



4.2. ábra: Jelzőtáblák csoportosítása szín, majd alak szerint

színe kerül megállapításra, majd aszerint történik a tábla alakjának megállapítása, majd maga a klasszifikáció, hogy mi is a konkrét tábla, ennek a folyamatnak a lépései láthatók a 4.2. ábrán.

A felismerés fázisa azonnal elkezdődik, amint az adott tábla be lett sorolva a megfelelő színcsoportba, majd alakzatosztályba. Ebben a projektben a klasszifikálás szintén SVM-mel lett megoldva egy Gauss kernel segítségével. A tréning elvégzéséhez a LIBSVM könyvtárat használták, ami elérhető többek között Python, C++, Java programozási nyelvekre. A bemenete ennek a fázisnak egy 31 x 31 pixelből álló szürkeárnyalatos kép. Az előzetes besorolásoknak köszönhető, hogy az egész rendszer hatékonyan tud működni, mivel a felismerés fázisában először a szín lesz megállapítva, majd ezután az alak, így a rendszer például egy STOP tábla esetében nem fogja átvizsgálni az összes szín összes alakját, hanem azonnal a piros színű táblák között fog keresni. [19-20]

Tesztelési körülmények	1	2	3	4	5
Képek száma	749	1774	860	995	798
Jelzőtáblák száma	21	21	20	25	17
Detektálások száma	218	237	227	285	127
Téves észlelések száma	0	3	4	8	7
Téves felismerések száma	4	4	4	2	7

4.1. táblázat: Teszteredmények
(Az 1. a 2. és a 3. számokkal jelölt oszlopok esetén a teszt napos körülmények, a 4. az esős időjárási körülmények között, míg az 5. az éjszaka zajlott)

A kész rendszer tesztelése során egy videókamera által rögzített videó képkockái lettek átadva a rendszer bemenetének. A kamera a szélvédő mögött volt rögzítve és a megengedett maximális sebességgel haladt a jármű a felvétel során. A felvétel képkockái „.bmp” fájlformátumba lettek konvertálva, hogy megfelelő legyen a bemenet a rendszer számára. Két megfelelő kép közti időkülönbség nagyjából 0,2 másodperc volt, ami azt jelenti, hogy hozzávetőlegesen olyan, mintha 5 FPS-sel (Frame Per Second) történt volna meg a videófelvétel. Az algoritmus C programozási nyelvben lett megalkotva és a feldolgozási idő egy 720 x 576 kép esetén 1,77 másodperc volt. A 4.1. táblázatban látható a rendszer működésének eredményessége. Jól látszik, hogy esős időben valamivel több volt a tévesen jelzett táblák száma, mint sötétben, illetve az is látszik, hogy sötétben a hibás felismerések száma jóval több volt, mint esőben.

4.3. ábra: Mesterséges kitakarások elhelyezése a képen

A fejlesztők tesztelni szerették volna az érzékenységét a rendszerüknek a különböző kitakarásokra, ezért mesterségesen létrehozott kitakarásokat helyeztek el az általuk készült képeken. A kutatók nem említik a kitakarások nagyságát, de három különböző méretű zavaró tényezővel dolgoztak. A 4.3. ábrán megfigyelhető, hogy nagyságrendileg

mekkorák voltak a kitakarások, illetve az, hogy hova helyezték azokat. Kis méretű elfedés esetén a rendszer 93,24%-os, közepes méretű elfedés esetén 67,85%-os és nagy méretű elfedés esetén 44,9%-os a pontosságot produkált. [19]

A kutatók felvetettek továbbfejlesztési lehetőségeket, amikkel még jobbá tudnák tenni saját rendszerüket. A valós időben való működést jelölték meg egy nagy újítási lehetőségnek, azonban ez hatalmas erőfeszítéseket igényelne, ismervén az aktuális rendszer képességeire vonatkozó számokat. A fals jelzések számát egy követés funkcióval csökkentenék, ami annyiért lenne felelős, hogy figyelje, az adott tábla egymás után két képkockában is megjelent-e és kiértékelésre került, ha ez nem történt meg, akkor téves riasztásnak könyvelnék el. A megoldásukkal tapasztaltak egy olyan hibát, hogy azonos színű és alakú tábláknál, nagyon kis eltérések esetén a rendszer hajlamos a helytelen klasszifikálásra, emiatt a szegmentációs fázist módosítanák. [19]

4.1.2. Faster R-CNN for Small Traffic Sign Detection

A tanulmány 2017 végén született. A kutatók célja egy olyan rendszer megalkotása volt, ami a neurális hálózat adta lehetőségeket kihasználva, képes legyen kifejezetten kisméretű közlekedési jelzőtáblák detektálására és azok klasszifikálására.

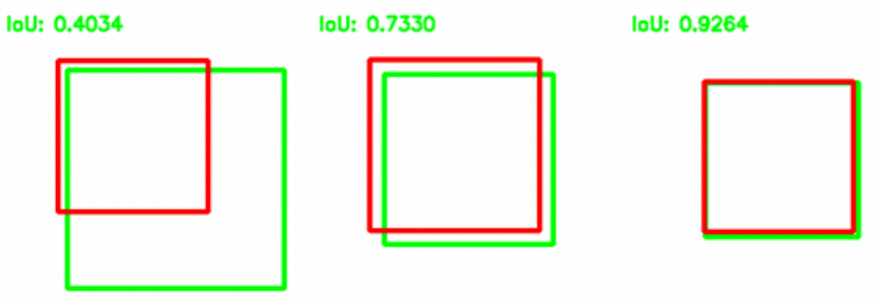
A kutatók projektjük alapjául a Faster R-CNN (továbbiakban FRCNN: Faster Region Based Convolutional Neural Network) neurális hálózatot választották. Ez a neurális hálózat napjaink egyik legnépszerűbb neurális hálózatarchitektúra valós idejű objektum felismerés esetén, ha valamilyen objektum felismerési feladatról van szó, mivel a detektálás mellett a klasszifikálást is elvégzi a hálózat. Annak köszönhetően, hogy a grafikus gyorsító kártyák egyre nagyobb teljesítménnyel rendelkeznek, az ilyen típusú hálózatok is egyre elterjedtebbek, mivel a pontosságuk mellett a sebességük is egyre nőtt. [21]

A kutatók a FRCNN neurális hálózatot a saját adathalmazukkal, illetve a VOC 2007 (Visual Object Classes Challange) adathalmazzal tesztelték. A két adathalmazban a fő különbség a detektálni kívánt tárgyak mérete volt. A táblák jóval kisebb területet foglaltak el a képeken, mint a VOC 2007 képein látható objektumok. Abban az esetben, ha növelni szerették volna a pontosságot, akkor a képeket fel kellett skálázni az eredeti méretük kétszeresére. Ezzel már a táblák mérete közelebb került a VOC 2007 képeken található objektumok méretéhez, azonban csökkent a sebesség. A másik esetben nem történt felskálázás, ami nyilvánvalóan jót tett a sebességnek, tehát nem lassult a hálózat, azonban ebben az esetben a pontosság romlott. A kutatók célja tehát egy olyan rendszer megalkotása, ami képes jó teljesítmény mellett kielégítő pontosságot is nyújtani. A képek felskálázása során a receptív mező (RF: Receptive Field) mérete nem változik, azonban arányaiban kisebb lesz, aminek következtében a kisebb méretű objektumok esetében növekedni fog a pontosság az FRCNN esetében. [21]

A kutatók a céljuk elérése érdekében módosítják a régió becslő hálózatot (továbbiakban RPN: Region Proposal Network), aminek segítségével a neurális hálózat már képes lesz jobb eredményeket produkálni kis méretű jelzőtáblák detektálásában, emellett módosítják az FRCNN felépítését is, konvolúciós hálózattal helyettesítik az eddigi teljesen csatolt hálózatrészt, ennek a döntésnek köszönhetően a hálózat pontossága nem változik, viszont a mérete csökkenni fog. [21]

A FRCNN kétlépcsős objektum detektálóval rendelkezik. A hálózat első része az RPN, aminek a feladata az objektum régiók generálása, majd ezt adja át a hálózatnak, ami az osztályokba besorolásért felelős. Mivel a helyesen jelölt régiók nagyban befolyásolják végül a hálózat klasszifikációs pontosságát, ezért kell ezen a részen módosítani, hogy a kisebb méretű táblákkal is jól működjön a rendszer. [21]

A receptív mezőnek nagy szerepe van a hálózat helyes működésében. Abban az esetben, ha ezen területek túlméretezettek, az az egész folyamatra negatív hatással hathat, emiatt fontos, hogy ez a terület a lehető legjobban körülölelje a detektálni kívánt objektumot és ne legyenek benne felesleges részek, amely a hálózat számára irreleváns információkat tartalmazhat. Mivel a kutatók tapasztalták, hogy a felskálázott kép esetén az arányaiban kisebb receptív mező is pozitívan hatott a hálózat pontosságára, ezért a receptív mező területének csökkentése mellett döntöttek. A receptív mező méretét a konvolúciós filter mérete és annak lépésköze határozza meg, tehát ezen paraméterek módosításával elérhető a receptív mező méretének csökkenése. [21]



4.4. ábra: Különböző IoU értékekkel rendelkező keretek összevetése

Az RPN különböző méretű és arányokkal rendelkező referenciaablakokkal dolgozik, melyek középpontjai azonosak. Egy képen több ezer ilyen ablak kerül elhelyezésre. Ezen ablakok átfedésének arányát adja meg az IoU (Intersection-over-Union). A rendszer az IoU értékéből tud következtetni, hogy a kép adott részén található-e objektum vagy sem. A rendszer akkor könyveli el a „jó keretnek" az ablakot, ha az IoU érték meghaladja a 0,5-öt. Kisméretű objektumok esetén az aránylag nagyméretű ablakokkal kevesebb helyes keret jöhet létre, aminek következményeként a detektálás és a klasszifikáció eredményessége is romlik. A kutatók ezért ezeknek az ablakoknak a méretét is csökkentették. Kisméretű objektum esetén a „javasolt régió" mérete is kicsi lesz, aminek következtében a jellemzőtérkép minősége nem lesz megfelelő.

Ezt a problémát a konvolúciós filter méretének változtatásával, illetve nyújtott konvolúció (dilated convolution) használatával oldják meg. [21]

A kutatók annak érdekében, hogy igazolják saját megoldásuknak hatékonyságát, az általuk készített rendszert az eredeti Faster R-CNN neurális hálózattal hasonlították össze az átlagos precízió (AP) alapján, az ehhez szükséges IoU threshold érték 0,5 volt. Az adatokat CCF BDCI 2016 adathalmaz szolgáltatta, ami 4000 képet tartalmaz, melyeknek mérete 1280 x 720 pixel. Ezeket a képeket egy forgalomban közlekedő autó rögzítette, ebből adódik, hogy a képek különböző szögekből készültek és a fényviszonyok sem voltak a legmegfelelőbbek és egységesek minden esetben. A rögzített képeken összesen 28000 darab közlekedési tábla található, tehát átlagosan képeként 7 tábla szerepel egy képen. A projektben résztvevők a 4000 képpel rendelkező adathalmazból kiválasztottak véletlenszerűen 3000 darabot a tanítási fázisra, a maradék 1000 darab képet a validációhoz használták fel. [21]

	Receptív mező	Model mérete	Sebesség	Average Precision
FRCNN-ZF-1x	171	255 Mb	0,14 s	30,7%
FRCNN-ZF-2x	85	255 Mb	0,37 s	44,6%
RFnet-ZF	83	34 Mb	0,41 s	50%
RFnet-ZF-dilated	85	29 Mb	0,39 s	48,2%

4.2. táblázat: A rendszerek összehasonlítása különböző szempontok alapján

A három különböző rendszert összesen négyféle módon konfiguráltak és ennek a négy konfigurációnak az eredményeit vetették össze a kutatók. Az eredeti Faster R-CNN-t tesztelték eredeti méretű képekkel, illetve kétszeres méretre felskálázott képekkel és a saját megoldásukat nyújtott konvolúcióval és anélkül. Az egyes konfigurációk által elért precíziókat a 4.2. táblázat hivatott bemutatni.[21]

Minél nagyobb az AP értéke annál pontosabb az adott rendszer. Az eredményekből jól látszik, hogy a kétszeres mérettel rendelkező képek esetében a 13,9%-kal jobb teljesítményt nyújt az F-RCNN, mint az eredeti mérettel rendelkező képek esetén. [21]

4.1.3. A YOLO Based Approach for Traffic Sign Detection

Ez a kutatás 2018-ban született a közúti jelzőtáblák gyors és hatékony felismeréséről. Ahogy a Faster RCNN projektnél is olvasható volt, a GPU-k fejlődésének köszönhető, hogy mostanra van esély konvolúciós neurális hálózatokat használni olyan problémák megoldására, ahol a valós időben való objektum detektálás és klasszifikálás a cél. A kutatók a saját projektükben egy YOLO architektúrára épülő megoldást hoztak létre, mivel ez lényegesen gyorsabb és ebből adódóan alkalmasabb a valós időben történő használathoz, mint az RCNN architektúra. A YOLO architektúra megközelítőleg 45

képkockát tud feldolgozni másodpercenként, ami egy ilyen típusú feladatra elegendőnek bizonyult. [22]

A kutatók a BTSD (Belgium Traffic Sign Dataset továbbiakban: BTSD) adathalmaz mintáit használták fel projektjük megvalósításához. Az adathalmaz összesen 12 osztályt tartalmaz, ami a táblák színére, illetve alakjára vonatkozik. A 12 osztály közül van egy címkézetlen (unlabeled) osztály is, ahova azok a képek kerültek, amik egyik csoportba sem besorolhatók. Háromszög alakú, rombusz alakú, piros kör alakú, függőleges téglalap alakú, kék kör alakú, piros és kék kör alakú, fordított háromszög alakú, vízszintes téglalap alakú, négyzet alakú táblák, illetve stop tábla, tiltó tábla és egyéb közúti jelzőtáblák. Az adathalmaz képei városi közlekedés során készültek. Az ilyen típusú projekteknél elengedhetetlen a megfelelő számú tréning, validációs és teszt kép. A tréning képek száma 5905, a validációhoz szükséges képek száma 1301, a teszt képek száma 1750 darab. Az összesen 8956 képen lehetnek olyan esetek, amikor egy képen több jelzőtábla is látható, de olyan is előfordulhat, hogy egyáltalán nem tartalmaz semmilyen táblát az adott kép. Annak köszönhetően, hogy a képek valós forgalomban közlekedve készültek, ezért sok esetben a táblák az adott képen viszonylag kis területet foglalnak el, emellett az orientáció és a képek megvilágítása is változó az adathalmazban. Az összes kép 448 pixel széles és 448 pixel magas méretre méretezték át a projekten dolgozók, mielőtt azokat megkapta volna a neurális hálózat. A BTSD adathalmazhoz tartozó annotációk tartalmazzák, hogy az adott képen hol helyezkedik el a jelzőtábla (x és y koordináták, illetve a szélessége és a magassága a táblát körülölelő doboznak) és a képen látható tábla kategóriáját, hogy melyik osztályba tartozik. [22]

A kutatás során egy módosított YOLO architektúrát hoztak létre a szerzők. A módosítás elsősorban az architektúra veszteségfüggvényének megváltozása jelentette. Az eredeti architektúrában a hálózat egyenlő mértékben koncentrál a kisebb, illetve a nagyobb objektumokra, ami jelzőtáblák esetén nem szerencsés, mivel a képeken a felfedezni kívánt objektum az esetek többségében kis területet foglalnak el a képen. Ennek következtében úgy módosították a kutatók a veszteségfüggvényt, hogy a hálózat kifejezetten a kis mérettel rendelkező objektumokra koncentráljon, amit úgy értek el, hogy a hálózat nagyobb büntetést kapott abban az esetben, amikor egy kis objektumot nem tudott pontosan detektálni. [22]

Annak érdekében, hogy a különböző hálózatok teljesítményét össze lehessen vetni, egységes metrikákra van szükség. Az objektum detektálásának, illetve a detektált objektumok klasszifikálásának a jóságát mAP-ben (Mean Average Precision) szokás mérni. Az átlagos precíziót (Average Precision) a PR (Precision-Recall) görbe alatti terület határozza meg. [22]

A hálózat által nyújtott predikciók négy csoportba besorolhatók:

- TP: True Positive
- FP: False Positive
- TN: True Negative
- FN: False Negative

Maga a precízió (precision) értéke annyit jelent, hogy a hálózat által adott predikciók mennyire pontosak. A precíziót úgy számítjuk ki, hogy a helyes predikciók számát elosztjuk a helyes és helytelen predikciók számának összegével, így kapjuk meg a pontosságot. [22]

Az adott predikció attól függően helyes vagy helytelen, hogy az IoU vagyis az Intersection over Union, illetve a hozzá tartozó threshold érték mekkora. Threshold értéknek általában 0,5 körüli értéket szokás megadni. Amennyiben a kapott érték meghaladja a threshold értékét, akkor a predikció helyes, azonban, ha nem haladja meg, akkor az a predikció falsként lesz elkönyvelve. További fontos metrika a Recall, ami azt adja meg, hogy az összes valóban megtalálható eset közül mennyit sikerült megtalálnia a hálózatnak, tehát ebből tudjuk megállapítani, hogy mekkora aránya helyes a predikcióknak. [22]

Osztály neve	**Eredeti loss függvény (AP)**	**Módosított loss függvény (AP)**
redbluecircle signs	0,393	0,673
redcircle signs	0,517	0,333
bluecircle signs	0,333	0,428
revtriangle signs	0,416	0,222
rectanglesdown signs	0,454	0,562
triangles	0,136	0,66
forbidden signs	0,449	0,25
diamond signs	0,126	0,666
rectanglesup signs	0,19	0,833
stop signs	0,095	0,25
other defined traffic signs	0,384	0,5
undefined	0,32	0,98
mAP	**0,332**	**0,525**

4.3. táblázat: Az eredeti és módosított loss függvénnyel rendelkező hálózat teljesítményének összehasonlítása

Az mAP értéke úgy számolható ki, hogy az összes osztály AP értékét összegezzük, majd ezt az összeget elosztjuk az osztályok számával. [22]

A neurális hálózat megalkotásához a kutatók a TensorFlow keretrendszert használták. A hálózat betanítására és használatára egy 56 maggal rendelkező processzort és egy Nvidia Tesla M40 GPU-t használtak. Az előbb említett hardverek is alátámasztják azt a tényt, hogy a YOLO architekturához hasonló neurális hálózatok valóban képesek valósidejű működésre, abban az esetben, ha ehhez megvannak a megfelelő eszközök. [22]

A kutatók a BTSD adatokkal tanították be neurális hálózatukat először az alap, majd az általuk módosított loss függvénnyel. A két megoldás összehasonlítása az mAP értékek szerint történt. Az eredményekből kiolvasható, hogy határozottan jobb eredményt produkál az a YOLO hálózat, ami a módosított veszteségfüggvényt használja. [22]

4.1.4. HASONLÓ RENDSZEREK ÉRTÉKELÉSE

A hasonló rendszerek fejezetbe szándékosan olyan, már megvalósított rendszereket válogattam össze, amelyek rávilágítanak arra, hogy milyen megoldások merülhetnek fel egy ilyen feladat megvalósítása esetén, illetve hogy az elkészült rendszerek mikre lehetnek képesek.

Az első hasonló rendszer hagyományosnak nevezhető eszközökkel dolgozik. A dokumentum 2007-es születése is ezt a tényt támasztja alá. A kutatás célkitűzése nem az volt, hogy a táblákat egy magasabb szintű osztályba sorolja be, tehát a kész rendszer nem azt állapította meg, hogy az adott tábla egy tiltótábla-e vagy sem, hanem az, hogy annak a táblának mi a konkrét típusa, azonban az osztályok pontos száma nem volt megemlítve a tanulmányban. Ebben a projektben kutatók a szegmentálás és a klasszifikálás fázisát elkülönítették egymástól, ami a későbbi rendszerekre nem feltétlen jellemző. Működés tekintetében jó megoldásnak bizonyult, hogy először színek szerint történik egy csoportosítás, majd alakzat szerint és csak ezután történik meg a valós klasszifikáció, ez a döntés sok erőforrást megspórolhatott. SVM-et használtak az alakzat, majd az osztályba besorolás fázisainál is, ami attól függetlenül, hogy eredetileg bináris klasszifikációra került létrehozásra, megfelelően működött ebben az esetben is. A kutatás végén továbbfejlesztési célként van megemlítve a valós időben való működés, ami az akkori eszközökkel eléggé elérhetetlennek tűnt, a következő két hasonló rendszert szándékosan azért választottam, mert azoknál már elérhető cél a valós időben való működés.

A következő két hasonló rendszer, a napjainkban egyre inkább elterjedő neurális hálózatokat, azon belül is konvolúciós neurális hálózatot használ. A már létező architektúrák önmagukban jó eredményre képesek ilyen típusú feladatokban, azonban mindkét esetben a kutatók módosítottak az eredeti hálózaton, hogy a kész rendszerük még jobb, még pontosabb legyen, mint az eredeti architektúra esetében.

A második hasonló rendszer a Faster-RCNN neurális hálózati architektúrára alapul. A kutatók felfedezték azt, hogy kis méretű objektumok esetén az eredeti hálózat kevésbé

működik jól, mint nagyobb méretű objektumok esetén, mivel a táblák az előbbi csoportba tartoznak, ezért a kutatás célja ennek a problémának a megoldása, csökkentése volt. A bemeneti képek módosítása helyett az architektúrán végeztek el módosításokat. Megfigyelték, hogy a kisebb receptív tér jótékony hatással van a hálózat működésére, ezért a módosították a receptív tér nagyságát, illetve szintén megfigyelték, hogy nyújtott konvolúció használata esetén a már módosított architektúra még jobban fog teljesíteni, mint addig. Az eredményekből, illetve a módosított architektúra paraméterein látszik, hogy egy jóval kisebb hálózatot sikerült megalkotniuk a kutatóknak, aminek sebessége közel azonos, viszont pontossága jóval magasabb, mint az eredeti hálózat esetében.

Az utolsó hasonló rendszer a YOLO neurális hálózati architektúráját használja fel saját megoldásában. A kutatók tanítási és validációs adatoknak a BTSD adathalmazt használták, ami összesen 12 osztályból áll, ezek közül az egyik osztály a címkézetlen osztály, tehát ebbe a csoportba tartoznak azok a képek, amiken semmilyen tábla nem szerepel. Véleményem szerint ennek a kutatásnak egy gyenge pontja, hogy csak 12 osztályból áll az adathalmaz, tehát klasszifikálásnál sem konkrét táblát, hanem csak egy csoportot ad vissza a hálózat válaszként, amibe az adott tábla beletartozik. Ennek a projektnek a kutatói is megfigyelték azt, hogy mivel a táblák kis területet foglalnak el az egész képből, ezért a hálózat teljesítménye nem a legjobb. Ezt sajátosságot azzal küszöbölték ki, hogy az eredeti architektúra veszteségfüggvényét módosították. A módosítás hatására a hálózat nagyobb büntetést kapott akkor, amikor egy kisebb objektumot nem tudott pontosan detektálni. A veszteségfüggvény módosítása egy jó döntésnek bizonyult. Az eredeti és a módosított architektúra eredményeit hasonlították össze a kutatók, amikből kiderül, hogy majdnem minden osztály esetén jobb értékeket produkált a módosított hálózat.

	Első rendszer	**Második rendszer**	**Harmadik rendszer**
Felhasznált technológia	SVM	FRCNN	YOLO
Sebesség (kép/másodperc)	5	2,6	45*
mAP érték	-	0,482	0,525
Rendszer válasza	konkrét tábla	konkrét tábla	táblacsoport
Adathalmaz	saját képek	CCF BDCI 2016	BTSD

4.4. táblázat: Általam választott hasonló rendszerek összehasonlítása

5. NEURÁLIS HÁLÓZATOK

A gépi tanuláshoz köthető neurális hálózat a biológiai neurális hálózatokhoz hasonlóan egymással összekötött neuronokból áll. A neurális hálózat egyik legnagyobb erénye, hogy nagy méretű, komplex feladatokat is meg tudunk oldani velük. A hátránya abból adódik, hogy a belső működésre egyáltalán nem látunk rá, tehát abban az esetben, ha a hálózat egy bizonyos bemenetre nem várt és egyben rossz kimenetet ad, nem tudjuk végigkövetni a döntések sorozatát, ebből következik, hogy a felmerülő hibák kijavítása hosszadalmas lehet.

A neurális hálózatokat a betanítás módja szerint két csoportba sorolhatjuk. Vannak azok a hálózatok, melyek felügyelt tanítással működnek és vannak azok a hálózatok, amelyek megerősítéses tanulással.

A felügyelt tanulással működő programok például az automata spamfelismerő a levelezőrendszerekben, kézírást felismerő szoftverek, beszédet felismerő szoftverek (például Siri, Google Assistant), fényképnézegető szoftverekben a csoportosító funkció. A felügyelt tanulásnak az alapja az, hogy legyen megfelelő minta adathalmazunk. Ebben az adathalmazban kívánt bemeneti értékekhez meg kell adnunk a megfelelő kimeneti értékeket is. Miután ez megtörtént a programnak olyan bemeneti adatokat kell adni, amiket addig még nem ismert és megfigyelni azt, hogy megfelelő kimeneteket ad-e a minta adathalmazt felhasználva. Leegyszerűsítve, a felügyelt tanulás feladata, hogy előre megadott bemenetek és kimenetek alapján megtanulja a szabályt, amiből később a számára ismeretlen bemenetekre a megfelelő kimeneteket adjon meg. Ennek gépi tanulási típusnak a hátránya, hogy a betanítása hosszú idő lehet, illetve rengeteg helyes mintára van ahhoz szüksége a rendszernek, hogy megfelelő pontossággal működjön.

A megerősítéses tanulással (reinforcement learning) működő rendszerek, a felügyelt tanulással ellentétben, nem az előre megadott minták alapján hozzák meg a számukra megfelelőnek tűnő döntést az adott helyzetben. A két módszerben megegyezik, hogy mindkettőnek szüksége van valamiféle visszajelzésre azzal kapcsolatban, hogy az adott pillanatban meghozott döntés helyes volt-e vagy sem. A megerősítéses tanulás esetében a visszajelzés jó döntés esetén a jutalom (reward), ellentétes esetben pedig a büntetés. A megerősítéses tanulás az, amikor az ágens (a tanuló) nem ismeri előre se a környezetet, amiben tanul, se a jutalomrendszert, tehát azt, hogy mire kap plusz pontot és mire kap büntetést. Az ágens a környezetben elhelyezkedő tanuló, akinek a feladata, hogy tanuljon meg egy bizonyos tevékenységet.

5.1. KONVOLÚCIÓS NEURÁLIS HÁLÓZATOK

A konvolúciós neurális hálózatokkal (Convolutional Neural Network: CNN) lehet nagyon hatékonyan megoldani azokat a problémákat, amelyeknek a középpontjában valamilyen képekkel kapcsolatos feladat áll. A hálózatot konvolúciós rétegek alkotják, ezen rétegek segítségével lehet felismerni különböző mintákat a képeken. A felismert

minták a felső rétegek esetén még egyáltalán nem bonyolult jellemzőket jelentenek, tehát felső réteg esetén a hálózat már képes például egy négyzetet felismerni. Minél mélyebbre megyünk, tehát minél több a réteggel rendelkezik a hálózat, annál összetettebb jellemzőket is képes felismerni a rendszer.

A hálózat egy képpel kapcsolatos feladat esetén a bemeneti képet konvolválja, tehát egy megadott filter/szűrő szerint mátrix szorzások történnek. A keret tulajdonságai mellett a lépésközt (stride) is beállíthatjuk, hogy a keret hány lépésenként dolgozza fel a képet. Ezen hálózatok pooling réteggel is rendelkeznek, ami a konvolvált eredményből nyeri ki a domináns jellemzőket, ez a réteg működhet úgy, hogy az adott elemek átlagát számolja, de úgy is, hogy az adott elemek közül a legnagyobbat választja ki. Az utolsó réteg teljesen összekötött (fully connected) réteg, aminek segítségével megtörténhet a klasszifikáció. Napjainkban egyre több előre elkészített architektúra van, ilyen például a VGG16, a ResNet50, az InceptionV3 stb., melyeket van lehetőség átképezni (transfer learning), így a saját problémánkra személyre szabni azokat.

5.2. OBJEKTUM DETEKTÁLÁS ÉS KLASSZIFIKÁLÁS KONVOLÚCIÓS NEURÁLIS HÁLÓZAT SEGÍTSÉGÉVEL

A hasonló rendszerek című fejezetből már kiderült, hogy léteznek olyan konvolúciós neurális hálózatarchitektúrák, amik kifejezetten a tárgyak detektálására és klasszifikálására lettek kitalálva. Ebben az alfejezetben két olyan népszerű architektúrát mutatok be, amiket ilyen esetekben használni szokás.

5.2.1. R-CNN, FAST R-CNN, FASTER R-CNN

Az R-CNN (Region-based Convolutional Neural Network) régióbecslőkre (region proposals) alapul. Az adott képen a hálózat először legenerál, majd elhelyez 2000 darab régióbecslőt a „selective search” nevű algoritmus segítségével. Majd ezek a régióbecslők vannak átadva a hálózatnak bemenetként. A hálózat egy 4096 dimenziós kimenetet hoz létre minden régióbecslőhöz külön, ezt adja át az SVM-nek, ami megállapítja, hogy az adott területen van-e objektum vagy sem. Ezzel az architektúrával a legnagyobb probléma a nyújtott teljesítménye. Egy-egy kép feldolgozása 40 másodpercnél is többet igénybe vehet, ami nagyon soknak számít, ilyen típusú feladatok esetén. [23]

Az R-CNN továbbfejlesztése a Fast R-CNN. Ebben az architektúrában a fő módosítás az eredeti hálózathoz képest, hogy a bemeneti kép azonnal egy CNN-be érkezik, ahol még továbbra is „selective search” algoritmussal megtörténik a becsült régiók generálása. A módosítás azonban teljesítményben nagyon sokat jelent, ennek a hálózatnak használatával egy kép feldolgozása megközelítőleg 2 másodpercet vesz igénybe. [24]

A legnagyobb lépést a valós időben való felhasználáshoz, azonban a Faster R-CNN létrejötte jelentette. Ebben az architektúrában jelent meg az RPN, ami nagyban felgyorsította a rendszer működését. Maga az RPN megkapta a képet mint bemenet, majd a kimenete az RPN által létrehozott becsült régiók (region proposals) voltak. Minden

ilyen régióhoz kilenc referenciaablak tartozik, melyeknek mérete és aránya különböző, azonban középpontjuk megegyező. Itt már valóban opció lehet a valós felhasználás, ugyanis egy kép feldolgozása csak 0,2 másodpercet vett igénybe, ami már 5 képkockát jelent másodpercenként, azonban vannak rendszerek, amik ennél is jobb teljesítményre képesek. [25]

5.2.2. YOLO: YOU ONLY LOOK ONCE

A YOLO konvolúciós neurális hálózati architektúra már egy valós alternatíva arra az esetre, ha valós időben kell feldolgozni képeket. Ez az architektúra a bemeneti képre egy rácsot húz, majd határoló kereteket hoz létre az esetleges objektumok köré. Ezekhez a keretekhez öt darab értéket társít. Az x, y, w, h paraméterek a keret elhelyezkedésére és kiterjedésére vonatkoznak. Az ötödik paraméter a konfidencia érték, amit betanítás során a képen látható objektumok eredeti határoló kerete és a hálózat által prediktált keretének az átfedése (IoU) alapján számol a hálózat. [26]

Ez az architektúra pontosság terén nagyjából a Faster R-CNN-nel van egy szinten, viszont gyorsaság terén megközelítőleg tízszeres a sebességkülönbség a két hálózat között, ami ebben az esetben már nagyjából 45 képkocka feldolgozását jelentette másodpercenként. Ez a teljesítmény sem nevezhető rossznak, de az architektúra megalkotói létrehozták a Fast YOLO-t, ami 150 képkockát tudott feldolgozni másodpercenként, mindezt úgy, hogy a pontossága nem maradt el sokkal az eredeti YOLO-val szemben. Azt el lehet mondani a YOLO architektúráról, hogy nem feltétlen alkalmas kis mérettel rendelkező objektumok feldolgozására. Idővel megjelent több továbbfejlesztése is az architektúrának. A továbbfejlesztések közül a YOLOv3 volt az, ami már kis mérettel rendelkező objektumokkal is jól tud dolgozni, a YOLOv4 a v3-hoz képest még tudott javulni feldolgozási idő, illetve pontosság terén is. [26]

6. A RENDSZER SPECIFIKÁCIÓJA

A táblafelismerő rendszerek című fejezetben áttekintettem azokat a rendszereket, melyek sikeresen elkészültek és valós körülmények között is eredményesen működnek. A megoldások között volt, ami hagyományos módszert és SVM-et használva oldotta meg a táblafelismerés problémáját, illetve olyanok is, amik a napjainkban egyre népszerűbbé váló, kifejezetten objektum detektálásra specializálódott konvolúciós neurális hálózatokat használtak fel.

Én egy olyan rendszert fogok létrehozni, amit felhasználói felület szerint két részre lehet bontani. A rendszernek lesz egy Android operációs rendszerre készült alkalmazása, illetve egy böngészőből elérhető weboldala, ami minden platformról elérhető és használható lesz.

Az Androidra készülő alkalmazás elsősorban egy navigációs applikáció lesz, ami kifejezetten autós felhasználásra készül. Ez annyit jelent, hogy nem lesz lehetőség benne gyalogos vagy tömegközlekedéses útvonaltervezésre úgy, mint a Google Maps vagy az Apple Maps esetében. Az alkalmazás két üzemmóddal fog rendelkezni. Az első üzemmód maga a navigációs mód lesz, amiben a telefon elnavigálja a felhasználót az általa választott célpontra. Az applikáció másik üzemmódja a rögzítő mód lesz. Ebben a módban a telefon hátlapi kamerája lesz használva. A telefon a műszerfalon vagy a szélvédőn kell rögzítve legyen, hogy képes legyen a kamerájával az utat figyelni. A telefon másodpercenként fog fényképet készíteni, amit azonnal továbbít a szerver felé a telefon koordinátájával és orientációjával együtt, attól függetlenül, hogy a képeken látható-e bármilyen tábla vagy sem, ezeket a képeket a telefon nem fogja tárolni. A szerver dolga lesz az, hogy a fényképről megállapítsa, hogy tartalmaz-e táblát vagy táblákat és ha igen, akkor az melyik osztályba tartozik. A klasszifikáció fázisa után a tábla típusa és a táblához tartozó orientáció, koordináták mentve lesznek az adatbázisban. Az a felhasználó, aki csak a navigációs módot fogja használni, annyit fog tapasztalni, hogy időről-időre egyre több tábla lesz látható a térképen köszönhetően a többi felhasználónak.

A szerveroldal feladata lesz a kliensektől kapott kérések kiszolgálása. Ilyen kérések lehetnek a legközelebbi jelzőtáblák listázása vagy egy tábla rögzítése. Az Android kliensektől kapott képeket szintén a szerveroldalnak kell feldolgoznia. A felhasználóktól kapott nyersképeken látható jelzőtáblákat kell felismernie a rendszernek, majd a válaszként kapott táblák azonosítóját és azok orientációját, illetve koordinátáját rögzíteni az adatbázisban. Erre a feladatra logikus választás lehetne valamelyik verziójú YOLO neurális hálózat, mivel ez a hálózat nagyon rövid idő alatt képes lenne szegmentálni a táblákat, majd klasszifikálni azokat, azonban a rendelkezésemre álló adatok kevésnek bizonyulnának ennek az architektúrának a betanítására, emiatt egy köztes megoldást választottam. A rendszer a nyersképeket először a YOLOv4 hálózati architektúrának fogja átadni, ahol a hálózat csak szegmentálni fogja a képeken látható táblákat, majd ennek a kimenete közvetlenül továbbmegy a GoogleNet InceptionV3 nevű neurális

hálózatba, ami már a szegmentált képekről fogja eldönteni, hogy pontosan milyen táblák. A YOLOv4-et azért fogom tudni használni a táblák szegmentálására, mert minden egyes táblához nem elérhető kellő mennyiségű Yolo architektúrának megfelelő adat, azonban annak betanítására, hogy mi egy tábla egy képen, lényegesen kevesebb adat is elegendő.

A rendszer webes része, egy egyszerű weboldal lesz, amit csak és kizárólag manuális táblarögzítésre lehet majd használni. A megjelenített térképen kell majd kiválasztani egy pontot, majd ha ez megtörtént a felhasználónak lehetősége lesz kiválasztani a tábla típusát, annak irányát, majd ezen adatok rögzítésre kerülnek az adatbázisban. A felhasználó látni fogja azt is, hogy eddig milyen táblák lettek rögzítve a térképen, attól függetlenül, hogy az Androidos kliensből vagy a webes felületen történt a rögzítés.

A projektem alapja a térkép és a navigáció, ezért először azt szükséges megvizsgálnom, hogy milyen lehetséges SDK-t (Software Development Kit, továbbiakban: SDK) tudok felhasználni a feladatom megvalósítására. A két leghíresebb SDK, ami kifejezetten térképalkalmazásokhoz készült az a Google által készített Google Maps SDK, a mellé tartozó Google Directions API-val (Application Programming Interface, továbbiakban: API), ami az útvonaltervezéshez szükséges, a másik SDK a Mapbox SDK-ja, szintén a hozzátartozó Directions API-val. A Google megoldása a Mapbox megoldásához hasonlóan nagyon jól dokumentált, sok mintakóddal rendelkezik és az SDK-t felhasználó fejlesztők száma is mindkettő esetében nagyon magas, ami egy-egy probléma megoldása esetén hasznos lehet. Mindkét SDK jóval több funkcióval rendelkezik, mint amire nekem szükségem van, azonban egy dolgot a Google által kiadott SDK nem tud. Számomra fontos, hogy a felhasználó a navigációs üzemmód használata esetén ne legyen átirányítva egy másik alkalmazásba. A Google Maps SDK használata esetén azonban ez csak akkor érhető el, ha a fizetős szolgáltatását választom. Abban az esetben, ha a Google ingyenes SDK-ját szeretném használni úgy, hogy a navigálás lépésről lépésre történjen (turn-by-turn navigation) akkor a Google Maps SDK-ja esetén megnyitná a felhasználó telefonján a Google Maps alkalmazást és az én alkalmazásomban megtervezett utat töltené be a navigálás megkezdéséhez. Ezzel szemben a Mapbox SDK-ja ingyenes és biztosítja annak lehetőségét, hogy az ilyen típusú navigálás alkalmazás váltás nélkül megtörténjen a saját applikációmban, ezért a saját projektemben a Mapbox SDK-ját fogom használni. [27-28]

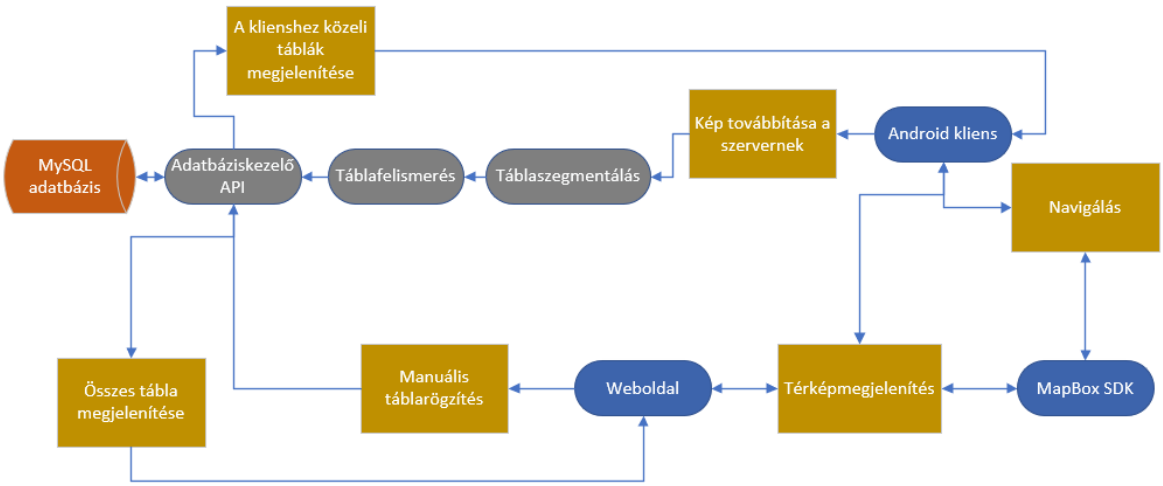
6.1. VÁZLATOS SPECIFIKÁCIÓ

- A rendszernek rendelkeznie kell külön kliens- és szerveroldallal
- A rendszernek tartalmaznia kell egy Android operációs rendszerre és egy webre készülő klienst
    - A weboldalon legyen lehetőség manuális táblarögzítésre
    - Az Android kliensnek két funkcióval kell rendelkeznie
        - Navigálás funkció MapBox SDK segítségével: A felhasználó által kiválasztott célpontra kell a programnak elnavigálnia

        - Automata táblarögzítés funkció: Az alkalmazásnak képesnek kell lennie minimum másodpercenként egy képkocka továbbítására a szerver felé
- A szervernek képesnek kell lennie a kliens által kapott adatokat feldolgozni, majd a feldolgozást követően az adatot rögzíteni az arra létrehozott adatbázisban
    - Android klienstől kapott kép esetén a szervernek először szegmentálnia kell a táblákat a nyers képen, majd ennek kimenetét kell továbbadnia a klasszifikáció fázisnak, ahol az InceptionV3 hálózatarchitektúra segítségével lesz megállapítva az adott tábla konkrét típusa

7. RENDSZERTERV

Ebben a fejezetben a projekt rendszertervét fogom ismertetni. Tisztázni kell, hogy milyen funkciók ellátásáért lesz felelős pontosan a szerver, az Android alkalmazás, illetve a webkliens. Szükséges megtervezni az adatbázis felépítését, mindezt úgy, hogy a szerver képes legyen a megkapott tartózkodási helyhez közeli koordinátával rendelkező táblák



7.1. ábra: Az elkészíteni kívánt rendszer vázlata

visszaadására.

A 7.1. ábrán az általam elképzelt rendszer vázlata látható, a rendszert két fő részre lehet bontani: szerver- és kliensoldalra.

Szerveroldal feladata:

- adatbázis kezelése, módosítása POST és GET kérésekkel
- táblaszegmentálás és táblaklasszifikálás, abban az esetben, ha a felismerés fázisa sikeres volt, akkor annak rögzítése az adatbázisban

Android kliens feladata:

- térkép megjelenítése
- táblák megjelenítése a felhasználó számára
- másodpercenként képtovábbítás a szerver felé a megfelelő koordinátákkal és orientációs adatokkal együtt
- navigálás a felhasználó által kiválasztott ponthoz

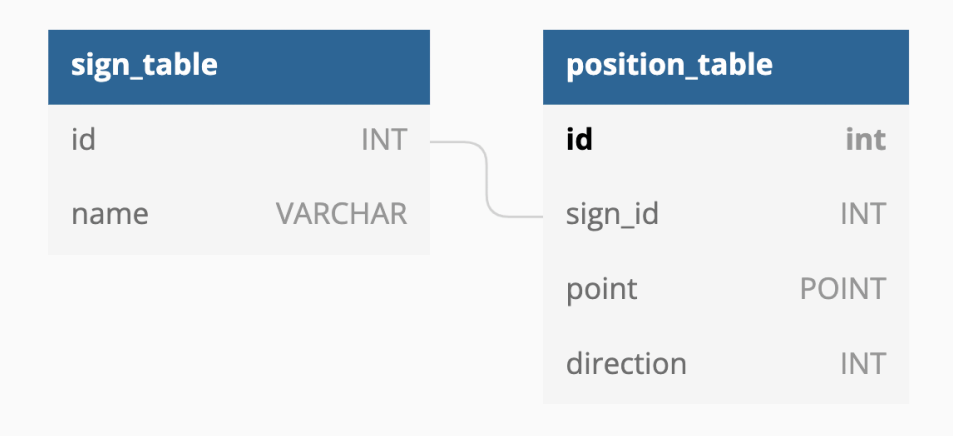
Webkliens feladata:

- térkép megjelenítése
- táblák megjelenítése a térképen
- táblák rögzítése az adatbázisban a felhasználó által megadott adatok alapján

Ahogy a felsorolásban is olvasható, a kép feldolgozásáért nem a kliens lesz a felelős, hanem a szerver. Döntésem mögött legfőképp a teljesítmény kérdése áll. Véleményem szerint nagyon sok erőforrást használt volna fel az, ha a telefonos applikációban helyezem el a neurális hálózatokat és ott történik meg a szegmentáció, illetve a klasszifikáció. Mivel nem kritérium, hogy valós időben a telefon meg is jelenítse az adott táblát rögzítésmódban, így nem gondoltam jó ötletnek kizárni a gyengébb hardverrel rendelkező telefonokat. Nem elhanyagolható előny, hogy így a rögzítés teljesen platformfüggetlen, tehát továbbfejlesztési lehetőség, hogy más platformokon is, például iOS-en működjön a program ezen része is.

7.1. ADATBÁZISTERV

A webes felületen, illetve a mobilalkamazásban történő táblamegjelenítés és táblarögzítés miatt mindenképpen szükségem lesz egy adatbázisra, mely kettő táblát fog tartalmazni. Az első tábla (sign_table) fogja tartalmazni az összes lehetséges táblát, illetve a hozzájuk tartozó id-t. Az adatbázisban megtalálható táblák listája teljes egészében megegyezik a GTSRB adatbázisában szereplő táblák listájával. A második táblában lesz a rögzített tábla id-ja, az a koordináta (point), ahol elhelyezkedik a térképen, illetve a tábla iránya, amerre a tábla néz, ez a későbbiekben az Android kliensen való megjelenítésnél lesz hasznos.



7.2. ábra: Ábra az adatbázis felépítéséről

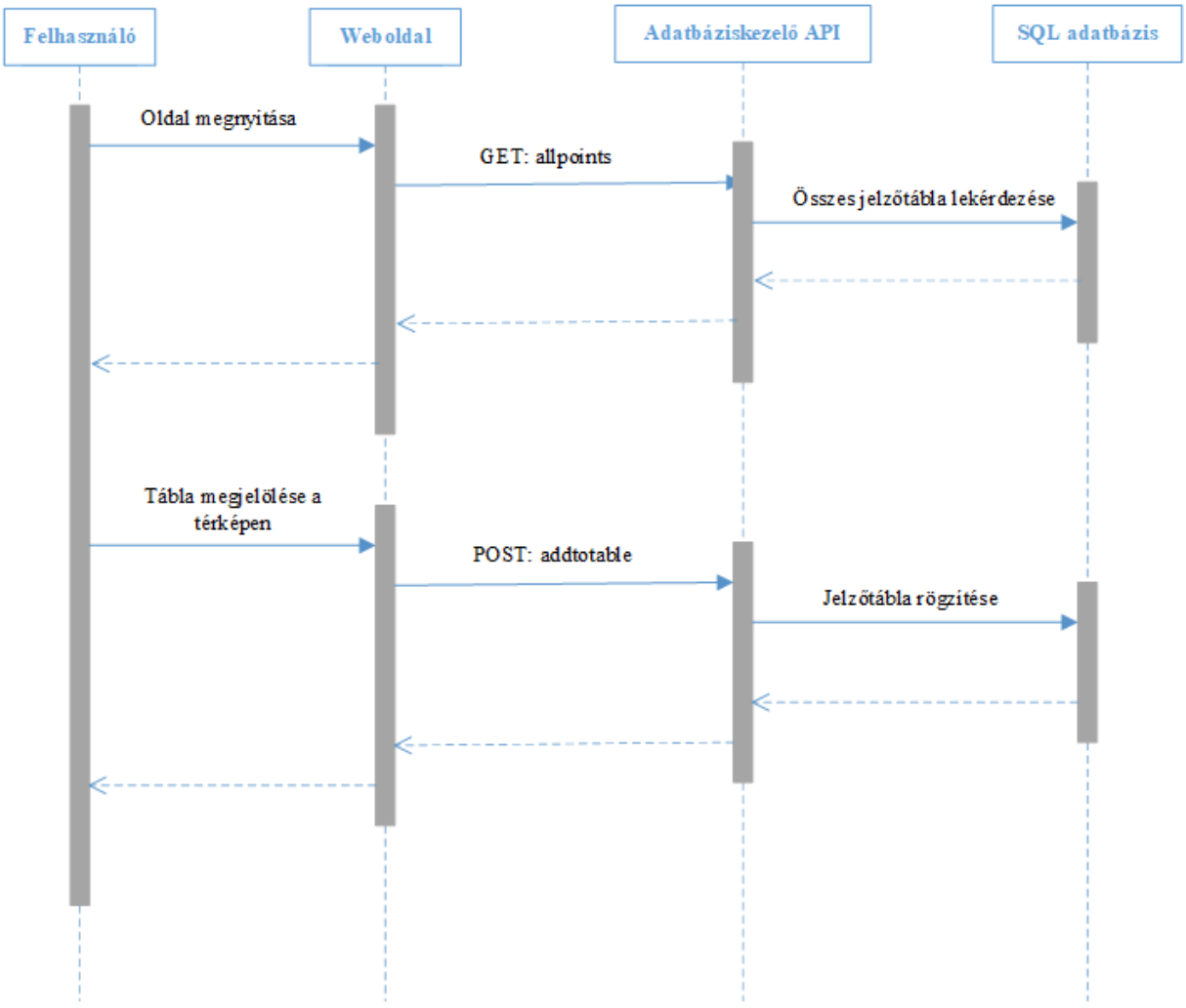
A MySQL adatbázis adta lehetőségeket kihasználva, szándékosan használom a "POINT" típust a position_table táblámban, mivel ebben a típusban egyszerre tudom eltárolni a szélességi, illetve a hosszúsági fokot. Erre a típusra leginkább azért van szükségem, mert az Androidos alkalmazásban nem szeretném megjeleníteni az összes jelzőtáblát egyszerre, ami az adatbázisban megtalálható, ennek segítségével viszont létre fogok tudni hozni olyan gyorsan lefutó lekérdezést, ami például csak a tíz legközelebbi táblát adja vissza válaszként, ezzel csökkentve azt az adatmennyiséget, amit a szervernek közölnie kell a klienssel, illetve amit meg kell jelenítenie annak.

7.2. A RENDSZER KOMMUNIKÁCIÓJA

Ebben az alfejezetben a rendszer komponensei közötti kommunikációról lesz szó, tehát arról, hogy a kérések és a válaszok milyen utat járnak be a rendszer használata során.

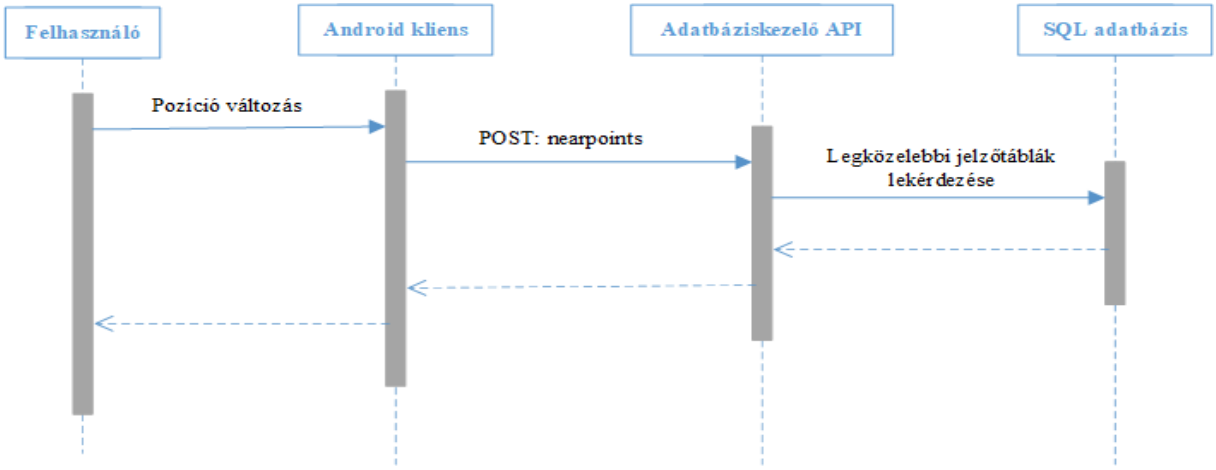
A weboldalon a manuális rögzítés, illetve a rögzített táblák megjelenítése érhető el. Ezeknek a funkcióknak a kommunikációját szemlélteti 7.3 ábrán látható szekvencia diagram. Az ábráról leolvasható, hogy minden megnyitásnál, tehát az oldal újratöltése esetén is egy GET kérést kap a szerver, aminek hatására lekéri az adatbázis összes tábláját és ezt adja vissza válaszként a böngészőnek.

A térképen történő tábla megjelölése esetén a böngésző egy POST kérést küld a tábla azonosítójával, illetve a táblához tartozó koordinátákkal. A szerver a megkapott adatokat rögzíti az adatbázisban, így abban az esetben, ha a felhasználó újratölti az oldalt azonnal látni fogja az aktuálisan rögzített táblát.



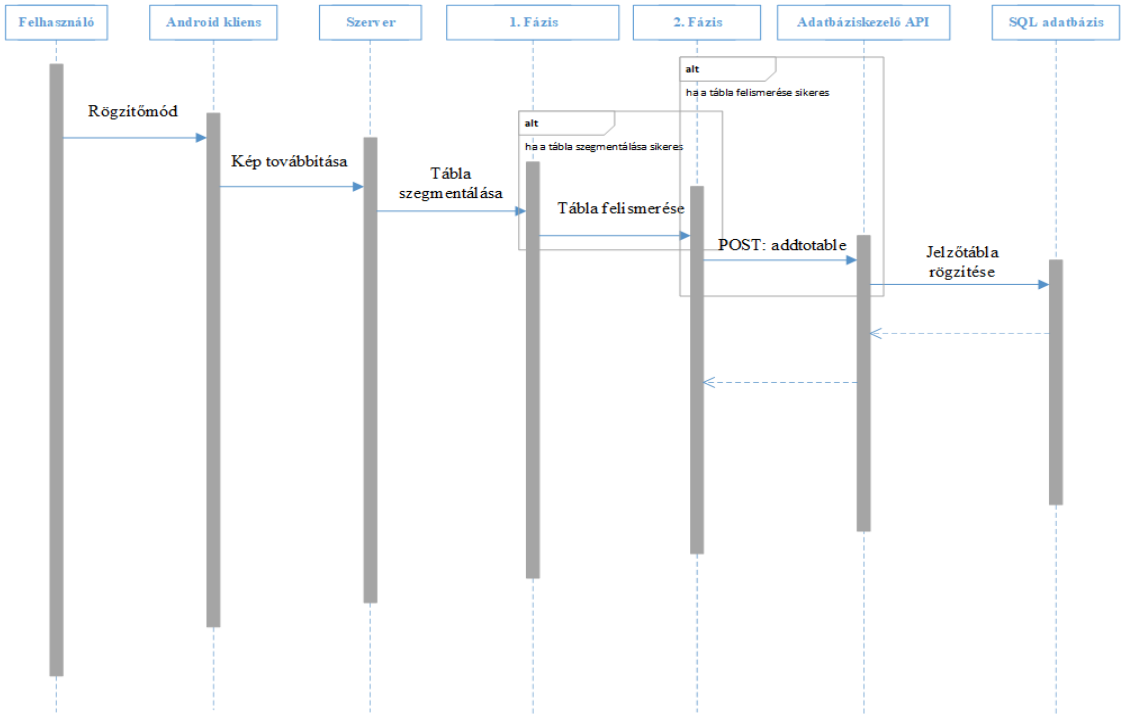
7.3. ábra: Szekvencia diagram a webkliensről

Az Android kliens esetében a jelzőtáblák lekérdezésének szekvencia diagramja látható a 7.4 ábrán. A probléma megközelítése nagyon hasonló a weboldalon történő összes tábla megjelenítéséhez. A különbség a kérés küldés gyakoriságában, illetve a visszakapott táblák számában van. Mobilkliens esetén akkor kerül egy kérés elküldésre, ha a telefon pozíciója megváltozott. Ekkor a szerver lekéri az adott pozícióhoz tíz legközelebbi táblát és ezt adja válaszul a kérésre, azért nem az összes táblát kapja a kliens válaszul, mert az nagyon leterhelné a kliens- és a szerveroldalt egyaránt.



7.4 ábra: Android kliens szekvencia diagram a közeli táblák lekéréséről

A mobilalkalmazás másik fontos funkciója a képtovábbítás, amire azért van szükség, hogy a szerveren megtörténhessen a 7.5. ábrán látható tábla szegmentálása, majd annak



7.5. ábra: Android kliens szekvencia diagram a rögzítőmódról

klasszifikálása. A 7.5 ábrán látható szekvencia diagram mutatja be ennek a funkciónak a működését. Rögzítőmódban a felhasználónak semmi dolga nincsen, a telefon automatikusan továbbítja a képkockákat a szerver felé. A szerver azonnal továbbadja a neurális hálózat részére a képet, amin abban az esetben, ha hálózat talál táblát, akkor továbbadja a következő neurális hálónak, majd az eredmény rögzítésre kerül a megfelelő adatbázistáblában.

8. IMPLEMENTÁLÁS

Ebben a fejezetben a szerveroldal és a kliensoldalak megvalósításának lépéseit fogom kifejteni, szemléltesés eszközeként legfőképp ábrákat, diagramokat és táblázatokat fogok használni.

8.1. A SZERVEROLDAL MEGVALÓSÍTÁSA

A szerveroldal feladata a kliensek kéréseinek a kiszolgálása, melyeket többnyire adatbáziskezelési műveletekkel valósít meg, illetve a jelzőtáblák szegmentálása és klasszifikálása, amit neurális hálózatok segítségével valósít meg.

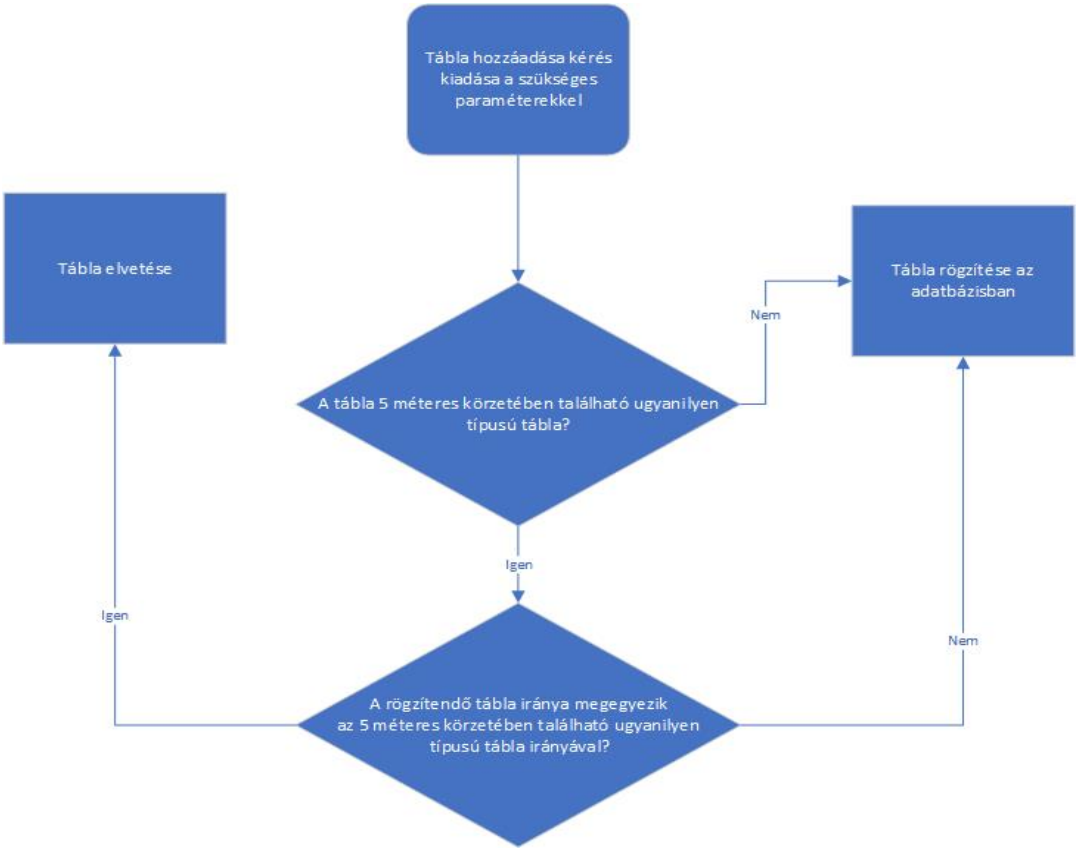
Kérés neve	Melyik kliensen elérhető
addtotable	webkliens
allpoints	webkliens
nearpoints	Android
upload	Android

8.1. táblázat: Táblázat a szerveroldalról elérhető kérésekről

8.1.1. TÁBLA HOZZÁADÁSA WEBEN KERESZTÜL

Táblát hozzáadni az adatbázishoz az */addtotable* kérés segítségével lehet a webes felületen. Ehhez a kéréshez a szervernek három darab paraméterre van szüksége a felhasználótól. Az első paraméter a táblaazonosító**,** a második a koordináta, ahol a tábla elhelyezkedik, a harmadik paraméter pedig a szögfok, ami azt adja meg, hogy a tábla pontosan milyen irányba néz. A webkliensen a felhasználónak a grafikus felületen lehetősége van kiválasztani a konkrét táblának a típusát, elhelyezni azt a térképen és úgy forgatni, ahogy a valóságban is áll az adott tábla. Szögfok megadására azért van szükség, mert előfordulhattak volna olyan esetek, ahol a felhasználónak olyan táblákat is megjelenített volna a rendszer, amik az ő irányával ellentétesen állnak, így viszont ez a probléma kiküszöbölésre került.

Táblarögzítés esetén a szerver először felveszi a kapcsolatot az adatbázissal, majd tesz egy elővizsgálatot arra, hogy tizenöt méteren belül van-e olyan tábla, aminek a szögfoka közel azonos az éppen rögzítendő táblának az orientációjával. A vizsgálat menetét szemlélteti a 8.2. ábra. Erre amiatt van szükség, mert így nem lesz lehetőség sem véletlenül sem szándékosan rögzíteni ugyanolyan táblákat azonos paraméterekkel.



8.2. ábra: A tábla hozzáadás funkció folyamatábrája

8.1.2. ÖSSZES TÁBLA LEKÉRDEZÉSE WEBEN KERESZTÜL

Az */allpoints* kérésre a szerver lekérdezi az adatbázisból az összes táblát, majd válaszul visszaadja az összes rögzített tábla koordinátáját, illetve a hozzájuk tartozó azonosítót. A webkliensen a megkapott adatok segítségével van megjelenítve az összes tábla a megfelelő helyen.


```
{
    "latitude": 47.46245624608267,
    "longitude": 18.962147580439137,
    "sign_id": 14,
    "direction": 324
},
{
    "latitude": 47.45513415988984,
    "longitude": 19.143658943876176,
    "sign_id": 14,
    "direction": 292
},
```


8.3. ábra: Példa egy szerver által küldött válaszra JSON formátumban

A kérés beérkezésekor a szerver először felveszi a kapcsolatot az adatbázissal, majd megtörténik a lekérdezés. A választ a szerver JSON formátumban közli a kliens számára.

8.1.3. KÖZELI PONTOK LEKÉRDEZÉSE ANDROID KLIENSEN

A */nearpoints* kérés bemeneti paramétere a felhasználó koordinátája, a rendszer válasza pedig maximum 50 darab tábla, távolság szerint sorba rendezve. Egy táblához tartozik a koordinátája, a távolsága az adott ponttól, a tábla azonosítója és a tábla szögfoka, ami megadja, hogy melyik irányba néz az adott tábla, a szerver a választ ebben az esetben is JSON-ben fogja közölni.

8.1.4. KÉPFELTÖLTÉS ANDROID KLIENS HASZNÁLATÁVAL

A feltöltés a */photoupload* kérés segítségével történik. Minden feltöltött kép szélessége 480 pixel, míg magassága 640 pixel. Ezt a méretet amiatt választottam, mert egyrészt ez már a neurális hálózatoknak is megfelelő nagyság, másrészt a klienstől a szerverig való átvitel gyorsan meg tud történni, mivel a képek várható mérete nem fogja meghaladni a 100 kilobájtot. A kérés feldolgozásakor az erre kijelölt mappába kerül mentésre az adott kép, minden kép mellé tartozik egy szöveges fájl, ami tartalmazza a táblarögzítéséhez szükséges egyéb információkat is. Ezek az egyéb információk, a manuális rögzítésnél említett paraméterekhez hasonlók, amik a koordináta és a szögfok.

8.1.5. FELTÖLTÖTT KÉP SZEGMENTÁLÁSA

A képfeltöltés után a szegmentálás fázisa következik. Ennek a fázisnak dedikáltan külön szál van definiálva a szerveroldalon, így hosszabb feldolgozási idő esetén sem fogja megakasztani a szerver felé küldött kérések feldolgozását.

A képeken látható táblák szegmentálása a DarkNet Yolov4 (továbbiakban: Yolo) neurális hálózati architektúra segítségével valósul meg. A Yolo hálózati architektúra nagyon hatékonyan tudja megoldani a különböző objektum detektálással kapcsolatos feladatokat, számos objektumot tud érzékelni módosítatlan állapotában is, azonban az általam választott probléma megoldásra külső betanítási adathalmaz nélkül nem volt képes, emiatt szükségem volt egy adathalmazra, amivel be tudtam tanítani a hálózatot. [29]

A neurális hálózat áttanításához szükségem volt egy adathalmazra, ami olyan képeket tartalmaz, amiken láthatóak közlekedési jelzőtáblák és ezekhez a képekhez tartozik valamilyen dokumentum, ahol a képeken látható táblák pozíciója van leírva. Adathalmazom forrása a kaggle.com weboldalon található Traffic Signs Dataset in YOLO format nevű adathalmaz volt. Ebben több kategóriára voltak osztva a táblák. mivel a rendszer ezen szakaszában csak a szegmentálás megoldása a cél, a több kategóriát összeolvasztottam, így egy osztályba került besorolásra az összes tábla, amik a képeken voltak láthatóak. [30]

Az adathalmaz képeihez tartozó annotációs fájlok a Yolo architektúrának megfelelő formában voltak, így csak az osztálymegjelöléseket kellett módosítanom az említett fájlokban.

A felhasznált adathalmaz betanításra szánt képeinek száma megközelítőleg 800 darab volt. A hálózat már ezzel is jól szegmentálta a táblákat, azonban voltak olyan táblatípusok, amiket kevésbé sikeresen tudott szegmentálni, ez köszönhető annak, hogy abból a fajta táblából kevesebb volt az általam használt adathalmazban. Ennek hatására készítettem egy segédalkalmazást, amiben meg tudom nyitni az általam készített képeket vagy olyan képeket, amikhez nem tartozik semmilyen annotációs fájl, majd ezeken a képeken ki tudom jelölni a táblákat, ezután a program a Yolo hálózatnak megfelelő annotációs fájlt készít a képekről, így az eredetileg nem annotált képeket is fel tudom használni a hálózat tanítására.

A tanítás megkezdése előtt az adathalmazt el kellett helyeznem a megfelelő almappákban, külön a képeket és külön a képekhez tartozó annotációs fájlokat. Miután ez megtörtént a Yolo konfigurációs fájljában definiálnom kellett a hozzáadott új osztályok számát, illetve azok neveit. Esetemben ennél a lépésnél még csak a táblákat szerettem volna szegmentálni a képekről, emiatt csak egy osztályt definiáltam. A betanítás nagyon időigényes folyamat, abban az esetben, ha processzorral végeztem a betanítást megközelítőleg 10 óra alatt végzett a hálózat, míg videókártyát használva, ami esetemben egy Nvidia GTX 1070Ti volt, nagyjából 90 percet vett igénybe ugyanez a művelet.

A szegmentálás megkezdése előtt betöltöm az architektúrába a tanításom súlyfájlját, majd megkezdődik a szegmentálás folyamata. A Yolo architektúra alapértelmezetten bekeretezi az adott tárgyat, amit megtalált, viszont nekem szükségem volt külön fájlban a bekeretezett képek tartalmára, emiatt létrehoztam egy metódust, ami a hálózattól kapott adatok alapján ki tudja vágni a szükséges részt. A program ezen részének kimenete, amennyiben található tábla a szegmentálni kívánt képen, lehet egy kép vagy több kép, abban az esetben, ha több tábla is látható a képen. Több esetben megfigyeltem, hogy a kivágott képek téglalap alakúak lettek, zömében ezeken a képeken nem is szerepelt semmilyen tábla, emiatt úgy módosítottam a kivágás menetét, hogy csak akkor történjen meg a kivágott kép mentése, ha egy adott arányszámnak megfelelnek az újonnan létrejövő kép oldalai. A már szegmentált képeket az erre kijelölt mappába menti a rendszer, majd ezután következik a klasszifikáció fázisa.

8.1.6. A SZEGMENTÁLT KÉP KLASSZIFIKÁLÁSA

Szegmentálás utáni állapotban a képnek már csak a táblát szabad tartalmaznia, a feladat, hogy kiderüljön a táblának a pontos típusa. Ennek a problémának a megoldására több, már létező neurális hálózati architektúra is szóba jöhet, éppen ezért első lépésként kipróbáltam a különböző lehetőségeket, ezek eredményeit összehasonlítottam, majd az eredmények tudatában döntöttem a pontos konfigurációról.

A feladathoz a GTSRB adathalmazát felhasználva teszteltem összesen három hálózati architektúrát. Az adathalmaz 39218 képpel rendelkezik, amit úgy osztottam fel, hogy a validációs és a teszt adathalmaz a képek 10-10%-át kapta, ami megközelítőleg 3900 darab fényképet jelent, a maradék 80% pedig a tanításhoz felhasznált képek aránya. Az osztályok számából adódóan 43 osztályú klasszifikációt kellett megvalósítani.

Három népszerű konvolúciós neurális hálózati architektúrát választottam, amik kifejezetten javasoltak ilyen típusú problémák megoldására. Ezek a GoogleNet InceptionV3, a VGG16 és a ResNet50. A feladatom során áttanítást (transfer learninget) használtam a hálózatoknál, amit kifejezetten akkor érdemes megtenni, amikor egy meglévő architektúrát szeretnénk felhasználni egy olyan probléma megoldására, amire alapból nincs felkészítve.

A módosítások, amiket alkalmaztam a kipróbálás során:

- Dropout használata: abban az esetben, ha használom, akkor 0,2 értéket adok neki.
- Tanulási ráta módosítása: két értéket használtam 0,001 és 0,0001.
- Kép méretének módosítása: hálózat architektúránként csak egy esetben változtattam a 224 pixelről, 299 pixelre.
- Batch méretének módosítása: a 16-os és a 32-es értékeket adtam meg.
- Optimizer módosítása: az esetek többségében RMSpropot használtam, emellett kipróbáltam az Adam-et, az SGD-t és a Adagrad-ot is.

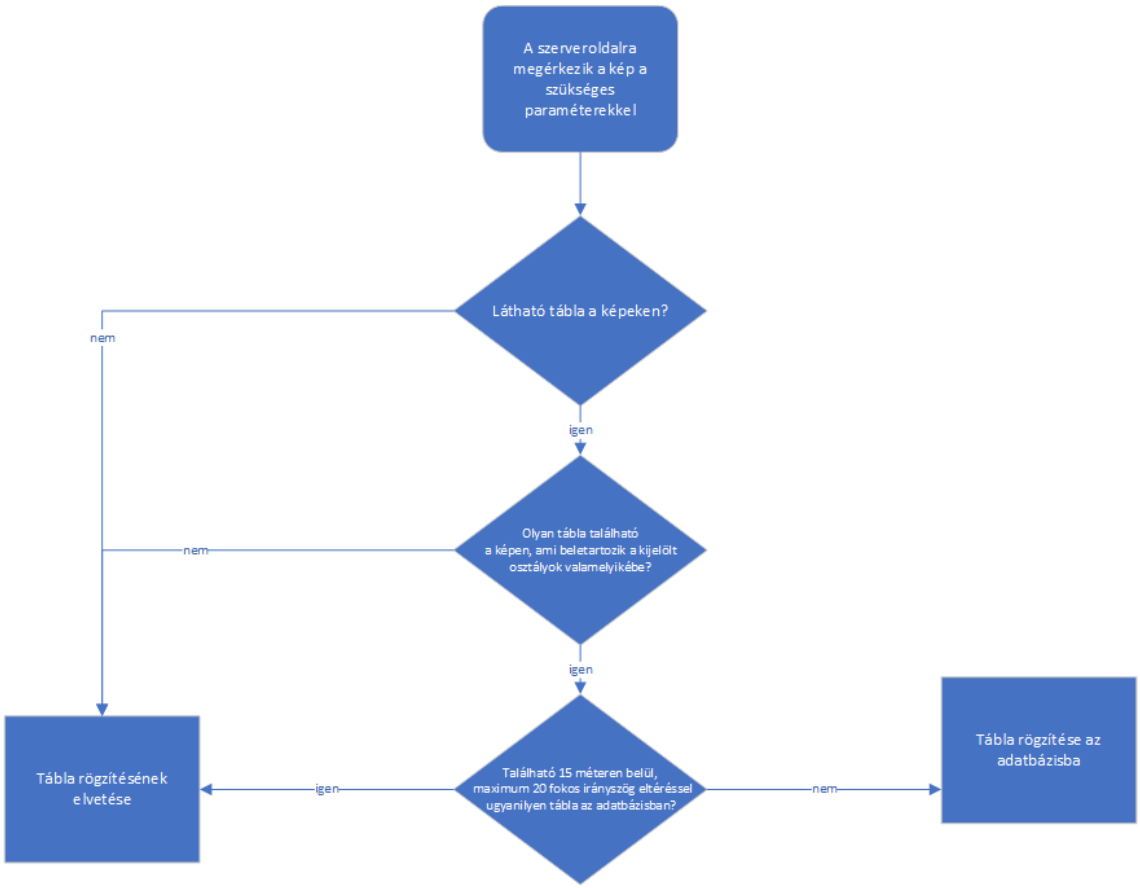
A tanítás során felhasználtam az EarlyStopping funkciót, így az alapértelmezett értékként beállított 100 epochot csak abban az esetben közelítette meg a hálózat, amikor az SGD optimizert használtam, a többi esetben átlagosan 10 körüli epoch számig jutott a tanítás.

Az eredményekről elmondható, hogy az InceptionV3 szerepelt a legjobban a tanítások során, ezt követte a ResNet50, majd szorosan következett a VGG16 architektúra. Az adatokból kiolvasható érdekesség, hogy a VGG16 legjobb eredménye (81,36%) az InceptionV3 architektúra legrosszabb eredményét előzi csak meg, tehát elmondható, hogy az InceptionV3 erre a feladatra a paraméterektől függetlenül kiegyensúlyozottan jól teljesített. A VGG16 és a ResNet50 architektúra esetében pedig az állapítható meg, hogy csak a megfelelő paraméterekkel képes viszonylag jó eredménnyel teljesíteni.

A 8.4. táblázatban látható az összes általam elvégzett tanítás és a jellemzőbb paraméterek. Mindegyik tanítás során első aktivációs függvénynek a Relu-t, második aktivációs függvénynek pedig a Sigmoid függvényt használtam. A táblázatban zölddel jelölt sorok jelzik az adott architektúra legjobb eredményét, a pirossal jelölt sor pedig azt jelzi, ha a tanítás nem volt sikeres, ennek oka mindkét esetben a kevés memória volt. Ahogy az már kiderült az InceptionV3 architektúra volt képes a legjobb eredményre, azonban látható, hogy a tanítási idő nagyon megnövekedett a legjobb tanítás esetén.

Sorszám	Architektúra	Dropout	Learning rate	Batch méret	Optimizer	Tanítási idő	val (%)	test (%)
1	InceptionV3	-	0,001	16	RMSprop	1321 sec	91,3	91,6
2	InceptionV3	-	0,001	16	RMSprop	1297 sec	91,0	
3	InceptionV3	0,2	0,001	16	RMSprop	536 sec	88,6	88,9
4	InceptionV3	-	0,001	32	RMSprop	452 sec	83,1	82,9
5	InceptionV3	0,2	0,001	32	RMSprop	336 sec	89,4	86,7
6	InceptionV3	-	0,0001	16	RMSprop	709 sec	90,7	91,2
7	InceptionV3	-	0,0001	32	RMSprop	342 sec	93,5	91,7
8	InceptionV3	0,2	0,0001	32	RMSprop	592 sec	91,4	92,3
9	InceptionV3	0,2	0,0001	32	Adam	420 sec	91,1	87,86
10	InceptionV3	-	0,0001	32	Adam	415 sec	89,69	70,46
11	InceptionV3	0,2	0,0001	32	SGD	5840 sec	92,9	92,4
12	InceptionV3	0,2	0,0001	32	Adagrad	5520 sec	93,8	92,8
13	VGG16	-	0,001	16	RMSprop	432 sec	74,5	68,6
14	VGG16	-	0,001	16	RMSprop	892 sec	71,9	75,3
15	VGG16	0,2	0,001	16	RMSprop	680 sec	77,0	68,9
16	VGG16	-	0,001	32	RMSprop	360 sec	74,5	69,3
17	VGG16	0,2	0,001	32	RMSprop	362 sec	76,7	75,2
18	VGG16	-	0,0001	16	RMSprop	429 sec	77	81,3
19	VGG16	-	0,0001	32	RMSprop	361 sec	76,8	67,7
20	VGG16	0,2	0,0001	32	RMSprop	370 sec	76,7	77,7
21	VGG16	0,2	0,0001	32	Adam	450 sec	74,0	47,2
22	VGG16	-	0,0001	32	Adam	361 sec	75,6	78,9
23	VGG16	0,2	0,0001	32	SGD	1324 sec	72,2	76,1
24	VGG16	0,2	0,0001	32	Adagrad	4470 sec	71,5	65,2
25	ResNet50	-	0,001	16	RMSprop	690 sec	75,3	68,1
26	ResNet50	-	0,001	16	RMSprop	1297 sec	77,2	
27	ResNet50	0,2	0,001	16	RMSprop	676 sec	76,9	77,9
28	ResNet50	-	0,001	32	RMSprop	502 sec	73,9	67,1
29	ResNet50	0,2	0,001	32	RMSprop	501 sec	78,1	21,6
30	ResNet50	-	0,0001	16	RMSprop	695 sec	78,0	42,9
31	ResNet50	-	0,0001	32	RMSprop	672 sec	79,9	41,1
32	ResNet50	0,2	0,0001	32	RMSprop	664 sec	80,8	62,4
33	ResNet50	0,2	0,0001	32	Adam	430 sec	76,9	72,3
34	ResNet50		0,0001	32	Adam	424 sec	77,9	60,0
35	ResNet50	0,2	0,0001	32	SGD	1432 sec	82,0	83,3
36	ResNet50	0,2	0,0001	32	Adagrad	973 sec	79,8	82,7

8.4. táblázat: Összefoglaló táblázat az elvégzett betanításokról



8.5. ábra: Beküldött képről történő táblarögzítés folyamatábrája

Az eredményeket figyelembe véve az InceptionV3 legjobb eredménnyel rendelkező konfigurációját választottam a rendszerem klasszifikációért felelős részéhez. A Yolo hálózatból kikerült kivágott tábla a bemenete ennek a hálózatnak. A kép ebben a fázisban előfeldolgozásra kerül a beépített InceptionV3-hoz tartozó előfeldolgozási függvénnyel, majd erre a képre történik a predikció. Ezután a rendszer rögzíti az adatbázisban a tábla azonosítóját a hozzátartozó paraméterekkel együtt. A teljes folyamatot a 8.5. ábra szemlélteti. A 43 osztálynak számát azonban csökkentenem kellett, mert az Android kliensen történő megjelenítés ennyi osztállyal nagyon átláthatatlan eredményt szült volna. A 43 osztályból 12 darab osztályt hagytam meg, így csak akkor történik meg a táblák rögzítése az adatbázisban, ha ezek közül az osztályok közül tartozik valamelyikbe az adott tábla. A 8.6. táblázatban látható a konkrét 12 darab táblatípus, melyek megjelenítésre kerülhetnek Android és webkliensen egyaránt.

Sorszám	Megnevezés
1	Sebesség 30kmh
2	Sebesség 50kmh
3	Előzni tilos
4	Főút
5	Elsőbbség adás kötelező
6	Stop
7	Két irányú behajtani tilos
8	Behajtani tilos
9	Gyalogos átkelő
10	Kötelező haladási irány: egyenes
11	Kötelező haladási irány: egyenes vagy jobbra
12	Kötelező haladási irány: egyenes vagy jobbra

8.6. táblázat: Táblázat a 12 darab megjelenítendő táblatípusról

8.2. ANDROID OPERÁCIÓS RENDSZERREL RENDELKEZŐ KLIENSOLDAL MEGVALÓSÍTÁSA

Az Android alkalmazás két főfunkcióval rendelkezik, egyrészt a táblarögzítő funkcióval, másrészt a navigáció funkcióval. A felhasználónak, attól függetlenül, hogy melyik funkciót szeretné használni szüksége lesz engedélyezni az applikáció kamerahasználatát, GPS használatát, illetve elengedhetetlen a folyamatos internetkapcsolat az alkalmazás használata során.

8.2.1. NAVIGÁCIÓ MEGVALÓSÍTÁSA MAPBOX SDK SEGÍTSÉGÉVEL

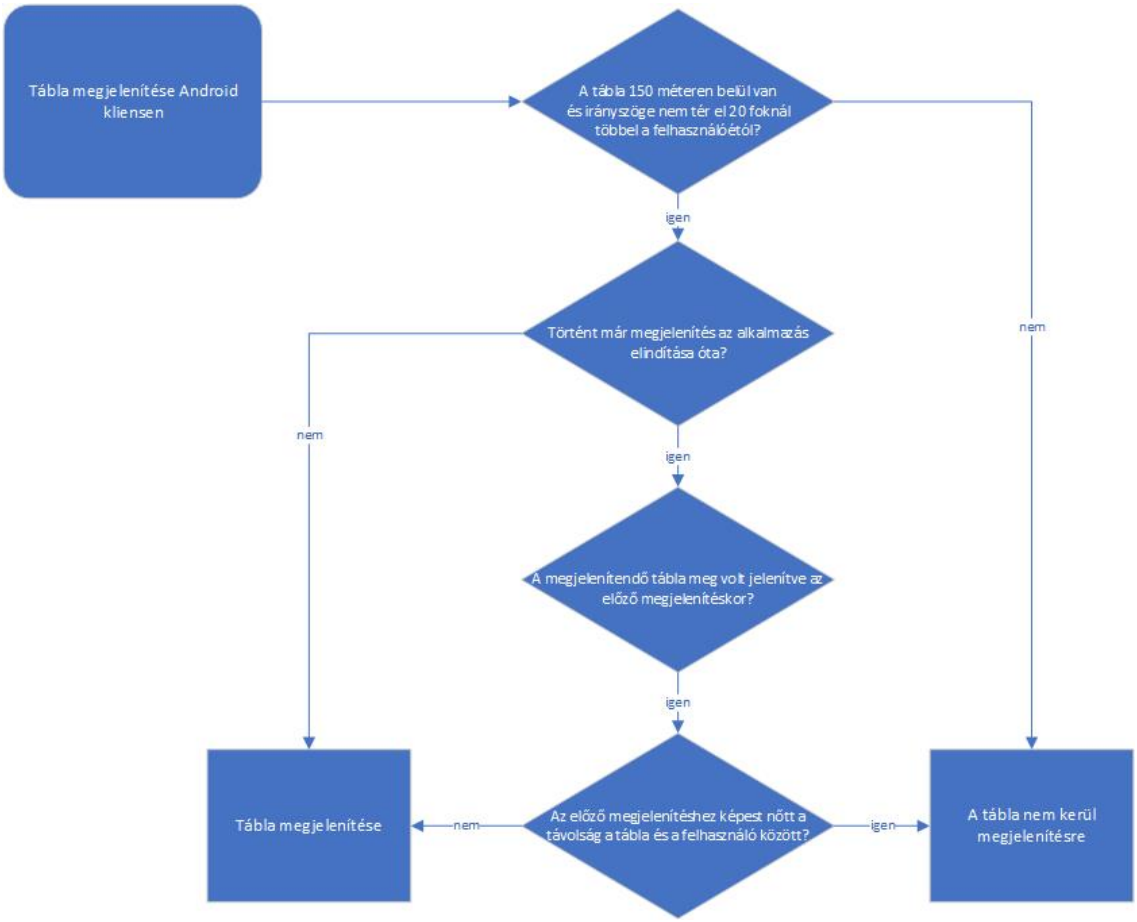
A térképpel kapcsolatos funkciókhoz összesen kettő képernyő tartozik. Az első képernyő tartalmazza a térképet, illetve egy keresőt. Ez a képernyő használható abban az esetben is, ha a felhasználó nem kíván navigációt igénybe venni, csak a térképet nézné. Ezen a képernyőn a felhasználó felülnézetből követheti haladását, abban az esetben, ha szüksége van navigációra, a képernyő alján található keresőmezőbe, beírhatja a célállomást. A találat kiválasztása után megnyílik a navigáció képernyője. Ezen a képernyőn az általános GPS nézetben látja a felhasználó a térképet, illetve a kirajzolt utat, amerre majd haladnia kell. A képernyő alján látható a még hátralévő út és idő. A térkép kirajzolásához, illetve az navigációs funkciók implementálásához a hivatalos MapBox SDK, illetve Directions API leírásokat használtam fel. [28][31]

Mindkét képernyő bal oldalán helyezkedik el egy négy jelzőtáblát megjeleníteni képes jelződoboz, aminek tartalmáért a szerver a felelős. Az Android kliens bizonyos

időközönként kérést küld a szerver felé, amiben az adott koordinátához legközelebbi táblákat kéri. A szerver távolság szerint növekvő sorrendben visszaadja a táblákat a kliensnek. A válasz formája JSON, ezért a kliens első feladata ennek a válasznak a feldolgozása. A kapott táblákat egy tömbbe helyezem, majd ebből a tömbből ismétlés nélkül feltöltöm a jelzősávot. Ügyelek arra, hogy a táblák távolsága az aktuális ponttól ne legyen több 150 méternél, illetve arra is, hogy a kliens aktuális szögfoka/orientációja közel azonos legyen az adott tábláéval. A szögfok értékét az egyik, telefonba épített szenzor segítségével állapítom meg, a készülék ezt a szenzort használja a gyári iránytű alkalmazás használata esetén is.

A jelzősáv frissítése koordináta változásonként történik. A funkció implementálása közben találkoztam egy olyan problémával, amire egészen addig nem gondoltam. Korábbi fejezetrészben már kifejtettem, hogy a szögfok bevezetésével megoldottam egy olyan problémát, hogy azonos helyen egymással szemben álló táblákat egyszerre biztos nem fog bejelezni az alkalmazás, viszont az aktuális probléma az, hogyha elhaladok egy tábla mellett, ami jelezve van a jelzősávban, akkor ennek a táblának a jelzése csak akkor fog megszűnni, ha vannak más táblák, amiknek a távolsága kisebb, mint az említett tábla távolsága, ami mellett elhaladtam. Összefoglalva, a távolság értéke független attól, hogy a tábla fizikailag előttem vagy mögöttem van. Ennek a problémának a megoldására bevezettem egy tömböt, ami minden esetben az előző kijelzés elemeit tartalmazza.

Kijelzéskor vizsgálom, hogy az adott azonosítóval rendelkező tábla benne van-e az újonnan létrehozott tömbben, ha igen akkor megvizsgálom, hogy az előző állapotban mekkora volt a kliens távolsága a táblától, illetve az aktuális távolsága mennyi a táblától, ha az aktuális kisebb érték, mint az előző, akkor közeledik a kliens, tehát meg kell jeleníteni a táblát, ha nagyobb az érték, akkor távolodik a kliens, tehát nem kell megjeleníteni az adott táblát.



8.7. ábra: Jelzőtábla megjelenítésének folyamatábrája Android kliens esetén

8.2.2. KÉPKÜLDÉS MEGVALÓSÍTÁSA

A képküldés funkció az alkalmazásban kétféle módon elérhető. Abban az esetben, ha a felhasználó nem szeretné az applikáció többi funkcióját párhuzamosan használni, lehetősége van az alkalmazás kezdőképernyőjén kiválasztani a rögzítő módot, ebben a módban látható a kamera élőképe, miközben a háttérben továbbításra kerülnek a képkockák. Abban az esetben, ha a felhasználó navigáció használata közben szeretné használni a rögzítőmódot erre is van lehetősége. Ekkor a jobb felső sarokban található legördülő menüre kell nyomnia a felhasználónak és kiválasztani a rögzítés menüpontját, ilyenkor az élőkép megjelenítése nélkül történik a képtovábbítás a szerver felé.

A képkockák elkészítéséhez valamilyen erre készült könyvtárra volt szükségem. Az Android operációs rendszerbe beépített Camera2 nevű könyvtár teljesen megfelelt az igényeimnek, ezért ezt használtam az funkció implementálása során. Célom az volt, hogy a képkészítés úgy történhessen meg, hogy azt eltárolás nélkül is továbbítani lehessen a szerver felé. Szerencsére a beépített kamera könyvtár erre lehetőséget ad, így a képet rövid időre egy bájt tömbbe tárolom el. A képek továbbításának az időtartamát viszonylag alacsonyan akartam tartani, emiatt a képek felbontása minden esetben 640x480 pixel.

Miután a tömbbe belekerült az aktuális képkocka, meghívásra kerül a továbbításért felelős metódus, melynek bemeneti paraméterei egy bájt tömb, koordináta adatok és az orientáció. A szerver felé küldött kérést az Apache könyvtárainak segítségével állítom össze, majd ennek segítségével küldöm is el.

A képnek az elkészítése, illetve annak továbbítása a szerver felé erőforrás- és időigényes művelet, emiatt a főszálon való futtatása az adott kódrésznek nem lenne felhasználói élmény terén kifogástalan, ebből adódóan ezeket a műveleteket egy külön szálon végzi az alkalmazás.

8.3. WEBKLIENS MEGVALÓSÍTÁSA

A rendszer webkliense egy egyszerű weboldal, ami egyrészt kommunikál a MapBox szervereivel, illetve az általam létrehozott szerverrel. Az oldal két főfunkciója a táblarögzítés, illetve a már rögzített táblák megjelenítése.

Táblarögzítéskor a felhasználónak a térképen pontosan ki kell jelölnie azt a pontot, ahova a rögzíteni szeretné az adott táblát, majd be kell állítania azt, hogy ez a tábla melyik irányba néz, végül ki kell választania a tábla típusát. A küldés gombra kattintva a kérést a kliens elküldi a szerver felé, ami ezután feldolgozza azt.

A megjelenítéskor a felhasználónak lehetősége van kiválasztani, hogy mely táblák legyenek megjelenítve a térképen, erre amiatt volt szükség, hogy átlátható maradjon a kijelzés sok rögzített tábla esetén.

9. TESZTELÉS

Ebben a fejezetben az általam megalkotott rendszer tesztelésének lépéseit, illetve azok eredményeit fogom ismertetni. Az alkalmazás tesztelése során egy Redmi Note 9 Pro okostelefont használtam, a szerverkörnyezet az asztali számítógépemen futott, annak érdekében, hogy ne csak belső hálózatról érjem el a szerver funkcióit, az **ngrok** nevű programot használtam, ami lehetővé teszi a localhost külső elérését.

9.1. ANDROID KLIENS FUNKCIÓINAK TESZTELÉSE

Először az Android kliens funkcióinak működését ellenőriztem manuális teszteléssel. Telepítés utáni első megnyitáskor az alkalmazás kéri a felhasználótól a kamera, illetve az aktuális pozíció használatára az engedélyt, az implementált funkciók azután elérhetőek, hogy ezeket a felhasználó engedélyezte.

9.1.1. NAVIGÁCIÓ TESZTELÉSE

Első teszteseteim a navigáció és a hozzátartozó alfunkciók átfogó kipróbálásából állt.

A navigációhoz tartozó kiindulási tesztesetem célja egy útvonalterv megjelenítése az Budapest, Üllői út 55 címhez. A kezdőképernyőn kiválasztom a navigáció funkciót, ekkor az alkalmazás továbbvisz a második képernyőre, ahol a térkép látható és a keresőmező. Itt a keresőbe begépelem a kívánt címet. Miután ez megtörtént a keresési találatok közül kiválasztom a számomra megfelelőt, ami ebben az esetben az első találat. A kiválasztás után az alkalmazás átvezet a következő képernyőre, ami már a navigáció képernyője. Itt először megmutatja a teljes útvonaltervet a térképen, kék színnel kijelölve és a helymeghatározó ikonra nyomva közelít rá az alkalmazás az aktuális pozícióra.

A navigáció kipróbálásához két tesztesetet hoztam létre. Az első esetében követtem azt az eredeti útvonalat, amit az alkalmazás tervezett. Az alkalmazás megfelelően működött, az alsó sávban található visszajelző sáv jól mutatta a hátralévő becsült időt, illetve a hátralévő távolságot, megérkezéskor az alkalmazás jelezte, hogy megérkeztünk a célállomáshoz. A második esetben az útvonalnak az újratervezését próbáltam ki. Bizonyos helyeken szándékosan nem követtem a navigáció által javasolt utat, az applikáció ekkor is jól működött, viszonylag gyorsan megtörtént az újratervezés, amivel együtt a hátralévő idő és távolság is frissült.

A navigáció funkcióról elmondható, hogy megfelelően működött a használat során, azonban a tesztelés közben többször úgy érzékelte az alkalmazás, mintha forgatnám a telefont, ennek tünete, hogy általában, amikor egyhelyben álltam, a térképet elkezdte forgatni az alkalmazás indokolatlanul. Ennek a kiváltó okára nem tudtam rájönni, sajnos más készülékkel nem volt lehetőségem tesztelni a funkciót. Emellett a készülék lekövetése a térképen nem minden esetben volt hibátlan, nagyobb sebesség esetén késésben volt a készülék pozíciójának a megjelenítése a valós pozíciómmal. Ezt a problémát próbáltam orvosolni, azzal, hogy a pozíció lekérésének létrehozásakor meg

kell adni egy időközt, hogy milyen gyakorisággal szeretném megkapni a telefon pozícióját. A hivatalos dokumentáció alapján, ha ezt az értéket 0-ra állítom, akkor a lehető legkevesebb idő telik el két pozíciólekérés között. Az értéket átállítva a helyzet nem javult, továbbra is tapasztalható volt késés bizonyos esetekben, ezt okozhatta a GPS jel erősségének hiánya, az internetkapcsolat minősége, de az is lehetséges opció, hogy a telefon hardvere nem volt a megfelelő a tesztelés során.

9.1.2. TÁBLAMEGJELENÍTÉS TESZTELÉSE

A táblamegjelenítés funkció két képernyőn is elérhető, azonban működés és megvalósítás szempontjából a két megjelenítés teljesen megegyezik, ezért csak az egyik képernyőn teszteltem a funkció működését. A teszteset célja annak ellenőrzése, hogy a képernyő bal oldalán elhelyezett jelzősávban helyesen jelennek-e meg a szerveroldaltól kapott táblák.

A tesztelés megkezdése előtt az üres adatbázisba felrögzítettem a webes felület segítségével az útvonalon és a környékén található táblákat. A tesztelés zökkenőmentes volt, nem volt probléma a kommunikációval a szerver és a kliens között, a jelzősávban mindig a négy legközelebbi táblát mutatta a rendszer ismétlés nélkül, abban az esetben, ha nem volt négy megjeleníthető tábla, akkor négynél kevesebb tábla volt kirajzolva a sávban. A táblák megjelenítésének a frissítési ideje, azonban nem volt tökéletes. A megjelenítési frissítés ideje a készülék pozícióváltozásához van kötve, ahogy az előző tesztesetben már kifejtettem, sokszor tapasztalható a telefon pozíciójának lassú lekövetése, aminek következménye, hogy a táblák megjelenítése esetenként több időbe telik a kelleténél. Több esetben tapasztaltam, hogyha egy kereszteződésnél lefordultam, akkor pár pillanat erejéig a kereszteződésen túl elhelyezett táblát is megjelenítette az alkalmazás, ez egyértelműen ennek a lassú lekövetésnek tudható be. Az előző tesztesetben leírtakhoz hasonlóan, ennek a hibának a kiküszöbölését ugyanazon okok miatt nem tudtam elvégezni, emiatt a funkció néha kisebb hibákkal működik, azonban ez a felhasználói élményt csak kis mértékben befolyásolja negatív irányba.

9.1.3. KÉPTOVÁBBÍTÁS TESZTELÉSE

A képek továbbításának lehetősége kétféleképen elérhető az alkalmazásban, ebben az esetben mindkét eshetőséget leteszteltem, mert működés terén van eltérés a két megvalósítás között.

Első esetben az erre a funkcióra készült képernyőn próbáltam ki a működést, ekkor meg van jelenítve az adott képkockarögzítés eredménye. A harmincperces tesztút során a telefon percenként átlagosan 45 darab képet tudott készíteni, majd továbbítani a szerver felé, ami azt jelenti, hogy ebben az esetben a rendszer 0,75 képet tud eljuttatni a szervernek másodpercenként.

A másik lehetőség ennek a funkciónak a használatára, ha navigáció közben használja a felhasználó a háttérben, ekkor értelemszerűen nincsen élő képmegjelenítés. A szintén félórás tesztút során ebben az esetben is folyamatos volt a képátvitel, azonban itt 55 darab

körül tudott képet készíteni és továbbítani a kliens. Ezt, illetve az előző eshetőséget is többször teszteltem és valóban elmondható, hogy a megjelenítés nélküli módban valamivel több képet tud továbbítani a rendszer, ami valószínűleg annak köszönhető, hogy maga a megjelenítés is időt vesz igénybe és emiatt kevesebb képet tud készíteni a készülék.

9.1.4. EGYSÉGTESZTELÉS

Az Android kliensre készült alkalmazásban három fontos funkcióra készítettem egységtesztet, hogy megbizonyosodjak arról, hogy azok az elvárt módon működnek.

Az első egységteszt a távolság kalkulációját ellenőrizte, ami elengedhetetlen funkció ahhoz, hogy megállapítsa a program, hogy mennyire van távol a felhasználó az aktuális táblától.

A második és a harmadik egységteszt is a táblák megjelenítéséhez köthető funkciók ellenőrzésére szolgált. A második egységteszt azt a funkciót ellenőrizte, ami azért felelős, hogy megállapítsa a felhasználóról, hogy az közeledik vagy távolodik az adott táblától.

A harmadik egységteszt pedig azt ellenőrizte, hogy a kliens irányszögétől függően megjelenítene egy adott táblát vagy sem.

Sorszám	Rövid leírás	Elvárt eredmény	Eredmény
1.	Két pont közötti távolság méréséért felelős funkció tesztelése	Az általam megadott két pont közötti távolság 80 méter	Sikeres
2.	A táblához a kliens 50 méterről 35 méterre csökkenti a távolságát	A rendszer megjelenítené a táblát	Sikeres
3.	A tábla és a kliens irányszögének különbsége 10 fokkal tér el	A rendszer megjelenítené a táblát	Sikeres

9.1. táblázat: Az egységtesztek eredményeinek összefoglaló táblázata

9.2. WEBKLIENS TESZTELÉSE

A webkliens tesztelését két tesztesetre lehet bontani. Az első teszteset a rögzített közúti jelzőtáblák megjelenítését ellenőrzi, a második teszteset pedig egy tábla rögzítését a felületen.

Az oldal betöltésekor alapértelmezetten egy táblát sem jelenít meg a rendszer. A felhasználónak ki kell jelölni azokat a táblatípusokat, amiket látni kíván a térképen. A

kiválasztás, illetve a megjelenítés jól és gyorsan is történik, lehetőség van egyszerre több táblatípusnak a megjelenítésére, ami szintén az elvártaknak megfelelően működik.

A táblák rögzítése szintén jól működik, abban az esetben, ha egy táblát szeretnénk rögzíteni egy olyan helyre, ahol már van ugyanilyen típusú tábla és a már rögzített, illetve a rögzítendő tábla szögfoka közel azonos, akkor nem történik meg a rögzítés, ami a megfelelő működést jelenti.

9.3. A SZERVEROLDAL ÉS A NEURÁLIS HÁLÓZATOK TESZTELÉSE

A neurális hálózatok tesztelését két részre tudtam bontani. Szükséges volt tesztelnem külön a szegmentációért és külön a klasszifikálásért felelős neurális hálózatot. Ebben az alfejezetben ezeknek a teszteknek a folyamatát fogom ismertetni.

Először a szegmentációért felelős Yolo hálózatot teszteltem, ugyanis a teljes rendszert használva is a kép először ehhez a komponenshez érkezik. Ebben az esetben az általam forgalomban készített képekkel való tesztelést megelőzte az eredeti adathalmazhoz tartozó teszt adathalmazzal való tesztelés. A kifejezetten a tesztelésre elkészített adathalmaz 75 darab képet tartalmazott, melyekhez rendelkezésre álltak a szükséges annotációs fájlok. A 9.2. táblázatból leolvasható, hogy az átlagos pontossága a hálózatnak ezeknél a képeknél 0,984, ami egy nagyon jó eredménynek mondható. Ennek az értéknek esetemben nincs nagy jelentősége, ugyanis a tesztadathalmaz nagysága súlyosan befolyásolhatta az eredményt, illetve a hálózatnak itt csak egy osztályba kellett besorolni az adott táblát, így könnyebb dolga volt, mintha több osztályba lett volna lehetősége besorolni a táblát.

Osztályazonosító	Képek száma	mAP
1	75	0,984

9.2. táblázat: Táblázat a betanított Yolo hálózat pontosságáról

A hálózat második tesztje már valós, saját adathalmazzal történt. Ebben az esetben a hálózatnak csak olyan képeket adtam át, amik tartalmaztak táblákat, ehhez az adathalmazhoz nem tartoztak annotációs fájlok, ezért a tesztelés eredményét manuálisan állapítottam meg. Az előző tesztesetben a táblát körülölelő keret pontossága volt ellenőrizve, ebben a tesztesetben azonban a hangsúly azon volt, hogy a kijelölt területen valóban tábla látható-e vagy sem. A képeket két külön út során rögzítettem, az első esetben kora reggeli felhős időjárás volt, míg a második esetben a képek kora délután készültek napos időjárási körülmények között. Az eredményekről elmondható, hogy az általam tesztelt képeknél a fényviszony nem befolyásolta a detektálás sikerességét. Abban az esetben, ha a nem detektált táblák számát figyelmen kívül hagyjuk, a detektálások 89,8%-a sikeres volt, ha figyelembe vesszük a nem detektált táblák számát is, akkor ez a szám 82,95% lesz. A 9.3. táblázatban láthatóak a hálózat tesztelési eredményei a saját adathalmazzal.

	Képek száma	Detektálások száma	Helyes detektálások száma	Helytelen detektálások száma	Nem detektált táblák száma
1. teszt	176	310	282	28	22
2. teszt	67	91	78	13	11
Összesen	**243**	**401**	**360**	**41**	**33**

9.3. táblázat: Táblázat a betanított Yolo hálózat pontosságáról saját képek esetén

A rendszer klasszifikálásért felelős komponense az InceptionV3 neurális hálózati architektúra. A megfelelő hálózati architektúra kiválasztásához számos tesztesetet kellett futtatnom, azonban ennek az Implementálás fejezethez volt köze, ezért annak folyamatát a 8.1.6 számú fejezetben fejtettem ki részletesen. Ebben a fejezetben már általam készített képekkel teszteltem a hálózatot, ezeket a képeket először szegmentálni kellett, amit a Yolo tett meg.

A teszteset adathalmazát egy tesztút alatt rögzített képek adták, melynek a száma 3840 darab volt. A Yolo hálózat ezeken a képeken 1532 darab táblát talált. A GTSRB adathalmaz 43 osztályt definiál, azonban ezt leginkább megjelenítési szempontok miatt csökkentettem le 12 darab osztályra. A 12 darab osztályba az átképzett InceptcionV3 neurális hálózati architektúra összesen 626 darab képet sorolt be, a különböző osztályokba. A 9.4. táblázatból leolvashatóak az elért eredmények, összességében elmondható, hogy a 85% feletti pontosság egy jó érték, azonban több osztálynál is látszik a 60%-nál is alacsonyabb elért pontosság. Úgy gondolom, hogy egyrészt ez a betanítási minta növelésével orvosolható lenne, illetve a GTSRB adathalmazban található táblák többsége csak részben egyezik meg a Magyarországon elhelyezett jelzőtáblákkal, ami szintén csökkenthette a pontosságot.

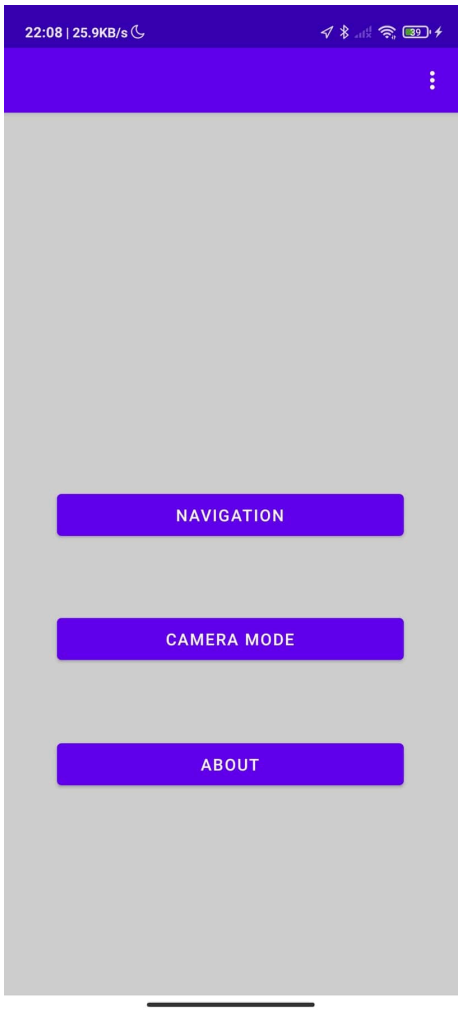
Sorszám	Megnevezés	Besorolások száma	Helyes besorolások száma	Elért eredmény (%)
1	Sebesség 30kmh	66	47	71
2	Sebesség 50kmh	0	0	-
3	Előzni tilos	43	25	58
4	Főút	31	30	97
5	Elsőbbség adás kötelező	114	106	93
6	Stop	41	24	58
7	Két irányú behajtani tilos	22	15	68
8	Behajtani tilos	37	30	81
9	Gyalogos átkelő	16	16	100
10	Kötelező haladási irány: egyenes	189	180	95
11	Kötelező haladási irány: egyenes vagy jobbra	47	47	100
12	Kötelező haladási irány: egyenes vagy balra	20	16	80
Összesen:		**626**	**536**	**85,62%**

9.4. táblázat: Táblázat a betanított InceptionV3 hálózat pontosságáról saját képek esetén

10. ELKÉSZÜLT RENDSZER BEMUTATÁSA

Az elkészült rendszer három részből áll. Áll a szerveroldalból, ideértve a neurális hálózatokat is, illetve áll a kliensoldalakból, amik közül az Android operációs rendszerre készült kliens egy összetettebb alkalmazás, míg a webkliens egy egyszerű weboldal, ami a táblák rögzítésére és megjelenítésére használható.

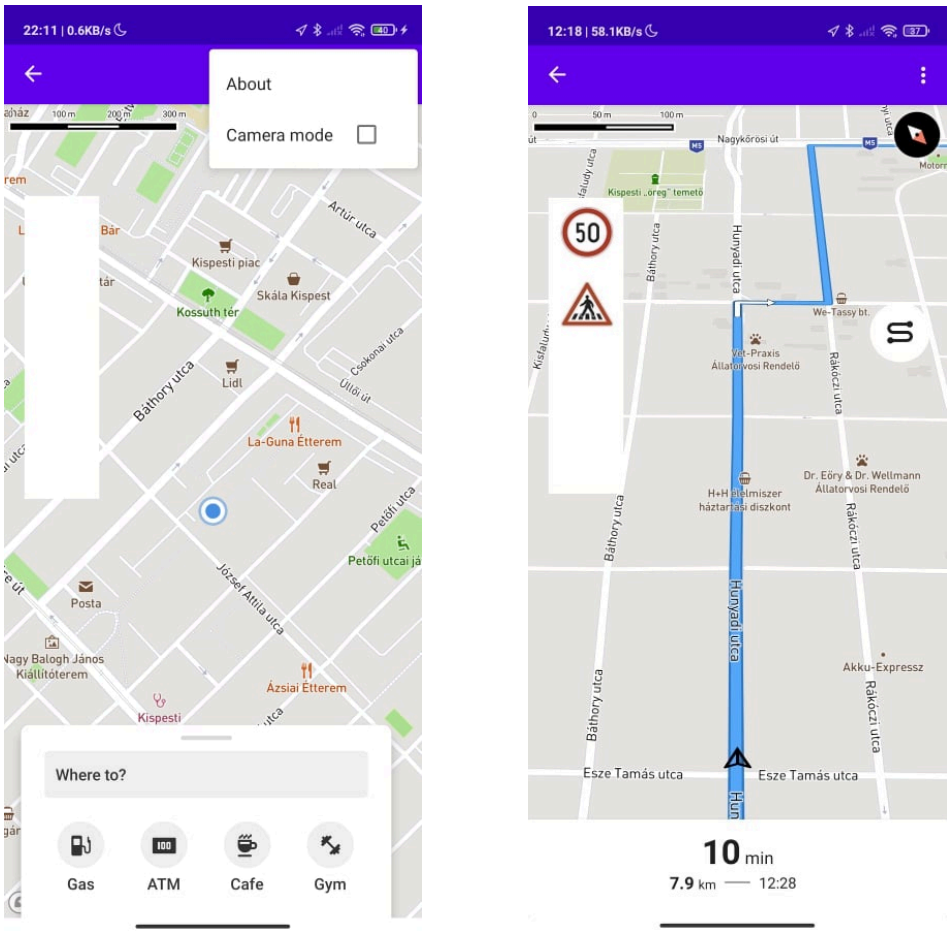
Az Androidra készült alkalmazás használatához a felhasználónak engedélyeznie kell az alkalmazás számára a helymeghatározást, illetve a kamerahasználatot. Miután ezt megtette egy egyszerű kezdőképernyő fogadja, ahol három menüpont közül választhat. Ezek a menüpontok a *Navigation*, a *Camera Mode* és az *About*. A jobb felső sarokban található gombra nyomva a felhasználó be-, illetve kikapcsolhatja a háttérben történő képtovábbítást a szerver felé.



10.1. ábra: Az Android alkalmazás kezdőképernyője

A *Navigation* menüpontot választva jelenik meg a térképnézet egy keresősávval, illetve a képernyő bal oldalán elhelyezkedő jelzősáv, ahol azok a táblák jelennek meg, amik a

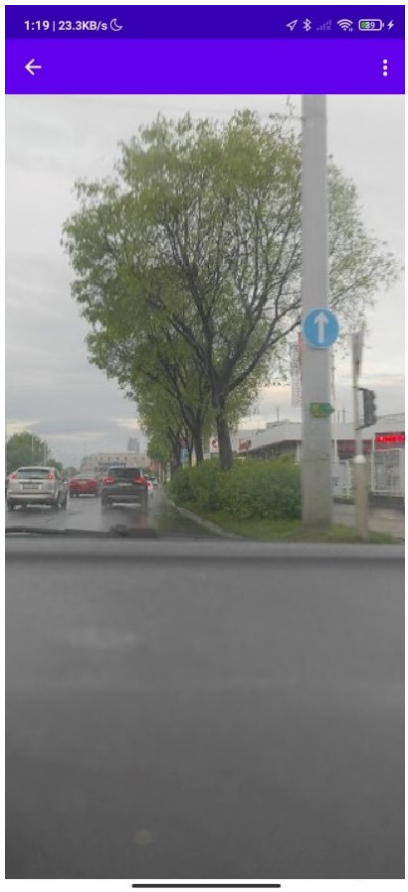
közelünkben vannak és már rögzítve lettek az adatbázisban. A 10.2. ábrán látható a térképnézet, illetve a navigációs nézet, ahol a jelzősávban két táblának a jelzése is látható. Ezt a nézetet akkor érdemes használnia felhasználónak, amikor ki szeretné használni a táblamegjelenítés funkciót, azonban konkrét navigációra nincs szüksége. Abban az esetben, ha navigációt is szeretne, akkor a keresőbe beírt célpont kiválasztása után fogadja a navigációs kijelző, aminek szintén a bal oldalán van elhelyezve a táblák jelzősávja, az előző képernyőhöz hasonlóan. Amennyiben rögzíteni is szeretne táblákat az út során, erre a jobb felső sarokban lévő gombra nyomva, majd ott a *Camera mode* jelölőnégyzetet kipipálva van lehetősége bekapcsolni. A táblamegjelenítés a felhasználó pozícióváltozásától függően történik, a táblák kirajzolása csak akkor valósul meg, ha a kliens helyzete megváltozott.



10.2. ábra: Az alkalmazás térképnézete (balra) és navigációs nézete (jobbra)

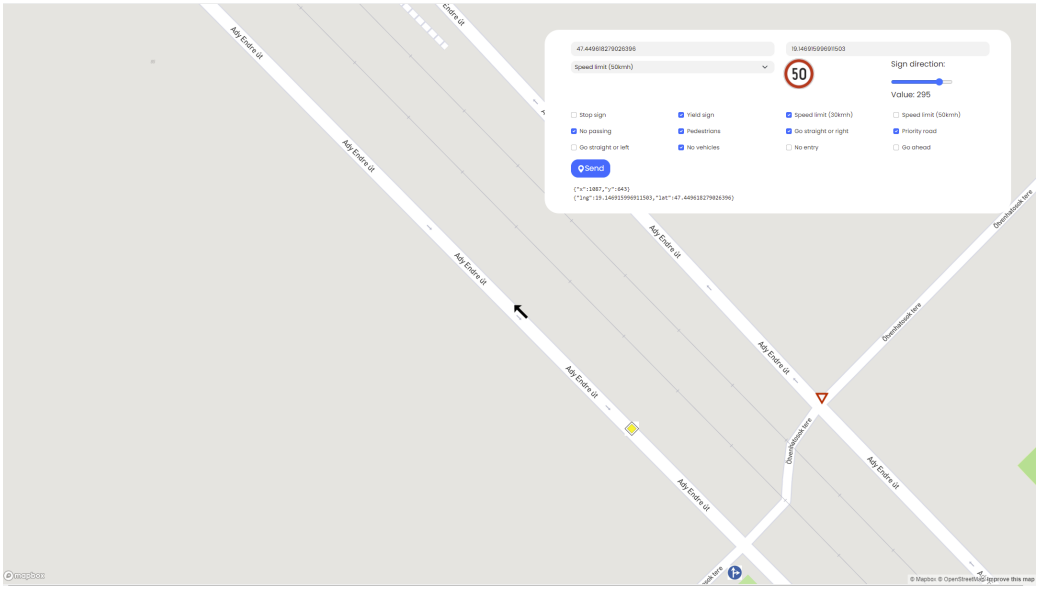
A főképernyőn a *Camera mode* menüpontot választva megjelenik az erre készült képernyő, ami teljes egészében a kamera élőképét mutatja a felhasználónak, melynek képernyője látható a 10.3. ábrán. Az alkalmazás ebben a módban 1-2 másodpercenként küld egy képet a szervernek, ami ezt feldolgozza. A szerverhez, amikor beérkezik kép,

először a szegmentálásért felelős neurális hálózathoz kerül a kép, ahol kivágja a látható táblákat, majd a kivágott képet vagy képeket továbbadja a klasszifikálásért felelős neurális hálózatnak. Miután a hálózat eldöntötte, hogy milyen tábla volt látható a képen, ezt rögzíti az adatbázisban a szükséges adatokkal együtt és ezek után a többi felhasználó számára látható lesz, mikor használják az alkalmazást.



10.3. ábra: Kamera mód használata élőkép megjelenítésével

A webkliens ad lehetőséget a felhasználónak a manuális táblarögzítésre, illetve arra, hogy a már rögzített táblákat lássa a térképen elhelyezve. A weboldal kinézete látható a 10.4. ábrán. Manuális táblarögzítéskor ki kell jelölni a pontot a térképen, beállítani, hogy a tábla melyik irányba néz, illetve ki kell választani, hogy milyen táblát szeretne rögzíteni a felhasználó. Miután ez megtörtént, a szerver megkapja ezeket az adatokat, majd ezek alapján rögzíti azt az adatbázisban. A táblamegjelenítésnél a szervertől kapott táblák vannak megjelenítve, a felhasználónak lehetősége van szűrni, hogy pontosan milyen táblát vagy táblákat szeretne látni a térképen.



10.4. ábra: A rendszer webes felülete

11. EREDMÉNYEK ÉRTÉKELÉSE

Ebben a fejezetben a szakdolgozatomban megvalósított rendszer eredményeit fogom értékelni. A rendszer eredményeinek viszonyítási alapja a korábbi fejezetekben általam vizsgált hasonló rendszerek elért eredményei lesznek.

11.1. ANDROID KLIENS EREDMÉNYEI

Az Android kliensben három fő funkciót kellett megvalósítanom, az egyik a szervernek való képküldés, a másik egy alapvető navigációs funkció, a harmadik pedig a rögzített táblák megjelenítése. A létrehozott alkalmazásban mindhárom funkció elérhető és működőképes. Hasonló rendszer szinten leginkább a Waze navigációs szoftverhez van közel, azonban a Waze funkció jóval kiforrottabbak és széleskörűbbek, mint az általam fejlesztett alkalmazásban megtalálható funkciók, azonban táblarögzítés és megjelenítés funkcióval egyáltalán nem rendelkezik leszámítva a sebességkorlátozások megjelenítését.

11.2. SZERVEROLDAL EREDMÉNYEI

A szerveroldal képes a kliensek kéréseit kiszolgálni, adatbázist kezelni, azonban a legfontosabb funkciója a képek fogadása, majd a képeken látható táblák szegmentálása és klasszifikálása.

A Tesztelés fejezetben, a klasszifikálás tesztelésénél elhelyezett táblázatban látható, hogy bizonyos tábláknak a besorolási pontossága nem tökéletes, többnél a 60%-ot sem éri el ez az érték. Ennek a problémának a megoldására úgy gondolom, hogy mindenképpen több betanítási adatra, illetve valós, Magyarországon elhelyezett táblákról készült képekre lenne szükség.

Hasonló rendszerek terén leginkább a harmadik rendszerrel lehet összevetni az általam elkészített komponenst, azonban teljes összehasonlítást nem lehet tenni a rendszerek között, ugyanis a szakdolgozatomban létrehozott rendszer külön neurális hálót használ a szegmentálásra és a klasszifikálásra is. Eredmények terén elmondható, hogy a szegmentálás esetemben hasonló eredményeket ér el, mint a harmadik rendszer nem besorolt táblák nevű kategóriája. Ez valószínűleg annak köszönhető, hogy abban az esetben és az én esetemben is, ebben az egy kategóriában többféle tábla volt megtalálható.

Az általam készített rendszernek nem volt szüksége a valós idejű működésre, azonban a feldolgozási idő terén összemérhető a másik három rendszerrel. A rendszerem a tapasztalataim alapján a szegmentálást, majd klasszifikálást 30-szor tudta megtenni másodpercenként, ami a hasonló rendszerek közül csak a YOLO-val megoldott rendszernél rosszabb.

	Road-Sign Detection and Recognition Base on Support Vector Machine	Faster R-CNN for Small Traffic Sign Detection	A YOLO Based Approach for Traffic Sign Detection	Saját rendszer
Technológia	SVM	FRCNN	YOLO	YOLO és InceptionV3
Sebesség	-	2,6 fps	45* fps	30 fps
mAP érték		0,482	0,525	0,984
Adathalmaz	saját képek	CCF BDCI 2016	BTSD	saját, online adathalmaz + GTSRB
Rendszer válasza	konkrét tábla	konkrét tábla	táblacsoport	konkrét tábla

11.1. táblázat: Az elkészült rendszer összevetése a hasonló rendszerekkel

12. TOVÁBBFEJLESZTÉS LEHETŐSÉGEI

Ebben a fejezetben a már elkészült rendszer továbbfejlesztési lehetőségeit fogom sorra venni. A rendszer szétbontható külön kliens- és szerveroldalra, először a kliensoldalak esetleges továbbfejlesztési irányaival fogok foglalkozni.

12.1. KLIENSOLDALAK

A jelenlegi webkliensen elérhető funkciók számát növelném a rendszer továbbfejlesztése esetén. Hasznos funkció lenne, ha a kilistázott tábláknak a szerkesztésére vagy adott esetben törlésére lenne lehetőség. Egyelőre erre csak az adatbázison belül van lehetőség, ez a két funkció jelentősen megkönnyítené ezeket a műveleteket. A weboldal, jelen állapot szerint nem rendelkezik védelemmel, emiatt csak belső használatra lenne ajánlott használni. Hasznos lenne egy regisztráció és bejelentkezés funkció, ami lehetőséget adna arra, hogy az oldal szélesebb kör számára is elérhető legyen, akár arra is, hogy a későbbiekben ellenőrizni lehessen egy-egy felhasználó rögzítéseit.

Az Android kliensről elmondható, hogy a kitűzött célokat eléri, azonban a működés hatékonyságán, sebességén lehetne javítani. A képek küldése jelenleg nagyjából másodpercenként történik, ezt a mennyiséget lehetne növelni, azonban az általam implementált változatban, ha csökkentem a képkockák küldése között eltelt idő mértékét az alkalmazás elkezd belassulni, ami a felhasználói élmény tekintetében nem előnyös változtatás, ezért valószínűleg egy másfajta implementáció lehetne a megoldás továbbfejlesztés esetén.

A kliensoldalak felhasználói felületeiről elmondható, hogy áttekinthetőek, illetve felhasználóbarátak, azonban a megjelenésükön lehetne úgy módosítani, hogy az megfeleljen napjainkban használt általános alkalmazáskinézet sztenderdjeinek.

Az Android kliens mellé le lehetne fejleszteni az IOS-re készült verziót is, az általam létrehozott rendszer előnye, hogy a magja a szerveroldal, így egy másik kliensoldal létrehozása kevésbé lenne időigényes és bonyolult.

12.2. SZERVEROLDAL

A szerveroldal jelenleg kívülről is elérhető a már említett *ngrok* alkalmazás segítségével, azonban nagy előrelépés lenne a projekt szempontjából egy dedikált szerver bérlése, ahol futhatna a szerveroldalra tervezett programkód.

A szerveroldalon leginkább a neurális hálózati architektúrákon lehetne módosításokat elvégezni. Ideális esetben nem lenne szüksége a második, InceptionV3 neurális hálózati architektúrára, ebből adódóan az lenne a lehetséges továbbfejlesztés ezen a téren, ha rendelkezésre állna megfelelő mennyiségű adat, így elég lenne csak a Yolo architektúrát betanítani és használni a klasszifikáció fázisára is.

13. ÖSSZEFOGLALÓ

A szakdolgozatom célja egy olyan rendszer megalkotása volt, amihez tartozik egy táblarögzítő és táblamegjelenítő funkciókkal bővített navigációs alkalmazás, egy manuális táblarögzítésre képes webkliens és egy szerveroldal, ami a táblák felismerésére és a kliensek kéréseinek kiszolgálására képes.

Összegyűjtöttem három navigációs alkalmazást, illetve három rendszert, amik képesek voltak a jelzőtáblák felismerésére. A hasonló rendszerek elemzése után állítottam össze a rendszertervet. A rendszerterv összeállításakor ügyeltem arra, hogy az Android operációs rendszerre elkészítendő alkalmazás hardverigénye ne legyen magas, emiatt a táblafelismerést a szerver végzi.

A navigációs, illetve a térképmegjelenítés funkciókhoz a MapBox SDK és az ahhoz tartozó egyéb könyvtárakat használtam fel.

A táblafelismerés két komponensből áll. Az első komponens a Yolo neurális hálózati architektúrát tartalmazza és ez felel a szegmentálásért. A második komponens a GoogleNet InceptionV3 neurális hálózati architektúrája, ami szegmentált képek klasszifikálásáért felel.

Az elkészült rendszer funkcióinak egy részét egységteszteltem, illetve manuális teszteltem, hogy megbizonyosodjak arról, hogy a rendszer megfelelően működik. A tesztelések eredményei kielégítőek voltak.

Végül a rendszer eredményeit elemeztem és az elérhető metrikák alapján összehasonlítottam az általam választott hasonló rendszerekkel.

A rendszer a specifikációban leírtaknak megfelel. A létrehozás során, illetve az elért eredmények tudatában több ötletem is van a már meglévő rendszer továbbfejlesztésére, amiket részletesen kifejtettem az ezt megelőző fejezetben.

14. SUMMARY

The goal of my thesis was to create a system that includes a navigation application with road sign recording and road sign display modes, a web client capable of manual road sign record function and a server side which is capable of recognizing road signs and serving client requests.

I collected three navigation applications and three similar systems that were capable of recognizing road signs. After analysing these similar systems, I put together a system design. When compiling the system design, I made sure that the hardware requirements of the application to be built for the Android operating system were not high, therefore the sign recognition is done by the server.

For the navigation and map display functions I used the MapBox SDK and their other libraries.

The table recognition consists of two components. The first component contains the Yolo neural network architecture, this is responsible for the segmentation part. The second component is the GoogleNet InceptionV3 neural network architecture, which is responsible for segmented image classification part.

I created unit tests and manual tests for some of the features of the completed system to make sure that it works properly. The results of the testing were acceptable.

Finally, I analyzed the results of the system and compared them with similar systems of my choice based on the available metrics.

The system is as described in the specification. In the course of its creation and in the light of the results obtained, I have several ideas for improving the existing system, which I have explained in detail in the previous chapter.

15. IRODALOMJEGYZÉK

[1]: (https://global.honda/heritage/episodes/1981navigationsystem.html) utoljára megtekintve: 2020. 10. 20.

[2]:https://onlinemasters.ohio.edu/blog/the-evolution-of-portable-gps/ utoljára megtekintve: 2020. 10. 20.

[3]: https://www.pocketgpsworld.com/tomtom-go.php utoljára megtekintve: 2020. 10. 20

[4]: How Nokia failed to nail the Smartphone market. 25th European Regional Conference of the International Telecommunications Society (ITS): "Disruptive Innovation in the ICT Industries: Challenges for European Policy and Business", Brussels, Belgium, 22nd-25th June, 2014

[5]: http://wazestats.com/activew.php utoljára megtekintve: 2020. 10. 20.

[6]: https://www.audi-technology-portal.de/en/electrics-electronics/driver-assistant-systems/night-vision-assistant utoljára megtekintve: 2021. 05. 01.

[7]: http://kozlekedesbiztonsag.kti.hu/veszelyben-a-gyalogosok-es-a-kerekparosok/ utoljára megtekintve: 2020. 10. 20.

[8]: https://www.ksh.hu/docs/hun/xstadat/xstadat_evkozi/e_feb002.html utoljára megtekintve: 2020. 10. 20.

[9]: https://blog.google/products/maps/look-back-15-years-mapping-world/ utoljára megtekintve: 2020. 10. 20.

[10]: https://googleblog.blogspot.com/2005/02/mapping-your-way.html

[11]: https://googleblog.blogspot.com/2007/02/stuck-in-traffic.html

[12]: http://googlepress.blogspot.com/2007/11/google-announces-launch-of-google-maps_28.html

[13]: http://googlemobile.blogspot.com/2009/10/announcing-google-maps-navigation-for.html

[14]: https://www.waze.com/hu/ utoljára megtekintve: 2020. 10. 20.

[15]: https://support.google.com/waze/answer/6078702?hl=en utoljára megtekintve: 2020. 10. 21.

[16]: https://www.statista.com/statistics/865413/most-popular-us-mapping-apps-ranked-by-audience/ utoljára megtekintve: 2021.05.11.

[17]: Economic Commission for Europe. Convention on Traffic Signs and Signals; Vienna Convention: Vienna, Austria, 1968

[18]: Safat, B., W., Majid, A., A., Mahammad, A., H., Aini, H, Salina, A., S., Pin, J., K., Muhamad, B., M.: Vision-Based Traffic Sign Detection and Recognition Systems: Current Trends and Challenges, 2019

[19]: Maldonado-Bascón, S., Lafuente-Arroyo, S., Gil-Jiménez, P., Gómez-Moreno, H., López-Ferreras, F.: Road-Sign Detection and Recognition Based on Support Vector Machines, 2007

[20]: Hsu, C., Chang, C., and Lin, C: A Practical Guide to Support Vector Classification, 2016

[21]: Zhang, Z., Zhou, X., Chan, S., Chen S., Liu, H.:
Faster R-CNN for Small Traffic Sign Detection, 2017

[22]: Mandikal, V.: A YOLO Based Approach for Traffic Sign Detection, 2018

[23]: Girshick R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5), 2014

[24]: Girshick, R.: Fast R-CNN, 2015

[25]: Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks, 2015

[26]: Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection, 2015

[27] https://developers.google.com/maps/documentation/android-sdk/overview utoljára megtekintve: 2020.11.02.

[28] https://docs.mapbox.com/android/maps/overview/ utoljára megtekintve: 2020.11.02.

[29] https://github.com/pjreddie/darknet utoljára megtekintve: 2022.04.20.

[30] https://www.kaggle.com/datasets/valentynsichkar/traffic-signs-dataset-in-yolo-format utoljára megtekintve: 2022.04.16.

[31] https://docs.mapbox.com/api/navigation/directions/ utoljára megtekintve: 2022. 03.25.

16. ÁBRAJEGYZÉK

3.1. ábra: saját készítésű ábra
4.1. ábra: Safat B. Wali, Majid A. Abdullah, Mahammad A. Hannan, Aini Hussain, Salina A. Samad, Pin J. Ker, Muhamad Bin Mansor: Vision-Based Traffic Sign Detection and Recognition Systems: Current Trends and Challenges

4.2. ábra: Saturnino Maldonado-Bascón, Sergio Lafuente-Arroyo, Pedro Gil-Jiménez, Hilario Gómez-Moreno and Francisco López-Ferreras: Road-Sign Detection and Recognition Based on Support Vector Machines

4.3. ábra: Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin: A Practical Guide to Support Vector Classification

4.4. ábra: Vikram Mandikal: A YOLO Based Approach for Traffic Sign Detection

7.1. ábra: saját készítésű ábra
7.2. ábra: saját készítésű ábra
7.3. ábra: saját készítésű ábra
7.4. ábra: saját készítésű ábra
7.5. ábra: saját készítésű ábra
8.1. táblázat: saját készítésű táblázat
8.2. ábra: saját készítésű ábra
8.3. ábra: saját készítésű ábra
8.4. táblázat: saját készítésű táblázat
8.5. ábra: saját készítésű ábra
8.6. táblázat: saját készítésű táblázat
8.7. ábra: saját készítésű ábra
9.1. táblázat: saját készítésű táblázat
9.2. táblázat: saját készítésű táblázat
9.3. táblázat: saját készítésű táblázat
9.4. táblázat: saját készítésű táblázat
10.1. ábra: saját készítésű ábra
10.2. ábra: saját készítésű ábra
10.3. ábra: saját készítésű ábra
10.4. ábra: saját készítésű ábra
11.1. táblázat: saját készítésű táblázat