ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Mészáros Róbert**

2023 Hallgató törzskönyvi száma: **T/006868/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Mészáros Róbert**
Törzskönyvi száma: T/006868/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Közlekedési sávdetektálás és KRESZ tábla felismerés
Lane detection and traffic sign recognition

Intézményi konzulens: Dr. Sergyán Szabolcs
Külső konzulens:

Beadási határidő: 2022. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és –fejlesztés
specializáció

A feladat:

Tervezzen meg és készítsen el egy gépkocsi vezetését támogató rendszert, amely képes felismerni a gépjármű haladási sávját autópályán, illetve normál utakon különböző képfeldolgozási algoritmusok alkalmazásával. A rendszer jelölje ki a felismert sávot, az autó pozícióját a sávon belül, illetve a sáv szélességét és görbületét. Továbbá vizuálisan jelenítse meg a különböző képfeldolgozási lépések eredményét. A vezetéstámogató rendszer legyen képes detektálni, és felismerni a közlekedési táblákat is, konvolúciós neurális hálózat felhasználásával. A neurális hálózat a GTSRB, német KRESZ tábla adatbázissal legyen betanítva. Ehhez végezzen el különböző előfeldolgozási lépéseket a KRESZ tábla adatbázison, a modell betaníthatósága érdekében. A rendszer jelölje ki ezen közlekedési táblákat, illetve írja fölé a nevét és azt, hogy milyen pontossággal ismerte fel. Tábla felismeréséhez próbáljon ki többféle konvolúciós neurális hálózatot, és mutassa meg a különböző modellek tanítási eredményeit. Ezt a két alrendszert lehessen együtt is használni, és a feldolgozott képkockákból készüljön videó.

A dolgozatnak tartalmaznia kell:

- a feladat leírását,
- a szakirodalom alapján megismert hasonló módszerek, rendszerek ismertetését és értékelését,
- a választott módszer bemutatását,
- a megvalósítandó feladat tervét,
- a felhasználói leírást,
- a tesztadatokat, feltüntetve azok forrását és a teszteredményeket,
- az elért eredmények értékelését,
- a továbbfejlesztési lehetőségeket,
- a dokumentációt, a programot, a szükséges input adatokat, valamint a rendszert bemutató prezentációt CD mellékleten.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

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, 2022.12.15.....

Mészáros Róbert

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Mészáros Róbert ..... [redacted] ..... Nappali .....

Telefon: Levelezési cím (pl: lakcím):

[redacted] .....

Szakdolgozat / Diplomamunka[1] címe magyarul:

Közlekedési sávdetektálás és KRESZ tábla felismerés .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Lane detection and traffic sign recognition .....

Intézményi konzulens: Külső konzulens:

Dr. Sergyán Szabolcs ....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.02.14	KRESZ tábla detektálás, és felismerés módszereinek átbeszélése	
2.	2022.03.04	Sávdetektálás átbeszélése	
3.	2022.04.22	KRESZ tábla felismerés, és sávdetektálás egybeépítése	
4.	2022.05.10	Dokumentáció tartalmának átbeszélése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.05.11 .....

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: Neptun kód: Tagozat:

Mészáros Róbert ..... [redacted] ..... Nappali .....

Telefon: Levelezési cím (pl: lakcím):

[redacted] .....

Szakdolgozat / Diplomamunka[1] címe magyarul:

Közlekedési sávdetektálás és KRESZ tábla felismerés .....

Szakdolgozat / Diplomamunka[2] címe angolul:

Lane detection and traffic sign recognition .....

Intézményi konzulens: Külső konzulens:

Dr. Sergyán Szabolcs ....

Kérjük, hogy az adatokat nyomtatott nagybetűkkel írja!

Alk.	Dátum	Tartalom	Aláírás
1.	2022.09.10	Teljesítendő feladatok átbeszélése	
2.	2022.10.04	Táblafelismerő tanítási eredményeinek bemutatása	
3.	2022.04.22	Tesztelés, és dokumentáció bemutatása	
4.	2022.11.23	Végleges program, és dokumentáció bemutatása, és átbeszelése	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. / Szakdolgozat II. (BSc) vagy Diplomamunka 1 / Diplo-mamunka 2 / Diplomamunka 3 / Diplomamunka 4[3] tantárgy követelményét teljesítette, beszámolóra / védésre[4] bocsátható.

Budapest, 2022.11.28 .....

.....
intézményi konzulens

[1] Megfelelő aláhúzandó!
[2] Megfelelő aláhúzandó!
[3] Megfelelő aláhúzandó!
[4] Megfelelő aláhúzandó!

Tartalomjegyzék

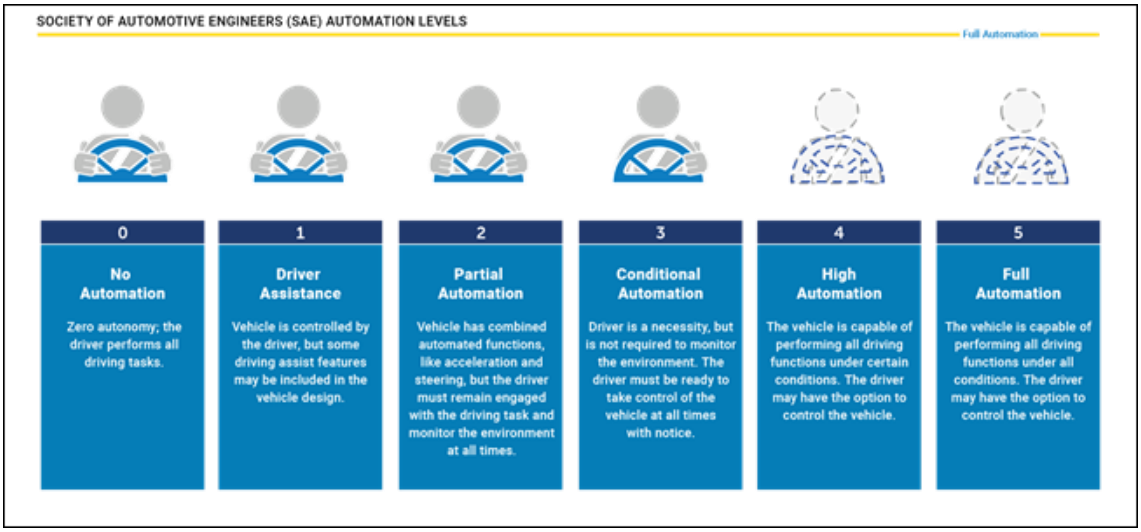
1. Bevezetés ... 3
2. Hasonló megoldások ... 6
2.1. Sávdetektálás ... 6
2.2. Táblafelismerés ... 7
3. Részterületek ... 8
3.1. Sávdetektálás ... 8
3.1.1. Canny éldetektáló algoritmusa ... 8
3.1.2. Sobel éldetektáló algoritmusa ... 8
3.1.3. Prewitt éldetektáló algoritmusa ... 10
3.1.4. Robert éldetektáló algoritmusa ... 10
3.1.5. Laplace (LoG) éldetektáló algoritmusa ... 10
3.1.6. Hough transzformáció ... 11
3.1.7. Perspektív transzformáció ... 11
3.1.8. HLS küszöbölés ... 13
3.1.9. Kalman szűrő ... 14
3.2. Táblafelismerés ... 17
3.2.1. Neurális hálózatok ... 17
3.2.2. Tanulás folyamata ... 21
3.2.3. Konvolúciós neurális hálózatok ... 21
3.2.4. Képfelismerés konvolúciós neurális hálózat segítségével ... 22
3.3. Sávváltás ... 23
3.3.1. Sávváltási döntések ... 24
3.3.2. Környezeti objektumok detektálása ... 25
3.3.3. Baleset elkerülés ... 27
3.3.4. Path planning algoritmusok ... 28
3.4. Összefoglalás ... 29
4. Implementáció ... 30
4.1. Sávdetektáló modul ... 30
4.2. Közlekedési tábla felismerő modul ... 41
4.2.1. Tábladetektálás ... 41

4.2.2. Táblafelismerés 44
4.2.3. Tanítási eredmények 50
4.2.4. Tábladetektálás, és táblafelismerés együttes használata 56
5. Eredmények kiértékelése 59
5.1. Sávdetektálás 59
5.2. Táblafelismerés 61
6. Továbbfejlesztési lehetőségek 63
7. Felhasználói leírás 65
8. Összefoglalás 67
9. Summary 68
10. Irodalomjegyzék 69

1. BEVEZETÉS

Az autóipar a 21. században nagyon sok fejlődésen esett át vezetősegítő funkciók terén. Az egyik nagy áttörést az automata parkolás jelentette 2003-ban. Ezután 2010-es években újabb funkciók jelentek meg, például sáv elhagyás figyelő, holtpont figyelő, keresztetírányú forgalomfigyelő, tábla felismerő, adaptív tempomat és még sok minden más. A következő nagy áttörést a Tesla jelentette 2014-ben, amikor bemutatta az autopilot technológiát, mely segítségével az autó képes automatikus kormányzásra, amelyet különféle vezetősegítő berendezéssel old meg. Ezen technológiák nemcsak a vezető kényelmét, illetve biztonságát szolgálják, hanem más közlekedésben résztvevőkét is.

Az önvezetés terén hatféle szint van, amellyel meg lehet határozni egy autó autonómiáját, azaz, hogy egy autó mennyire képes saját magát kezelni. Ezeket a 1.1-es ábrán lehet látni.



ábra 1.1: Autóipari automatizálási szintek.[1]

- A nulladik szint azt jelenti, hogy a sofőrnek kell minden feladatot ellátnia vezetés közben, tehát semmilyen automatizálás még nincs itt jelen.
- A következő szint már magába foglalja a különböző vezetősegítő funkciókat, amely csak információt szolgáltat a sofőrnek, még semmilyen beavatkozást nem végez.
- A második szinten már megjelennek bizonyos beavatkozások az autó részéről, mint például a gyorsítás, lassítás, sávtartás, viszont a sofőrnek ugyanúgy kell figyelnie a vezetésre, a körülötte lévő környezetre, úgy mintha nem lenne semmilyen vezetéssegítő funkció.
- A harmadik szinten már nem muszáj állandóan informálódjon a körülötte lévő környezetről, ugyanakkor mindig készen kell álljon az irányítás átvételére.

- A negyedik szinten az autó szinte minden vezetéshez szükséges feladatot el tud látni bizonyos feltételek mellett, a sofőr kedve szerint átveheti az irányítást.
- Végül az utolsó szinten az autó képes minden vezetői feladatot ellátni, minden feltétel mellett, és szintén megvan a lehetősége a sofőrnek az irányítás átvételére.

Fontos kiemelni, hogy ezen funkcióknak szükséges folyamatosan információt kinyerniük a környezetükből, környező forgalmat, sávokat, táblákat, távolságokat ahhoz, hogy megfelelően működjenek. Ezen információk kinyerésére számos szenzor, illetve algoritmus szükséges, ami feldolgozza a bejövő információkat. Számos részterület van az önvezetés témáján belül, ezeket szeretném a következőkben ismertetni.

A szakdolgozatom célja egy olyan rendszer kifejlesztése, amely különböző képfeldolgozó algoritmusok segítségével képes detektálni autópályán az aktuális sávot, illetve konvolúciós neurális hálózat segítségével pedig közlekedési táblákat felismerni.

A két modul, a sávdetektáló és a tábla felismerő is az autonóm vezetés két elengedhetetlen rendszere, melyek a biztonságos közlekedést segítik elő.

A sáv detektálását számos képfeldolgozói algoritmus segítségével valósítanám meg. Számos tényezőt figyelembe kell venni a folyamat során, mivel a sávok felfestései nem mindig tökéletesek, ha pedig mégis, akkor is előfordulhat, hogy valamilyen környezeti hatás miatt mégsem láthatóak annyira. Így figyelembe kell venni az olyan eseteket is, amikor nem lehet sávot detektálni, vagy pedig az hibásan történik. Erre szeretnék bevezetni egy hibakezelést, mely során hibás sáv detektálása esetén a már előzőleg detektált sáv paramétereit felhasználnám az aktuális sáv detektáláshoz. Természetesen ezt nem lehetne állandóan használni, mivel csak egy ideig tárolnám el az előző sávok paramétereit annak érdekében, hogy ne legyen későbbiek során hibás detektálás. Mindig a legfrissebb valid detektálás eredményeit tárolnám el. Továbbá szeretném kiírni a detektálás különböző fázisait is, hogy könyebben lehessen látni az egyes lépések eredményeit. Ezenkívül szeretném meghatározni a detektált sáv különböző paramétereit is, mint például a szélességet, illetve hogy a sávon belül hol helyeszkedik el a jármű. Ezt az információt majd fel lehetne használni arra, hogy a sáv megtartása érdekében milyen szögben kelljen eltekerni a kormányt.

A közlekedési táblák feldolgozásánál detektálni szeretném az összes látható táblát az adott képen, ami lehetséges. Ehhez majd konvolúciós neurális hálózatot (Convolutional Neural Network - CNN) szeretnék használni. Ennél is figyelni kell a különböző környezeti hatásokra, melyek ronthatják a detektálás minőségét. Emiatt fontos, hogy a tanító adatbázis ne csak egy környezetben tartalmazzon táblákat. Egy hatékony tábla detektáló és felismerő rendszer képes az összes táblát detektálni és felismerni. A

szakdolgozatom során szeretnék kipróbálni többféle modellt, melyeket többféle adattal tanítanék be, és ki szeretném majd értékelni is az eredményeket, hogy melyek bizonyulnak a legjobbnak. A tanításhoz a GTSDB és a GTSRB adatbázist használnám, az előbbi a tábladetektáló modell betanításához, az utóbbit a táblafelismerő betanításához használnám. A GTSRB adatbázis mivel nem kiegyenlített, így különböző előfoldolgozói lépések segítségével szeretném kiegyenlíteni, majd kiértékelni az eredeti és a kiegyenlített adatbázissal betanított modelleket.

2. HASONLÓ MEGOLDÁSOK

2.1. Sávdetektálás

Sávok detektálására az egyik klasszikus módszer az előbb említett hagyományos képfeldolgozói metódusokon alapszik, amik természetesen sok számítással járnak. A másik lehetőség konvolúciós neurális hálózatok használata. Az ilyen módszereknél fel kell építeni egy megfelelő konvolúciós neurálos hálózatot, majd be is kell tanítani. A tanító adatok úgy néznek ki, hogy a sávokról készült képeket felcímkézik, vagyis kijelölik, hogy hol találhatóak a sávvonalak. Az egyik ilyen megoldás Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans és Luc Van Gool [2] által megalkotott neurális hálózat. Magára a sávdetektálásra egy szegmentációs problémaként tekintettek, így két részre bontották a neurális hálózatot. Az egyik rész a sávvonalak szegmentációjáért felel, ami azt jelenti, hogy egy binarizált képet készít, amelyen csak a sáv pixelek láthatóak. A tanító mintáknál arra is ügyeltek, hogy a sávokat úgy címkézzék fel, hogy a nem látható részeket is bejelöljék, amik például egy autó által vannak eltakarva. Így olyankor is képes lesz sávokat detektálni, ha azok valamiért nem látszódnak rendesen. A neurális hálózat másik része pedig azért felel, hogy az első rész által kiadott pixeleket kluszterezze a megfelelő sávokba.

A következő lépésben az így kapott pixelekre egy polinomot kell illeszteni, hogy ki tudják rajzolni a sávvonalakat. Ehhez tipikusan egy perspektív transzformáció szükséges, hogy könyebben és pontosabban tudják meghatározni a polinomot. Természetesen a transzformációs mátrix minden képre ugyanaz, ami olyankor lehet problémás, mikor az út nem teljesen sík, például egy lejtőnél. Ennek kiküszöbölése érdekében egy másik neurális hálózatot tanítottak be, amely meghatározza a megfelelő transzformációs mátrixot az útviszonyoknak megfelőlen. Így nem teljesen egy felülnézetes kép lesz az eredmény, hanem inkább egy olyan kép, amelyre a polinomokat megfelelően rá lehet illeszteni.

Qing Lin, Youngjoon Han és Hernsoo Hahn által bemutatott publikációban [3] egy modell alapaú rendszert dolgoztak ki, amelyet tulajdonságon alapuló algoritmusokkal bővítettek ki. A modell alapú számításoknak az a lényege, hogy a sávvonalakat egy görbeként írják le, melyek néhány paraméteren alapszanak. Ezáltal a rendszer sokkal robosztusabb lesz a zajokkal vagy a roszabbul látható burkolati felfestésekkel szemben. A tulajdonságon alapuló módszereknél a sávoknak a különbözű tulajdonságait detektálják, mint pédául a széleit, színét, szélességét vagy orientációját.

Miután kinyerték az úgynevezett ’Region of Interest’-et (ROI), a sáv hipotézis generálása következik. Ennek az a célja, hogy a sávvonalak régióit kinyerjék a ROI-ból. Ehhez először éldetektálást használtak, majd egy él összekötő algoritmust annak érdekében, hogy jobban megtalálják egy sávvonalhoz tartozó éleket. Ezután pedig megnézik, hogy az így kapott élek közötti távolság megfelel-e egy sáv szélességének, ha igen, akkor azokat a területeket kiválasztják. A sáv hipotézis verifikálás lényege az,

hogy megnézzék az előbb detektált sáv régiók színeit, ezzel is kiszűrve a zajokat. Az így kapott eredményre pedig már rá lehet illeszteni a sávvonalakat.

2.2. Táblafelismerés

A közlekedési táblák detektálására és felismerésére is többféle megoldás létezik. Az egyik ilyen megoldás Yi Yang, Hengliang Luo, Huarong Xu, Fuchao Wu által készített publikációban látható [4]. Ebben a megoldásban a detektálás egy szín alapú szegmentáción alapszik. A közlekedési táblák meghatározott színűek lehetnek. A szegmentációs folyamat úgy működik, hogy először az RGB értékeket Ohta [5] értékekké konvertálják, majd egy szín valószínűségi táblázatot készítenek belőle. Mindegyik színhez meghatároznak egy küszöb értéket, tehát a nagyobb intenzitású területeken lesz valószínüleg közlekdedési tábla. Természetesen az így kijelölt területek nem biztos, hogy megfelelőek, így a stabilizálásához Maximally Stable Extremal Regions (MSER) [6] detektálót is alkalmaztak. Az így kapott eredményekben előfordulhatnak fals pozitív detektálások is, így ahhoz, hogy ezeket csökkentsék, illetve majd bekategorizálják a megfelelő szülő osztályba (Tiltó, Kötelező, Veszély) Support Vector Machine-t [7] (SVM) alkalmaztak.

Az RGB színtérből az Ohta színtérbe való konvertálás, illetve a valőszínüségi táblázat kiszámítása időigényes, így nem lehet valós időben használni. Ahhoz hogy mégis lehessen alkalmazni valós időben, egy look up táblázatot készítettek, melyben kiszámították az összes lehetséges valószínüséget. A detektálás során pedig kiszámítják minden pixel indexét az RGB értékei alapján, és megkeresik a hozzátartozó valószínüséget az előre elkészített look up táblázatban.

A felismeréshez három CNN modellt használtak mindhárom osztályhoz, hogy meghatározzák az egyes kategóriákon belül, melyik tábla látható. A felismerésnél szürke képekké konvertálják a bejövő adatokat, hogy csökkentsék a számításokat. Mindhárom CNN modell felépítése ugyanaz az utolsó, teljesen kapcsolt réteg neuron számán kívül, mivel mindhárom kategóriában különböző számú tábla található.

A [8]-ban látható megoldásban 3 fő lépés figyelhető meg. Az első lépésben szintén szín alapú szegmentációval határozzák meg a ROI-kat. A második lépésben pedig alakzat alapú detektálást valósítanak meg. Az utolsó, harmadik lépésben pedig a klasszifikáció történik.

A szín alapú szegmentációhoz HSI színtérbe konvertálják a képet, majd a kék, illetve a piros színhez beállítanak egy küszöbértéket. Az így kapott ROI csak abban az esetben lehet jó, ha megfelelő méretű. A következő lépésben az alakzat alapú klasszifikációt nem neurális hálózattal oldják meg, hanem invariáns geometriai momentumokon [9] alapuló módszerrel. A felismeréshez pedig Random Forest klasszifikátort [10] alkalmaztak, ami sokkal robosztusabb zajokkal szemben. A lényege az, hogy tetszőleges számú fa van, melyek mindegyike lead egy voksot egy osztályra, és a legtöbb osztályra leadott voks lesz a végleges kimenet.

3. RÉSZTERÜLETEK

3.1. Sávdetektálás

A sáv detektálása az egyik alapkő az önvezető autók számára. Enélkül az autó nem tudná kormányozni önmagát, nem tudná hol kellene mennie a közlekedés során. Azt is fontos meghatározni, hogy az autó hogyan helyezkedik el az úton, így észlelve azt, hogy ha elhagyná az aktuális sávot.

3.1.1. Canny éldetektáló algoritmusa

A Canny algoritmus[11] [12] [13] 5 lépésből áll: Először az eredeti képen csökkentjük a zajt Gauss szűrővel, utána kiszámoljuk a kép gradienseit minden pixelre, ezzel megkapjuk, hol változnak a kép intenzitás értékei jelentősen, tehát a gradiensnek a lokális maximuma adja meg az élpontokat. Ezzel megkaptuk, mely pontokban vannak élpontok. Ezt követően a számunkra nem lényeges, tehát a nem maximum pontokat el kell nyomnunk, ezt nevezzük ***Non-maxima suppression***-nek. Ezt követően még maradhatnak zajok, vagy hibás élpontok a képen. Az ilyen pontok eltávolítására hiszterézis küszöbölést kell alkalmaznunk. Kétféle szűrő érték van megadva: egy magas és egy alacsony. Ha egy pixelnek nagyobb az értéke a magas szűrőnél, akkor az egy él pixel. Ha egy pixel értéke nagyobb az alacsony szűrőnél, illetve az egyik szomszédja egy él pixel, akkor az is él pixel, minden más esetben pedig nem él pixelről van szó.

Gradiens kiszámolására használt maszkok a 3.1-es ábrán láthatóak.

-1	0	1	1	2	1
-2	0	2	0	0	0
-1	0	1	-1	-2	-1

ábra 3.1: Canny maszkok [17]

Hátránya, hogy csak jó látási viszonyoknál működik megfelelően.

3.1.2. Sobel éldetektáló algoritmusa

Sobel algoritmusa [13] [14] [15] [16] gradiens alapú, vagyis kiszámoljuk a ***x*** és ***y*** koordináta parciális deriváltjait, ami megadja, mely pixelek intenzitása változik a legjobban. Ehhez kétféle maszkot használ: egy horizontálisat, és egy vertikálisat, amelyek a 3.2-es ábrán láthatóak.

$$VM = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, HM = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

ábra 3.2: Sobel operator maszkok.[15]

Horizontális, illetve vertikális gradienseket a 3.3-as ábrán látható módon lehet kiszámolni egy 3x3-as kép esetén.

P_0	P_1	P_2
P_3	P_4	P_5
P_6	P_7	P_8

ábra 3.3: 3x3-as kép, ahol az egyes cellák egy pixelnek felelnek meg [15]

Horizontális gradiens: Gx=f1-f2, ahol

f1=(P6+2*P7+P8)

f2=(P0+2*P1+P2)

Vertikális gradiens: Gy=f3-f4, ahol

f3=(P2+2*P5+P8)

f4=(P0+2*P3+P6)

Ebből kifolyólag az eredő gradiens az G=|Gx|+|Gy|

Miután meghatároztuk minden pixel gradiensét, utána ezen értékeket össze kell hasonlítani a küszöbértékkel, hogy meghatározhassuk, hogy egy pixel élpixel-e vagy sem. Ha a pixel gradiense nagyobb, mint a küszöbérték, akkor az élpixelt jelöl, különben pedig csak zajt.

3.1.3. Prewitt éldetektáló algoritmusa

Prewitt algoritmusa [17] nagyon hasonló Sobel algoritmushoz, a maszkban térnek csak el. A maszkokat az 3.4-es ábrán lehet látni.

-1	0	1
-1	0	1
-1	0	1

-1	-1	-1
0	0	0
1	1	1

ábra 3.4: Prewitt maszkok [17]

3.1.4. Robert éldetektáló algoritmusa

Robert algoritmusa [17] szintén hasonlít Sobel algoritmusához, viszont amíg Sobel 3x3 maszkokat használ, addig Robert csak 2x2-es maszkokat, ahogy a 3.5-ös ábrán is lehet látni, és ez a két maszk 90°-kal van elforgatva egymáshoz képest.

1	0
0	-1

0	1
-1	0

ábra 3.5: Robert maszkok [17]

3.1.5. Laplace (LoG) éldetektáló algoritmusa

LoG algoritmusa [17] megkeresi, mely pixelek változnak a legjobban a képben. A LoG szűrű a pixelek másodfokú deriváltjait veszi, és ezeknek megkeresi azon pontjait, amik áthaladnak a 0 ponton. Ez a megoldás viszont nagyon érzékeny a zajokra, így szükséges algoritmus futása előtt egy Gauss szűrőt alkalmazni a képen a zajok csökkentésére. A pixelek másodfokú deriváltjait a 3.6-os ábrán látható módon lehet kiszámolni.

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

ábra 3.6: pixel másodfokú deriváltja [17]

Ez a algoritmus is maszkokat használ a gradiensek kiszámolására, amelyek a 3.7-es ábrán láthatóak.

1	1	1	-1	2	-1
1	-8	1	2	-4	2
1	1	1	-1	2	-1

ábra 3.7: LoG maszkok [17]

3.1.6. Hough transzformáció

Hough transzformációval [18] [19] [20] egyeneseket, illetve görbéket lehet egy képen detektálni, úgy, hogy a szakadozott vonalak esetén is működik. Ezen algoritmus működéséhez továbbá szükség van egy éldetektáló algoritmusra is, hogy detektálni tudjuk a sávokat.

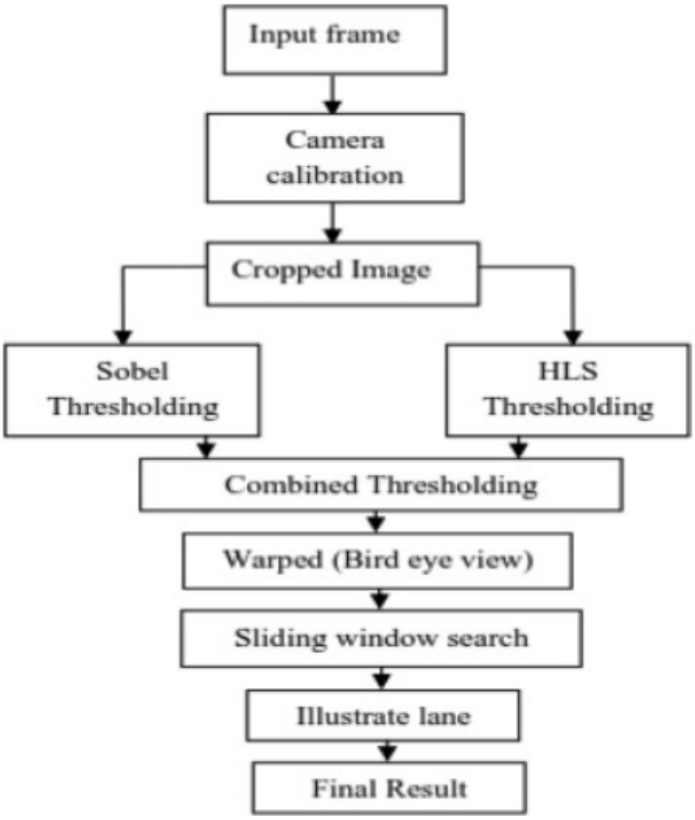
Az algoritmus alapja a Descartes-, és a polárkoordináta rendszer közötti átmenet. Egy egyenes egyenlete az ***x, y*** tengelyen ***y=mx+b***, ami polárkoordinátával felírva ***ρ=x*cosθ+y*sinθ***, ami egy pontot reprezentál.

Az algoritmus úgy működik, hogy először létrehoz egy 2D-s számláló tömböt a paramétertérben mely (***θ, ρ***) dimenziójú. A ***θ*** itt egy szöget jelöl, a ***ρ*** pedig egy pont a paramétertérben, ami egy távolságot jelöl. Az algoritmus ezt követően végigmegy minden pixelen a bemeneti képen, kiszámolja mindegyiknek a Hough transzformáltját, amely megadja a ***ρ*** értékét, és akkor az így létrejött (***θ, ρ***) koordináta értékét növelem 1-gyel. Majd ezután megkeressük, hogy melyik cella értéke a legnagyobb, majd ennek a cellának megfelelő egyenes lesz a legrelevánsabb a képen. Ezután pedig inverz Hough transzformációval meghatározzuk az egyenest, és kirajzoljuk a képre.

Hátránya az algoritmusnak az, hogy mindegyik pixelen végigmegy, emiatt sok számítást kell végrehajtania, továbbá a 2D-s számláló tömb mérete a bemeneti kép nagyságától is függ, szóval a memória igénye is jelentős.

3.1.7. Perspektív transzformáció

A perspektív transzformációs [21] [22] megoldáshoz több komplex folyamatot kell elvégezni, hogy megfelelően tudjuk használni. A szükséges lépéseket a 3.8-as ábra mutatja:



ábra 3.8: Sávdetektálási folyamatok [22]

A képből láthatjuk, hogy számos folyamat elvégzése szükséges, mielőtt még a perspektív transzformációhoz érünk. Ezen folyamatokat szeretném ismertetni a következőkben.

Az első a kamera által felvett kép kalibrálása. A bemeneti videó a kamera fizikai tulajdonságaiból adódóan torzulásokkal lehet tele. Ezek mind ronthatnak a sávdetektálás pontosságán, emiatt fontos, hogy ezeket a torzulásokat megszüntessük. Ez pedig a kamera kalibrálásával lehetséges.

Ezután a torzulásmentes képről le kell vágni a felesleges részeket. Ez amiatt fontos, hogy csak arra a részre fókuszáljunk, ahol a sáv megtalálható. Ezzel sok felesleges számítást elkerülhetünk, ami növeli a teljesítményt. Ezután pedig a megmaradt képet szürkeárnyalatossá konvertáljuk.

A következő lépésben jön a kép binárissá alakítása Sobel és HLS küszöböléssel annak érdekében, hogy pontosabb legyen a sávok detektálása.

Miután mindkét szűrőt alkalmaztuk, jöhet a perspektív transzformáció. Ennek hatására olyan 2D-s képet kapunk, ami olyan mintha madártávlatból készült volna. Ez amiatt szükséges, mert az eredeti képen a sávok vonalai bizonyos távolság után olyanok,

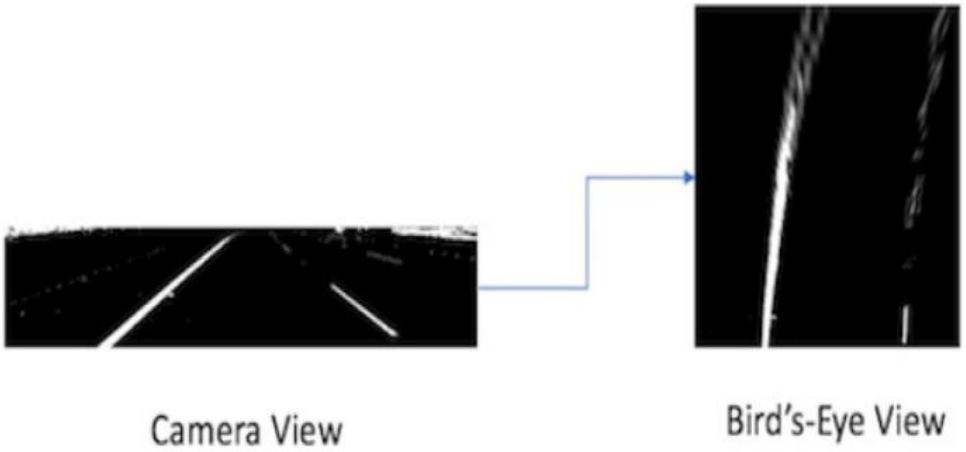
mintha egymásba mennének, viszont tudjuk, hogy valójában mindig párhuzamosak. A transzformációs mátrixot az alábbi módon lehet kiszámolni:

D* = *M**S, ahol ***D***, illetve ***S*** egy négyszög pontjait jelölik a cél, illetve forrás képen, ***M*** pedig maga a transzformációs mátrix, ami 3x3-as. Az ***M*** mátrix kiszámolása a 3.9-es ábrán látható módon történik:

$$dst\,(x,\ y) = \ src\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}},\ \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right),$$

ábra 3.9: M traszformációs mátrix kiszámítása [21]

A transzformáció után viszont végig párhuzamosak lesznek, ahogy a 3.10-es ábrán is látható. Ezután pedig láthatjuk a sáv vonalait, amit már csak fel kell ismerni, és ki kell jelölni. Mindezt a csúszó ablak kereséssel tudjuk megoldani. Az a lényege, hogy a képen megkeresi azokat a részeket, ahol sok az élpixel, majd ezután rá lehet illeszteni egy másodfokú polinom görbét azokra a pontokra, ahol a legnagyobb az élpixelek sűrűsége. Ezután pedig a kijelölt vonalak a bemeneti képen fognak megjelenni. Ezt követően ki lehet számolni a sáv görbületét és az autó pozícióját a sávon belül.



ábra 3.10: Perspektív transzformáció előtt és után [22]

3.1.8. HLS küszöbölés

A HLS [23] három komponensből áll: Színárnyalat (hue), világosság (lightness) és telítettség (saturation).

A HLS színtér az RGB színtérnek egy alternatív változata. Ezzel a színtérrel könnyebben lehet a színeket elválasztani egymástól. RGB-ből HLS-be történő konverzió gyors és egyszerű, ami a 3.11-es ábrán látható módon történik.

$$V_{max} \leftarrow \max(R, G, B),$$

$$V_{min} \leftarrow \min(R, G, B),$$

$$L \leftarrow \frac{V_{max} + V_{min}}{2},$$

$$S \leftarrow \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & if \ \ (L < 0.5), \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & if \ \ (L \geq 0.5), \end{cases}$$

$$H \leftarrow \begin{cases} (G - B) / (V_{max} - V_{min}) & if \ \ (V_{max} = R), \\ 120 + 60 (B - R) / (V_{max} - V_{min}) & if \ \ (V_{max} = G), \\ 240 + 60 (R - G) / (V_{max} - V_{min}) & if \ \ (V_{max} = B). \end{cases}$$

ábra 3.11: RGB-ből HLS-be történő konverzió lépései [23]

Miután a képet felosztottuk színtartományokra, azután kétféle szűrőt kell alkalmaznunk: sárgát és fehéret, mivel a sávokat jelölő vonalak ilyen színűek lehetnek. Ezeket az értékeket a 3.12-es ábrán lehet látni.

Yellow				**White**			
Channel	H	L	S	*Channel*	H	L	S
Upper	40	255	255	*Upper*	255	255	255
Lower	10	0	100	*Lower*	0	200	0

ábra 3.12: Sárga és fehér szűrő HLS értékei [23]

3.1.9. Kalman szűrő

Sok mai rendszer, mely fel van szerelve többféle szenzorral, képes ismeretlen változókra valamilyen becslést adni. A legnagyobb kihívás viszont az, hogy ezek a rendszerek még akkor is képesek legyenek az ismeretlen változók pontos becslésére, amennyiben valamilyen szintű bizonytalanság is jelen van a rendszerben. GPS esetében ilyen lehet például műholdak pozíciójának megváltozása vagy akár valamilyen atmoszférikus hatás.

A Kalman szűrő [24] egy népszerű becslő algoritmus, amely Rudolf E. Kálmán után lett elnevezve, ami képes akár a jövőbeli állapotok megjóslására is.

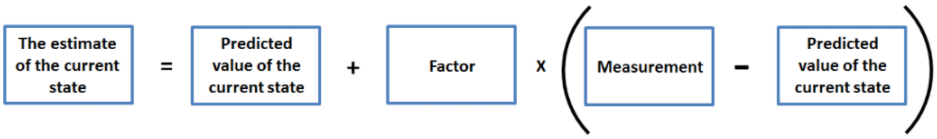
Amely a sávdetektálást illeti, ebben az esetben elegendő lesz egy egydimenziós Kalman szűrő implementálása is, így a következőkben ezt fogom majd részletesebben kifejteni.

Kalman szűrőről általánosan

A Kalman szűrő 5 fajta algoritmusra bontható, ezek: Állapot frissítés, Predikció, Súly számítás, Kovariancia frissítés, Kovariancia extrapoláció.

Állapot frissítés:

Modelje a 3.13-as ábrán, illetve a kiszámítási egyenlete pedig a 3.14-es ábrán látható.



ábra 3.13: State Update model

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n\left(z_n - \hat{x}_{n,n-1}\right) = \left(1 - K_n\right)\hat{x}_{n,n-1} + K_n z_n$$

ábra 3.14: State Update egyenlet

Ahol:

- ***X*** – magasság valódi értéke
- $\boldsymbol{Z_n}$ – súly mérési értéke ***n*** időben
- $\boldsymbol{X_{n,n}}$ – ***x*** becslése ***n*** időben (a becslés $\boldsymbol{Z_n}$ mérés után történik
- $\boldsymbol{X_{n,n-1}}$ – ***x*** előző becslése ***n-1*** időben (a becslés az előző $\boldsymbol{Z_{n-1}}$ mérés után történik
- $\boldsymbol{X_{n+1,n}}$ – ***x*** jövőbeli becslése ***n+1*** időben. A becslés ***n*** időben történik $\boldsymbol{Z_n}$ mérés után. $\boldsymbol{X_{n+1,n}}$ a becsült állapot másnéven.
- ***Kn*** = Factor, $\boldsymbol{0 \leq Kn \leq 1}$

Kn-t másnéven Kalman Gainnek is nevezik, amely minden iteráció alkalmával kiszámolódik. Ez az a súly, amivel kiegészítjük magát az aktuálisan mért adatot, és azt mondja meg, hogy mennyivel változtatom meg az aktuális becslést adott méréshez.

Predikció

Ebben a fázisban a rendszer következő állapotát határozzuk meg a 3.15-ös ábrán látható módon:

$$x_{n+1} = x_n + \Delta t \dot{x}_n$$

ábra 3.15: Prediction egyenlet

Ahol:

- $\boldsymbol{X_{n+1}}$ = következő állapot,
- $\boldsymbol{X_n}$ = aktuális állapot,
- $\Delta \boldsymbol{t}$ = két állapotfrissítés között eltelt idő,
- $\boldsymbol{\dot{X}_n}$ = sebesség

Az eddigi algoritmusaink nem számoltak semmilyen bizonytalansággal, viszont a való életben sokszor adódhatnak különböző okokból. Emiatt muszáj számításba venni ezeket, hogy korrigálni tudjuk a becsléseinket. A kovariancia frissítés és extrapoláció egyenletek határozzák meg a becslés bizonytalanságát. A fellépő zajnak a jelölése: $\boldsymbol{q}$

Kovariancia frissités

$$\boldsymbol{P_{n,n}} = (1 - K_n)P_{n,n-1}$$

Kovariancia extrapoláció

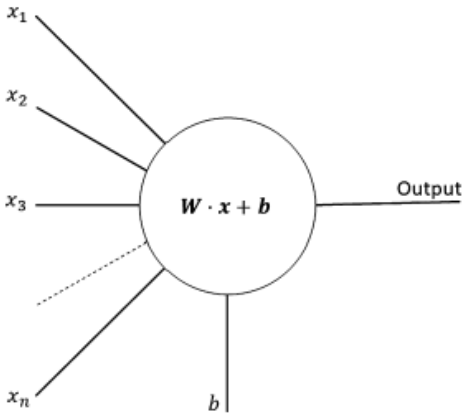
$$\boldsymbol{P_{n+1,n}} = P_{n,n} + q_n$$

3.2. Táblafelismerés

3.2.1. Neurális hálózatok

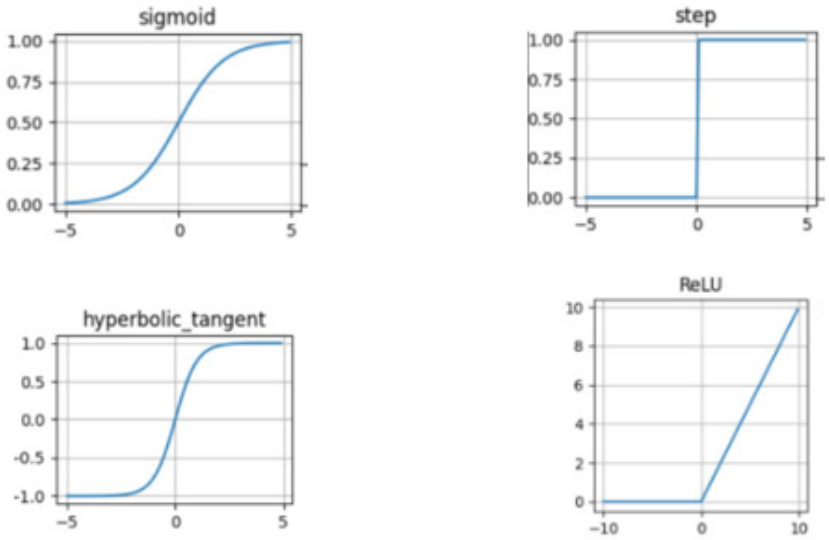
A Neurális hálózatok [25] [26] [27] olyan információfeldolgozó eszköz, melynek két fontos fázisa van: az egyik maga a betanítás, amellyel a hálózatot építjük fel, amely történhet felügyelt, illetve felügyelet nélküli módon. A betanítás során minták alapján építjük fel a neurális hálót és tároljuk el az információkat. Miután maga a hálózat felépült, akkor a második, tehát az előhívási fázisban tudjuk használni. Ez azt jelenti, hogy ebben a fázisban tudjuk felhasználni azokat az információkat, amiket korábban betanítottunk a neurális hálózatnak.

Ezen hálózatok alapja maga a neuron, ami egy műveletvégző elem, melynek több bemenete van, de csak egy kimenete, illetve rendelkezhet akár memóriával is. A bemenetek és a kimenet között valamilyen nemlineáris leképezés valósul meg. A 3.16-os ábra mutatja egy neuron felépítését.



ábra 3.16: Neuron felépítése [28]

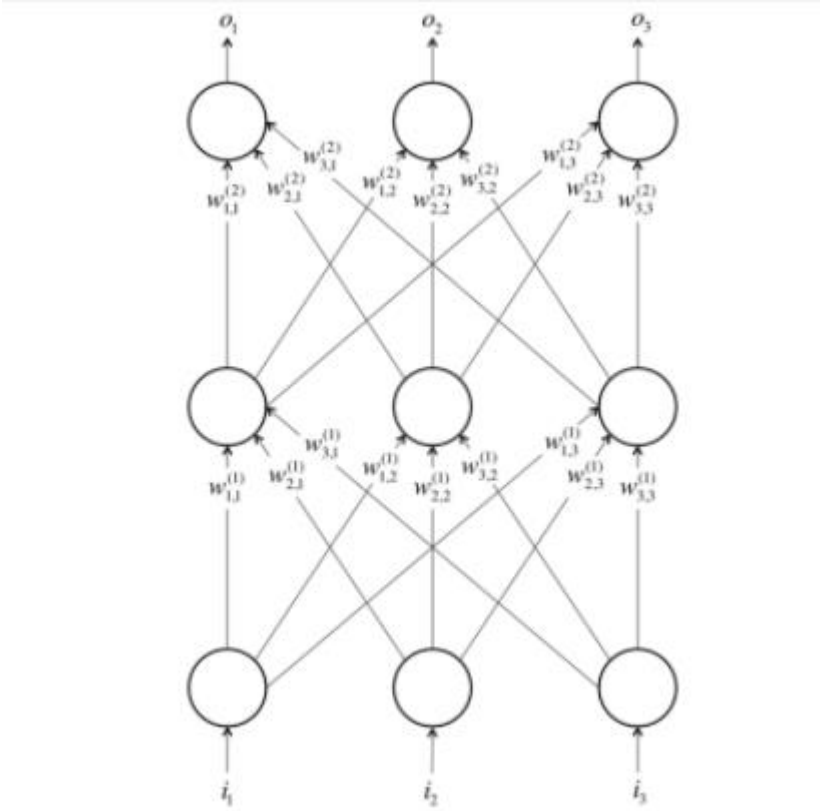
A ***W*x+b*** egyenlet a neuron logitját jelenti. ***W*** azt a súlyt jelenti, amellyel az egyes ***xi*** bemeneteket súlyozzuk, ***b*** pedig egy eltolást jelent. Ezen neuronoknak bizonyos bemenetek hatására aktívnak vagy passzívnak kell lennie. E célt szolgálja az aktivációs függvény [26], amely a bemeneti- és a tárolt értékekből előállítja az aktuális kimenetet. Azt, hogy milyen értékekre aktiválódjon a függvény az eltolás (***b***) mértéke határozza meg. Az aktivációs függvény egy nem-lineáris függvény, amely több fajta is lehet. A leggyakrabban használt függvények a szigmoid (Logisztikus), tanh (tangens hiperbolikusz), ReLU (Rectified Linear Unit), step (lépcsős). Ezeket a 3.17-es ábrán lehet látni.



ábra 3.17: Aktivációs függvények

ReLU függvény az alábbi módon írható fel: ***y=max(0, Ks)***. Ez a függvény a leggyakrabban használt aktivációs függvény.

Egy neuron önmagában nem képes komplex feladatok megoldására, emiatt több neuronra van szükségünk. A neuronok a neurális hálózatokban rétegekben helyezkednek el, úgy, mint az emberi agyban. Az emberi agykéreg hat rétegből épül fel, és az információ ezeken a rétegeken halad végig az utolsóig, ahol már képesek vagyunk megérteni azt. Például az alsó réteg a szemünkből jövő vizuális információt kapja meg, innen az információ eljut az utolsó rétegig, ahol már tudjuk, hogy mit is látunk valójában, autót vagy repülőt. A 3.18-as ábra egy három rétegű neurális hálózatot reprezentál.



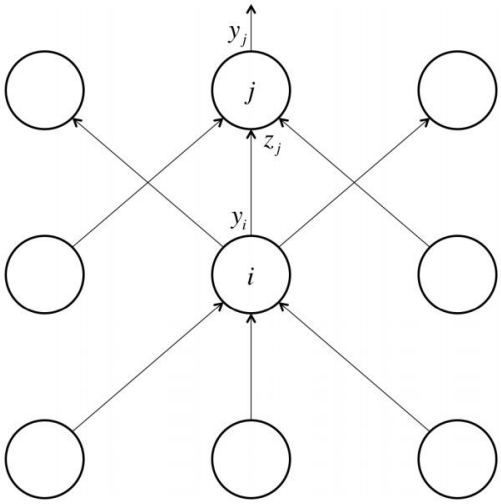
ábra 3.18: három rétegű neurális hálózat [25]

A legalsó réteg fogadja a bemeneti adatokat. A középső réteg egy rejtett réteg, ahol a neuronok bemenetein már megjelenik a $\boldsymbol{W}$ súly. Ezen súlyoktól függ maga az a vektor paraméter, ami meghatározza egy adott problémára a megoldást. Ahogy a 3.18-as ábra is mutatja, az adatok lentről haladnak felfelé, nincs lehetőség fordított adatfolyamra, továbbá egy rétegben lévő neuronok között nincs kommunikáció. Az ilyen hálózatokat nevezzük előrecsatolt neurális hálózatoknak [25]. A 3.18-as ábrán látható neurális hálózat teljesen összekötött, hisz mindegyik neuron mindegyik kimenete csatlakozik a felette lévő neuronok mindegyik bemenetére, és mindegyik rétegben ugyanannyi neuron található.

A rejtett rétegekben általában kevesebb neuron van, mint a bementi, vagy a kimeneti rétegekben, hogy a hálózat szűkített bemeneti adatokkal tudjon dolgozni. Továbbá nem szükséges mindegyik neuron kimenetét a felette lévő mindegyik neuron bemenetéhez kapcsolni. Ez általában tapasztalat útján derül ki. A neuronok kimenetei, illetve bemenetei vektorokból állnak. Ennek következtében egy neurális hálózatot képesek vagyunk vektor, illetve mátrix műveletekkel leírni.

Az ilyen hálózatok betanításakor leggyakrabban használt algoritmus az úgynevezett backpropagation [25]. A lényege az, hogy nem tudjuk pontosan, hogy a rejtett rétegeknek mit is kellene csinálnia, de azt megtudjuk nézni, hogy milyen gyorsan változik a hiba, miközben változtatunk egy rejtett aktivitáson. Ebből pedig meg tudjuk

nézni, azt is, hogy milyen gyorsan változik a hiba, akkor, ha csak egy kapcsolatnak a súlyát (***W***) változtatom. Ezt nevezzük hiba gradiensnek.



ábra 3.19: Rétegek [25]

A 3.19-es ábrán látható ***y*** egy neuron aktivitását jelenti, ***i, j*** pedig egy neuront. Ha megvannak a hiba gradiensek a ***j*** rétegben, akkor az alatta levő ***i*** réteg hiba gradienseit kell kiszámolnunk. Ahhoz, hogy ezt megtegyük előbb meg kell tudnunk hogyan hatnak az ***i*** rétegbeli neuronok kimenetei a ***j*** rétegbeli neuronokra. Miután ez megvan már kitudjuk számolni az ***i*** réteg hiba gradienseit a ***j*** réteg figyelembevételével. Ezután meg tudjuk határozni azt, hogy a súly függvényében hogyan változik maga a hiba, így meg tudjuk mondani, hogy a súlyokat hogyan kell változtatni mindegyik tanulási példa után ahhoz, hogy egyre jobb eredményt kapjunk.

Képfelismerésnél nem tudjuk 100%-ra megmondani, hogy például egy képen milyen tábla is látható. Ennek érdekében szükséges lehet a kimeneten egy valószínűségi eloszlást átadni, hogy meg tudjuk melyik tábla szerepel a legnagyobb valószínűséggel a képen. Erre használják a softmax függvényt, avagy softmax kimeneti réteget. Egy neuron softmax retégbeli kimenete a többi neuron kimenetétől függ, ami szintén a softmax rétegben található, mivel az eloszlások összege 1-gyel egyenlő. Ez a függvény megadható 3.20-as ábrán látható módon.

$$\mathrm{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}.$$

ábra 3.20: softmax függvény [27]

3.2.2. Tanulás folyamata

Egy tanuló algoritmus [27] adatokból képes tanulni. Három fő részből áll: feladatokból, teljesítmény mérésekből és tapasztalatokból. A feladatok az a problémák, amelyeket meg szeretnénk oldani a gépi tanulás segítségével. Azt mondja meg egy feladat, hogy hogyan is kellene feldolgozni egy példát. A példa egy jellemzőkből álló halmaz, mely jellemzőket egy objektumból nyertük ki. Ezeket a jellemzőket vektorok formájában tároljuk, például egy kép jellemzői általában a képen található pixelek értékei. Többféle feladat létezik, mely megmondja hogyan kellene feldolgozni egy példát, ilyen a klasszifikáció, illetve a regresszió.

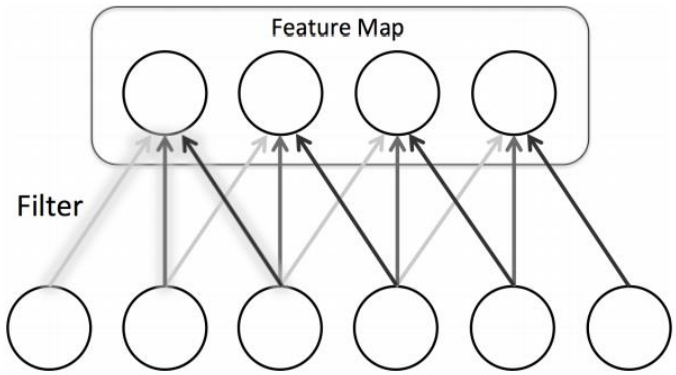
A klasszifikáció feladata az, hogy a bejövő adatokat kategorizálja, például, hogy a képen látható állat kutya-e vagy macska. Ehhez egy f függvényt használ, amely egy valószínűségi eloszlást mutat meg az osztályok között, ezzel megmutatva, hogy egy adott kép melyik osztályba tartozik a legnagyobb valószínűséggel.

A regresszió feladata az, hogy megjósoljon egy értéket egy adott bemenetre. Nagyon hasonló a klasszifikációhoz annyi különbséggel, hogy a kimenet itt más. Például egy regressziós feladat lehet egy értékpapír árának előrejelzése.

3.2.3. Konvolúciós neurális hálózatok

A konvolúciós neurális hálózat [25] [27] [29] (Convolutional Neural Network - CNN) olyan neurális hálózat, amely mátrix jellegű adatokkal dolgozik, például képekkel, amelyeket fel lehet bontani 2D-s pixel mátrixokká. A konvolúciós neurális hálózatok elnevezésében a konvolúció egy lineáris matematikai művelet jelent. Az ilyen hálózatok olyanok, mint a sima neurális hálózatok annyi eltéréssel, hogy a sima mátrixszorzás helyett konvulóciót használ a mátrixok szorzására legalább egy rétegben.

Egy konvolúció lényege, hogy veszünk egy szűrőt, amely kisebb a bemeneti képnél. Ezek a szűrők kapcsolatokat reprezentálnak, amely az egész képen replikálva van, így létrehozva az aktivációs térképeket. Ezek a szűrők úgy vannak betanítva, hogy valamilyen mintákat fedezzenek fel a bemeneti képen. A 3.21-es ábrán lehet látni a szűrőket és az ezekből létrejövő aktivációs térképet.



ábra 3.21: Szűrők és az ezekből létrejövő aktivációs térkép [25]

Ezen konvolúciós rétegek között helyezkednek el az úgynevezett pooling rétegek. Egy pooling réteg feladata az, hogy csökkentsék az előtte lévő konvolúciós réteg által generált aktivációs térkép méretét, ezzel a rákövetkező konvolúciós rétegnek kevesebb bemenete lesz, amely hatására kevesebb számítást kell megoldania. Az egyik ilyen pooling eljárás a max pooling, amely az aktivációs térképet több egyenlő cellára osztja fel, majd ebből egy sűrített aktivációs térképet hozunk létre úgy, hogy a kisebb cellákból a legnagyobb értékű cellákat választja ki. Ennek az eljárásnak az előnye az, hogy ha a bemeneti adat kicsit megváltozik, attól még a kimenet ugyanaz marad. Ez nagyon hasznos képfelismerésnél például.

3.2.4. Képfelismerés konvolúciós neurális hálózat segítségével

Táblák detektálásával és felismerésével kutatók hosszú ideje foglalkoznak. A probléma két tényezőből áll: Táblák detektálása (Traffic Sign Detection - TSD) és táblák felismerése (Traffic Sign Recognition - TSR). TSD felel a lényeges rész kinyeréséért a bemeneti képből (Region of Interest - ROI), illetve a tábla körvonalának detektálásáért. Egy hatékony TSD minden lényeges táblát meg kell találjon a lehető legkevesebb hibával. Egy hatékony TSR pedig úgy klasszifikálja a táblát az előre meghatározott osztályokban, hogy közben minél kevesebb rossz felismerést végez.

TSD-nek általában kétféle megközelítése van: Szín alapú detektálás vagy forma alapú detektálás. Szín alapú detektálás lényege az, hogy egy bizonyos szín alapján választja ki a ROI-t. A probléma ezzel a megközelítéssel az, hogy nagyon érzékeny a környezeti változásokra, mivel a környezeti hatástól függően más erősségben láthatjuk a tábla színeit. Ennek megoldására a kutatók különböző színtereket, illetve küszöbölést alkalmaznak. A HSI színtér például nem annyira érzékeny a környezeti változásokra, emiatt ezt használják leginkább szín szegmentációhoz.

Alakzat alapú detektálásnál a Hough transzformációt gyakran alkalmazzák. Annak ellenére, hogy megfelelő eredményt ad, komplex számítási feladatokat tartalmaz, amelynek nagy tárhelyre van szüksége.

Másik ilyen algoritmus a távolság transzformáció (Distance Transform - DT), amely a sarkokat keresi meg először. ***DT feature vektor*** képet úgy kapjuk meg ezután, hogy kiszámoljuk mindegyik pixel távolságát a hozzá legközelebb lévő sarokhoz. Habár ez a metódus hatékony bizonyos alakzatok detektálásában, túl sok időbe kerül a ***DT feature vektor*** kép kiszámítása, továbbá nem alkalmazható real-time rendszereken.

Annak érdekében, hogy jobb eredményeket kapjunk a szín-, és az alakzat alapú detektálást együttesen használjuk. Szín szegmentálást követően gépi tanulás segítségével történik a tábla felismerése.

Vannak olyan KRESZ táblák, amiket nem lehet csupán a színük alapján megkülönböztetni. Szürkeárnyalatos képek esetében a CNN nagyobb eséllyel ismeri fel a táblákat. Egy konvolúciós neurális hálózat működése három fő részből áll: bejövő adatok feldolgozása, tanulás, illetve predikció.

A bejövő adatok feldolgozása fázisban kell korrigálni a képet, hogy ne legyen rajta torzulás, majd TSD segítségével megkeressük a ROI-t, illetve detektáljuk a táblát.

A következő fázisban, TSR konvolúciós neurális hálózat segítségével történik maga a tanulás, illetve a kategorizálás. Minél több táblát mutatunk meg a neurális hálózatnak, annál pontosabban tudja majd megmondani, hogy egy tábla melyik kategóriába is tartozik.

A TSR-nek is általában kétfajta megközelítése van: sablon-, illetve klasszifikációs technika. Én most klasszifikációs technikával fogok foglalkozni, mivel gépi tanuláson alapszik. Először is létre kell hozni a feature vektort a képből, így csökkentve a számolási komplexitást a későbbiekben. Ezt követően az osztály címkéjét kinyerjük egy osztályozó segítségével.

A CNN-ben a konvolúciós rétegek felelnek a képek feldolgozásáért, a pooling rétegek feladata, hogy csökkentsék a feldolgozandó adatok számát a következő konvolúciós rétegnek úgy, hogy a lényeges információk megmaradjanak. A rejtett réteg(ek) feladata a klasszifikáció, TSR-hoz leginkább softmax klasszifikációt használunk, ezzel létrehozva egy valószínűségi eloszlást az osztályok felett.

A konvolúciós rétegek a CNN alapkövei. Ezen rétegek felelnek a bementi képből való lényeges információ kinyeréséért, amit különböző lineáris függvények segítségével tesznek meg. Konvolúció során egy befogadó mező (***receptive field***), melyet a maszk mérete határoz meg. Továbbá ez a befogadó mező annak a bemeneti adatnak a mérete, amelyet egy neuron vagy egy egység képes feldolgozni. Ez a mező megy végig a bemeneti képen vagy az előző réteg kimenetén. A konvolúciós réteg maszkja (***kernel***) számítja ki a súlyok és a befogadó mezők közötti pont mátrixot a kép egy adott területén, ezzel kinyerve az adott részterület lényeges információit.

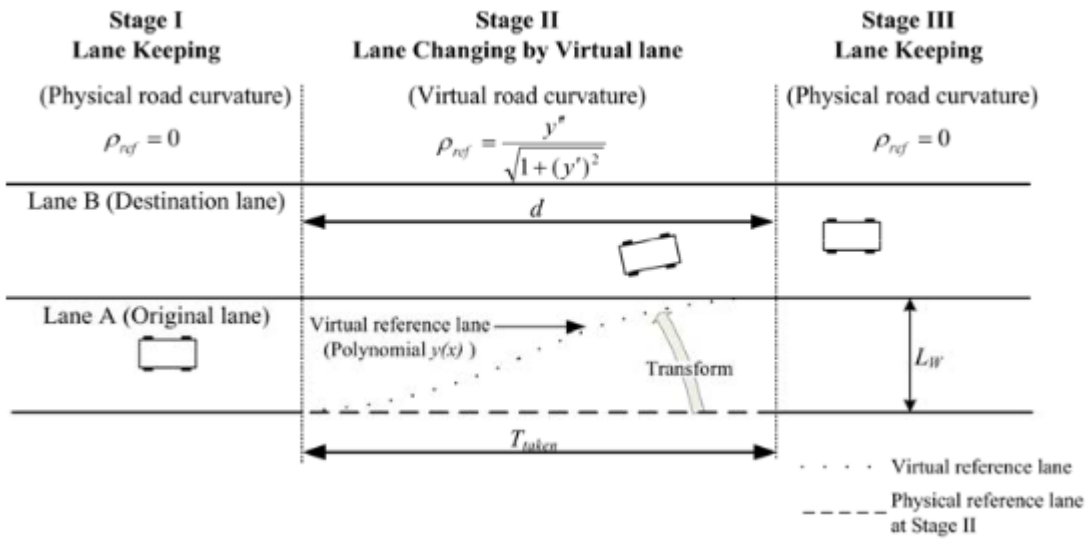
Annak érdekében, hogy kevésbé legyen érzékeny a TSR a képen lévő zajokra, muszáj alulmintavételezni a bemeneti képet. Az egyik ilyen megoldás erre a max pooling, amelynek működését lásd a 3.2.3-as szekcióban. Azonban a pooling következtében elveszhetnek olyan adatok is, melyek lényegesek a képfelismerésében, amit átfedő pooling területek alkalmazásával lehet csökkenteni.

3.3. Sávváltás

Autonóm járművek sávváltása [30] több feladatból épül fel. Először ki kell választani azt a sávot, amelyre át szeretnénk térni. Ezt követően meg kell határozni azt, hogy biztonságosan át lehet-e térni abba sávba, ami azt jelenti, hogy meg kell nézni, hogy van-e ott akadály (más jármű), vagy lesz-e ott akadály bizonyos idő elteltével. Ehhez már más szenzorok segítsége is kelleni fog, például egy lidar szenzor.

Sávváltás előtt meg kell győződnie a járműnek arról, hogy biztonságosan el tudja-e végezni a manővert. Ehhez szükséges a környezetében lévő járművek helyzetét, illetve sebességét detektálnia. A sávváltás három részre bontható, ahogy a 3.22-es ábrán

látható: Az első részben a jármű az aktuális sávban tartja magát, majd utána egy virtuális sávot generál pontokból a sávváltáshoz úgy, hogy figyelembe veszi az aktuális sáv vastagságát, a maximum oldalirányú gyorsulást és a szomszédos sáv távolságát, majd utána pedig átmegy az új sávba, és tartja azt.



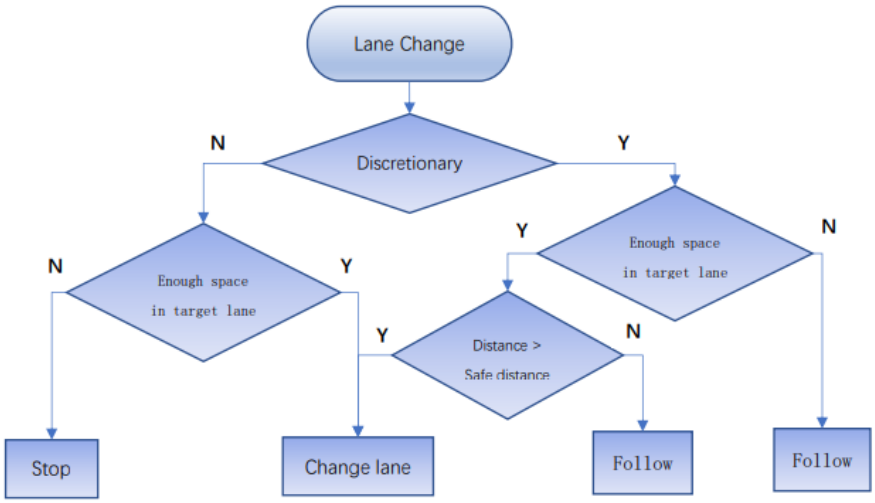
ábra 3.22: Sávváltás három része [30]

A második részben ki kell számítani a virtuális sáv görbületét annak érdekében, hogy meghatározzuk a kormányzás szögét. Továbbá szükségünk van az aktuális sáv nagyságára, a sávváltás hosszára és a maximális oldalirányú gyorsulásra, hogy megfelelően, és gördülékenyen menjen a sávváltás.

3.3.1. Sávváltási döntések

Közlekedés során a járműnek döntéseket kell hoznia [31], hogy melyik sávban közlekedjen, és mikor váltson sávot például. Ezt kétféle részre lehet bontani: kötelező sávváltás, illetve tetszés szerinti sávváltás. Kötelező sávváltásról akkor beszélünk, amikor az aktuális sáv megszűnik, vagy valamilyen akadály található az aktuális sávban, és emiatt át kell menni a mellettünk lévő sávba, ilyen például a gyorsítósávról való váltás az autópályán. Tetszés szerinti sávváltás pedig azt jelenti, hogy nem a sáv megszűnése miatt váltunk sávot, hanem például amiatt, mert az előttünk lévő jármű lassabban halad.

Sávváltáskor háromféle döntés születhet, ahogy a 3.23-as ábrán is lehet látni: megállás, követés, sávváltás. Megállni akkor kell, ha nem tud a jármű biztonságosan sávot váltani. Követés akkor lehet hasznos, ha tetszés szerinti sávváltáskor, nincsenek meg a szükséges feltételek, így kénytelen követni az előtte lévő járművet, addig amíg nem válik lehetségessé a sávváltás. Sávváltás pedig akkor fog történni, ha minden szükséges feltétel teljesül a manőver elvégzéséhez.



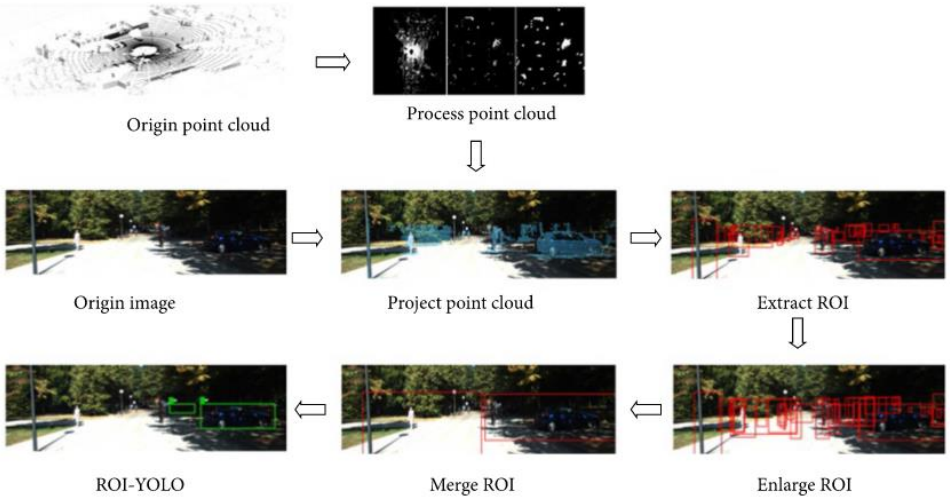
ábra 3.23: Sávváltási döntések [31]

3.3.2. Környezeti objektumok detektálása

A lidar szenzor [32] képes távolság és háromdimenziós adatok fogadására nagy távolságon belül is. Továbbá a lidar nem érzékeny a különböző megvilágítási körülményekre, emiatt nagyon széles körben használják környezet detektálásra.

Általában háromféleképpen lehet megközelíteni a problémát: Az egyik az, hogy a kamerával és a lidarral külön detektáljuk az objektumokat, majd a két eredményt egybeolvasztjuk, a másik az, hogy a kamerával detektáljuk az objektumokat, a lidar pedig csak megerősíti azokat, és a harmadik pedig az, hogy a lidar meghatározza a ROI-t, a kamera pedig detektálja az objektumokat.

A ROI-t a 3.24-es ábrán látható módon tudjuk meghatározni. Először feldolgozzuk azt a pont felhőt, amit a lidar által kaptunk. Ebből a pont felhőből meg tudjuk határozni a max-min magasságtérképet, amit majd nyolc összekapcsolt terület jelölő (***marker***) csoportosít. Utána pedig egy morfológiai tágulást alkalmaznak ezen a csoportosított magasságtérképen. Ezt követően mindegyik összekapcsolt területen egy minimum téglalap kerül kiszámításra, majd ezen téglalapok pontjai kivetítésre kerülnek az eredeti képre. A kivetítés után pedig megkapjuk mindegyik téglalap széleinek koordinátáit. Ezeket a téglalapokat majd kinagyítjuk és egybeolvasztjuk, ezzel megkapva a ROI-t. Ezt követően pedig már csak egy osztályozóra van szükség az objektumok detektálására.



ábra 3.24: Lidarral történő ROI meghatározás [32]

Bővebben kifejtve, az első lépés a grid térkép létrehozása. Mivel a lidar szenzor nagyon sok pontot ad vissza másodpercenként, ezért erőforrás megkímélése érdekében max-min magasság térképet használva építjük fel a környezetet. Egy MxN-es grid térképet hozunk létre, ahol a bal alsó sarok az origó és mindegyik cella egy négyzet. A lidar koordináta rendszerből grid koordináta rendszerre való leképezés a 3.25-ös ábrán látható:

$$P_m = \left\lfloor \frac{(y_l - d_f)}{d_c} \right\rfloor + 1,$$

$$P_n = \left\lfloor \frac{(x_l + (N/2) * d_c)}{d_c} \right\rfloor + 1,$$

ábra 3.25: grid koordináta rendszerre való leképezés [32]

ahol $\boldsymbol{d_c}$ egy cella oldalának a nagysága, $\boldsymbol{x_l}$ és $\boldsymbol{y_l}$ egy lidar pontot jelöl a lidar koordináta rendszerben, $\boldsymbol{P_m}$ és $\boldsymbol{P_n}$ pedig a grid pontot jelölik grid koordináta rendszerben.

Az objektumokat pedig úgy tudjuk észlelni, hogy megnézzük a legnagyobb és a legkisebb magasság közti különbséget, és ha nagyobb, mint a beállított küszöbérték, akkor objektumot találtunk. Minél távolabb van egy objektum, annál nehezebb azt szkennelni, emiatt használunk morfológiai tágítást, ami kiszélesíti azt a területet, ahol objektum található, ezzel könnyebben meghatározva a ROI-t.

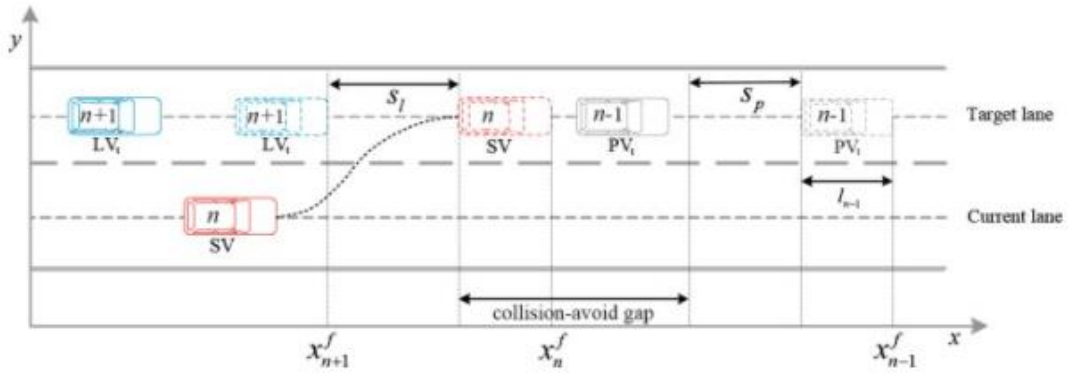
Ezután csoportosítjuk a grideket nyolc összekapcsolt terület markerrel. Ha egy objektum cella össze van kapcsolva az őt körülvevő nyolc cella valamelyikével, akkor ugyanahhoz az objektumhoz tartoznak, amit ugyanazzal a számmal jelölnek.

Ezt követően vetítjük ki a grid térképet az eredeti képre. Ezt úgy tudjuk megtenni, hogy visszakonvertáljuk a grid pontokat lidar pontokká, majd pedig kinyerjük az összes objektum pontot a pontfelhőből. Majd utána kivetítjük ezeket a pontokat az eredeti képre.

Ezután pedig meg kell határozni a ROI-t. Az objektumok koordinátákból ki tudjuk nyerni az objektumok horizontális, illetve vertikális koordinátáit az eredeti képen. Majd ezen koordináták segítségével téglalapokat hozunk létre, mindegyik téglalap egy objektumot jelöl ki, vagy az objektum egy részét, mivel minél távolabb van egy objektum, annál nehezebb észrevenni, így előfordulhat, hogy csak egy része kerül detektálásra, ami kihat a klasszifikációra. Emiatt szükséges az egy objektumhoz tartozó ROI-k felnagyítása, majd az átlapoló ROI-k egybeolvasztása. Ezután pedig klasszifikáció segítségével képesek vagyunk beazonosítani az objektumokat.

3.3.3. Baleset elkerülés

Annak érdekében, hogy a sávváltás biztonságos legyen [33], a járműnek megfelelő távolságot kell tartson a szomszédos sávban lévő előtte, illetve mögötte haladó járművek között. A 3.26-os ábrán látható a baleset elkerülés menete. A 3.26-os ábrán látható ***SV*** az az autonóm jármű, ami sávváltásra készül, ***LV*** pedig az a jármű, ami mögötte van a szomszédos sávban, ***PV***, ami pedig előtte. A pontozott járművek mutatják meg, hogy hol lesznek a járművek, miután az autonóm jármű befejezte a sávváltást. ***LV*** és ***SV*** közti biztonságos távolságot a sávváltás végén ***Sl***, ***SV*** és ***PV*** közti biztonságos távolságot pedig ***Sp*** jelzi. ***LV*** és ***PV*** végleges pozícióját X^f_{n+1} , és X^f_{n-1} , ***SV*** végleges pozícióját pedig n jelzi.



ábra 3.26: Baleset elkerülés menete [33]

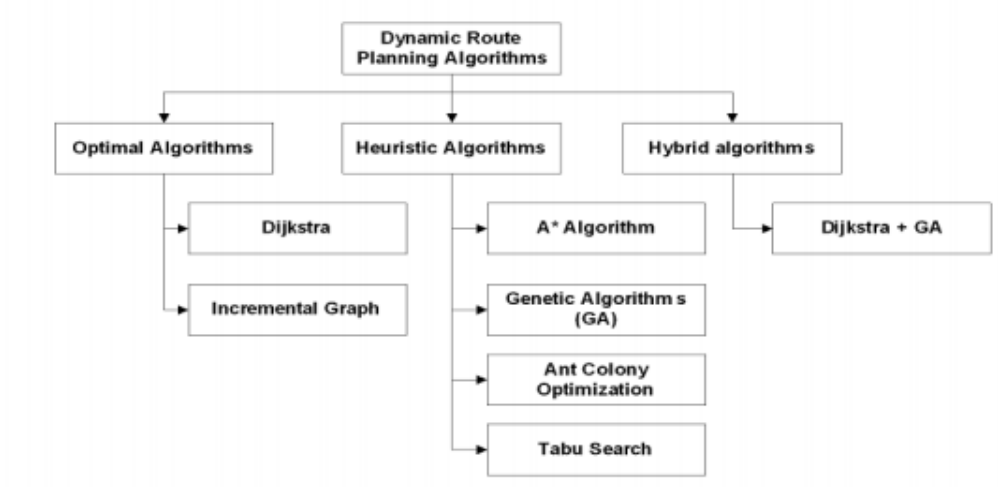
Az a pozíciótartomány, amin belül haladva sikerül a biztonságos távolságot tartani az:

$$[X^f_{n+1} + S_l, X^f_{n-1} - l_{n-1} - S_p]$$

Amíg a jármű ebbe a pozíciótartományba esik miután elvégezte a sávváltást, nem lesz baleset.

3.3.4. Path planning algoritmusok

A sávváltási út meghatározásához számos algoritmus létezik [34]. A 3.27-es ábra mutatja a különböző útkereső algoritmusokat. Léteznek heurisztikus algoritmusok, mint például A* algoritmus [37], Genetikus algoritmusok [37] [38], hangya kolónia keresés [39], illetve a tabu keresés [40]. Ezt követően van még a Dijkstra [35], illetve inkrementális gráf [36] algoritmus. Továbbá léteznek hibrid megoldások is, amelyben a Dijkstra algoritmust genetikus algoritmusokkal keverjük.



ábra 3.27: Útkereső algoritmusok [34]

Leggyakrabban a Dijkstra algoritmust, illetve inkrementális gráf algoritmust használjuk, mivel optimális útvonalat állítanak elő. A heurisztikus algoritmusok megkeresnek néhány lehetséges útvonalat, és ebből a nagyjából optimálisat választják.

A **Dijkstra algoritmus** olyan gráf alapú megoldás, amely úgy működik, hogy egy kezdőcsúcsból megtalálja az összes csúcshoz vezető legrövidebb utat egy gráfban abban az esetben, ha nincs negatív él. Az algoritmus a futása során kiválasztja azt a csúcsot, amelyhez vezető út a legrövidebb, majd azt egy prioritásos sorba rakja addig, míg nem érte el a végpontot.

Az **inkrementális gráf algoritmus** pedig úgy működik, hogy minden egyes lépésben meghatározza a következő potenciális pontot az aktuális állapot alapján, amibe beletartozik az aktuális pozíció, sebesség, környezeti információk, illetve út információk.

Az **A* algoritmus** egy heurisztikus függvényt használ a legrövidebb út kiszámításához. A hátrány az, hogy a megfelelő működés érdekében megfelelő függvényt kell választani, ami pedig nagyon nehéz, mivel egy jó heurisztikus függvény mindig ugyanazt a távolságot kellene visszaadja, és a rossz függvény esetén pedig nagyobb költsége lesz az útnak az optimális megoldásnál.

Genetikus algoritmusok működése több részből áll: genetikus kódolás, populáció kiválasztása, fitness függvény létrehozása, kiválasztási eljárás, kereszt műveletek, mátrix műveletek. Egy út egy kromoszómának felel meg, ami tartalmaz egy kiinduló pontot, egy célpontot és e kettő között lévő pontokat. A kromoszómán belüli pontokat pedig géneknek nevezzük. Populáció létrehozása történhet véletlenszerűen. Ebben az esetben létrejöhetnek olyan utak, amelyek bejárhatatlanok, mivel valamilyen akadály található bennük. Emiatt fontos, hogy minden ilyen utat ellenőrizzünk, és a bejárhatatlan utakat pedig bejárható utakra cseréljük. A fitness függvény határozza meg, hogy egy út, vagy kromoszóma mennyire megfelelő, és csak a legmegfelelőbb utak (kromoszómák) élik túl. A keresztműveletek pedig abban segítenek, hogy a szomszédos utak jellemzőit megcserélje, amelyek a mutáció során megint változnak egy keveset, és így előáll a következő generáció. Ezek a műveletek többször megismétlődnek, tetszőleges számban, és közel optimális utat állítanak elő.

A **hangya kolónia optimalizálás** a valódi hangyák viselkedésén alapul. A mesterséges hangyák utakat keresnek és információkat cserélnek az adott út jóságáról, ami egy közel optimális út megtalálásában segíthet.

A **tabu keresés** lényege, hogy egy mohó algoritmus alapján megkeres egy lehetséges útvonalat, amit iterációk mentén próbál optimalizálni egy szomszédságon belül. Ez az algoritmus mindaddig fut, amíg nem képes jobb útvonalat előállítani.

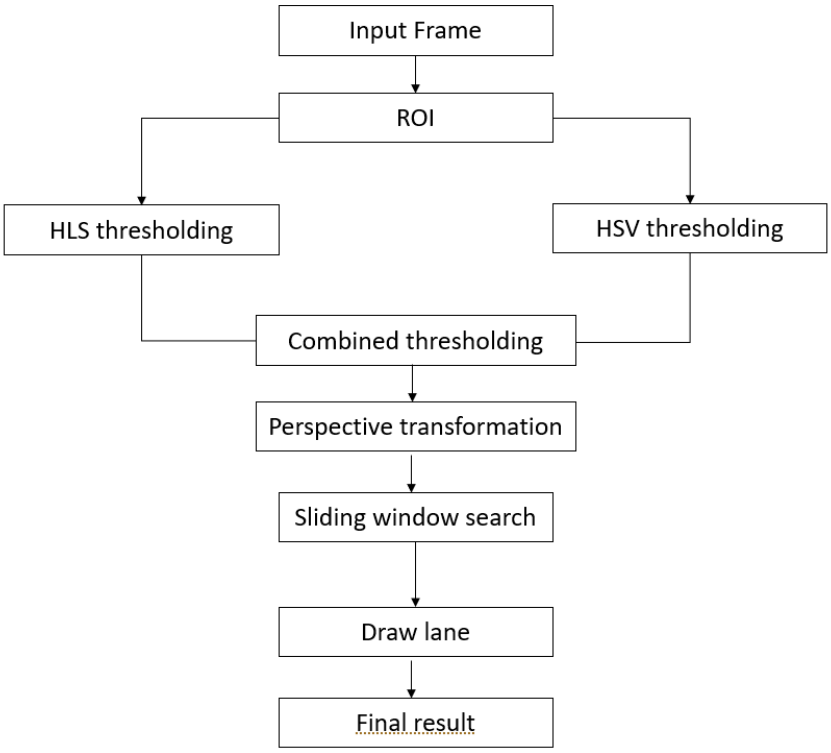
3.4. Összefoglalás

Összefoglalásként szeretném leírni, hogy mely módszereket választottam ki. Sáv detektálásához előbb ki kell nyerni a ROI-t, ezt követően a küszöböléshez HLS, illetve HSV színteret használnám fel, majd perspektív transzformációval fölülnézetes képet készítenék a sávvonalak láthatósága érdekében. Ezt követően csúszó ablakos kereséssel meghatároznám sávvonalak pixeleit, majd kirajzolnám az ezekből a pontokból készült polinómokat. Az így kapott sávot validálni szeretném a sáv különböző paraméterei alapján. Amennyiben a sáv valid, akkor a paramétereit elementeném későbbi felhasználásra, amikor a validálás nem sikerült. A tábla detektálásához, illetve a felismeréséhez konvolúciós neurális hálózatot használnék, melyeket a GTSDB, illetve a GTSRB adatbázissal tanítanék be. A GTSRB adatábázison különböző előfeldolgozást hajtanék végre, hogy több tanító adatot tudjak generálni, és hogy kiegyenlítsem az adatbázist a hatékonyabb tanítás érdekében. Ilyen előfeldolgozási lépések lennének a képeknek a forgatása, világosítása, sötétítése, illetve transzformációja. A végén pedig szeretném kiértékelni mindkét rendszert, illetve összehasonlítani más rendszerekkel.

4. IMPLEMENTÁCIÓ

Maga a rendszer két modulból épül fel: az egyik a sávdetektálásért, a másik a közlekedési táblák felismeréséért felelős. Mindkettőt Python 3.9 nyelven implementáltam. Először a sávdetektáló modul implementációját szeretném ismertetni.

4.1. Sávdetektáló modul



ábra 4.1: Sávdetektáló modul felépítése

Első lépésként, ahogy azt a 4.1-es ábra is mutatja, meg kell határoznunk a Region Of Interestet (ROI), tehát azt a területet az adott képkockán, ahol az aktuális sáv megtalálható, annak érdekében, hogy majd a későbbiekben kevesebb adatot kelljen feldolgozni.

Ahogy a 4.2-es ábrán is látható, egy trapéz formájú terület került kinyerésre, amely a bemeneti kép aljától vett ***15%***-kal fentebb kezdődik, annak érdekében, hogy az autó motorháztetője is ki legyen takarva. Továbbá a trapéz közepétől a bementi kép ***55%***-ig tartó háromszög alakú részlet is ki lett takarva, hogy az esetleges úthibák, amelyek rontanának a detektálás pontosságán, ki legyenek takarva szintén.

ábra 4.2: ROI kinyerése

Ezt követő lépésben jön a kép binárissá való átalakítása. Ez amiatt szükséges, hogy a lehető legtöbb felesleges információt eltávolítsuk a képről, és csak a sávvonalakat lehessen látni. Ehhez kétfajta küszöbölési algoritmust alkalmazok, melyeket aztán kombinálva kapjuk meg a végső kimeneti bináris képet. Ez a kétfajta technika a HLS, illetve a HSV küszöbölés [41]. HLS küszöbölésnél, ahogy a 3.12-es ábra is mutatja, a fehér szín küszöböléséért az ***L-csatorna*** felelős, amelynek a küszöbértékét ***210***-re állítottam. A HSV küszöbölésnél pedig ***220***-as küszöbértéket állítottam be a ***V-csatornára***. A HSV küszöbölés nagyon hasonlít a HLS-re, egyik szembetűnő különbség az, hogy ***L*** (lightness) helyett ***V*** (value) komponens van, ami a brightness-nek felel meg, így a HSV érzékenyebb a fényváltozásokra. Ezeket a beállításokat a 4.3-as ábrán lehet látni.

```
# Thresholding settings
settings = [{'cspace': 'HLS', 'channel': 1, 'clipLimit': 2.0, 'threshold': 210},
            {'cspace': 'HSV', 'channel': 2, 'clipLimit': 2.0, 'threshold': 220}]
```

ábra 4.3: Küszöbülési beállítások

```
clahe = cv.createCLAHE(params['clipLimit'], tileGridSize=(8, 8))
norm_img = clahe.apply(gray)
```

ábra 4.4: Contrast Limited Adaptive Histogram Equalization

A bináris képek egyesítése előtt normalizáltam a képeket a 4.4-es ábrán látható módon. Ahogy a képen is látható az ***opencv*** beépített osztályát, a ***CLAHE*** osztályt használtam fel, ami a Kontraszt korlátozott adaptív hisztogramkiegyenlítés valósítja meg. Ennek hatására a gyengébb kontrasztú pontok fel lesznek erősítve. Ahogy a 4.4-es ábrán is látható, a ***CLAHE*** osztálynak kell egy ***clipLimit*** paraméter, ami a küszöbölési értéket jelenti. Ezt az értéket a 4.3-as ábrának megfelelően ***2***-re állítottam be mindkét színküszöbölésre. A másik paraméter pedig a ***tileGridSize,*** ami megmondja, hogy mekkora rácsokra legyen felosztva a bemeneti kép normalizálásánál. Ezt az értéket 8x8-ra állítottam.

Ezt követően pedig egyesítettem a két képet, hogy megkapjam a végső kombinált bináris képet. Ezen lépések eredményei a 4.5-ös ábrán láthatóak.

HSV küszöbölés

HLS küszöbölés

Egyesített küszöbölés

ábra 4.5: Küszöbölési lépések

A küszöbölést követően jön a perspektív transzformáció. Ennek hatására egy olyan képet kapunk, amely mintha madártávlatból készült volna, így a sávvonalakat jobban lehet majd a csúszó ablakos keresésnél detektálni. A 4.6-os ábrán lehet látni a perspektív transzformáció előtti, illetve utáni állapotot.

Perspektív transzformáció előtt

Perspektív transzformáció után

ábra 4.6: Perspektív transzformáció előtt, és után

A perspektív transzformációhoz kétféle transzformációs mátrix kell: egy forrás, illetve cél transzformációs mátrix. A forrás transzformációs mátrix egy trapéz pontjait jelöli az eredeti képen, míg a cél transzformációs mátrix pedig egy négyzetet a transzformált képen, amibe transzformálni szeretnénk a képet. A forrás, illetve a cél mátrix elemei az 4.1-es táblázatban láthatóak.

Forrás mátrix	Cél mátrix
[100, 720] - Bal alsó sarok	[320, 720] - Bal alsó sarok
[475, 375] -Bal felső sarok	[320, 70] -Bal felső sarok
[805, 375] – Jobb felső sarok	[960, 70] – Jobb felső sarok
[1180, 720] – Jobb alsó sarok	[960, 720] – Jobb alsó sarok

táblázat 4.1: Forrás, illetve cél matrix értékei a perspektív transzformációhoz

A forrás mátrix elemeit a 4.7-es ábrának megfelelően hoztam létre. Van három darab változó: ***xOffsetBottom***, ***xOffsetMiddle***, ***yOffset***, melyeknek az alábbi értékeket adtam: ***100***, ***475***, ***375***. Ez a három változó fogja meghatározni a forrás mátrix elemeit, amely egy trapézt jelöl, és ezt figyelembe véve határoztam meg a három változó értékét. Az ***yOffset*** a bal felső és a bal alsó sarok ***y*** koordinátáját határozzák meg, az ***xOffsetBottom*** a bal alsó, és a jobb alsó sarok ***x*** koordinátáját határozza meg, ***y*** koordinátája pedig a képnek a magassága.

```
def setSource(self):
    """Source matrix"""
    xOffsetBottom = 100  # 200
    xOffsetMiddle = 475  # 595
    yOffset = 375  # 450
    sourceBottomLeft = (xOffsetBottom, self.image.shape[0])
    sourceBottomRight = (self.image.shape[1] - xOffsetBottom, self.image.shape[0])
    sourceTopLeft = (xOffsetMiddle, yOffset)
    sourceTopRight = (self.image.shape[1] - xOffsetMiddle, yOffset)

    return np.float32([sourceBottomLeft, sourceTopLeft, sourceTopRight, sourceBottomRight])
```

ábra 4.7: Forrás matrix létrehozása

A cél mátrixnál pedig kétféle változót hoztam létre, ahogy a 4.8-as ábrán is látszik: ***xOffset***, ***yOffset***, melyeknek az alábbi értékeket adtam: ***320*** – ami a kép szélességének a negyede, ***70***. Az ***xOffset*** mindegyik pont ***x*** koordinátáját határozza meg, az ***yOffset*** a két felső pont ***y*** koordinátáját, illetve a két alsó pont ***y*** koordinátáját pedig a kép magassága.

```
def setDestination(self):
    """Destination matrix"""
    xOffset = self.image.shape[1] / 4
    yOffset = 70
    destinationBottomLeft = (xOffset, self.image.shape[0])
    destinationBottomRight = (self.image.shape[1] - xOffset, self.image.shape[0])
    destinationTopLeft = (xOffset, yOffset)
    destinationTopRight = (self.image.shape[1] - xOffset, yOffset)
    return np.float32([destinationBottomLeft, destinationTopLeft, destinationTopRight, destinationBottomRight])
```

ábra 4.8: Cél matrix létrehozása

Magát a transzformációt két további lépésben hajtom végre. A forrás és a cél mátrixból kiszámítom az ***M*** transzformációs mátrixot. Ezt követően pedig az ***M*** mátrix felhasználásával transzformálom a bemeneti képet. Ezeket a lépéseket az **opencv** könyvtár beépített függvényeinek segítségével oldottam meg.

Miután megvan a transzformált bináris kép, akkor olyan képet kell látnunk, amelyen látszódnak a sávvonalak úgy, mint a 4.5-ös ábrán. Ezt követően jöhet az úgynevezett csúszó ablakos keresés, hogy meghatározzuk a bal, illetve jobb sávvonal pontjait a képen. Mindkét sávvonalhoz ***9-9*** csúszó ablakot helyeztem el, melyeknek magassága egyenlő a bementi kép magassága osztva ***9***-cel. A szélességük pedig kétféle érték lehet, attól függően, hogy az előző sávot jól, vagy rosszul detektáltuk. Ha jól detektáltuk, akkor ***25 pixel*** széles lesz, ha pedig rosszul, akkor ***50 pixel*** széles, ezáltal szélesebb területen nézi, hogy van-e sávpont, vagy sem.

A csúszó ablakos keresésben első lépésben meg kell határozni a bemeneti bináris kép hisztogramját. Ehhez a kép pixeleit oszloponként összegeztem, így a végén egy ***1x1280***-as vektort kaptunk. Ezután két részre osztottam fel a hisztogramot a középpontja mentén, hogy a bal, és a jobb sáv kezdő pozícióit meg tudjam határozni. Ezt úgy tettem, hogy mindkét részen meghatároztam a legnagyobb elem pozícióját. Természetesen előfordulhat, hogy a kezdőpozíciók tévesen lettek meghatározva, ami pedig gondot okozhat a későbbiekben. Ezt úgy orvosoltam, hogy téves meghatározás esetén egy adott pozícióba helyezem az adott sáv kezdőpozícióját. Ezek az értékek pedig a középponttól ***-300*** pixel a bal sávvonalra nézve, és ***+200*** pixel pedig a jobb sávvonalra. Abban az esetben van téves kezdőpozíció bal sáv esetén, ha ez az ***x*** koordináta kisebb, mint ***200*** pixel, jobb sáv esetén pedig nagyobb, mint ***1000*** pixel. Ezt követően kibővítettem a bemeneti bináris kép dimenzióit, hogy legyen egy plusz dimenzió az RGB-csatornának is, így majd lehetővé válik a csúszó ablakok vizuális megjelenítése.

A bemeneti bináris képen sok olyan pont van, ahol nincsenek pixelek, így ezeket nem kell vizsgálnunk, csak azokat az ***x, y*** koordinátákat, ahol a pixelérték nem nulla. Ezeket a koordinátákat a ***numpy.nonzero()*** beépített függvénnyel nyerem ki. Ezen belül is csak azokat a pontokat kell vizsgálni, amelyek benne vannak a csúszó ablakokban. Ehhez végig kell iterálni az ablakok számán, és az aktuális iterációban vizsgálni kell a jobb és a bal oldali ablakot, hogy mely pontok vannak ezen belül. Ezen pontoknak az ***x*** koordinátáját eltárolom egy tömbben, mivel a későbbiekben azt is meg kell határozni, hogy ezen kinyert pontok közül melyek jelölnek tényleges sávpontot.

A valós sávpontok meghatározását, vagyis a csúszó ablakok újra pozicionálását a ***validateWindow*** metódusban hajtom végre. Ahhoz, hogy egy adott ***x*** koordinátán sávpont legyen minimum ***200 pixelt*** kell találni. Természetesen ez a feltétel még nem biztosítja azt, hogy valóban csak a tényleges sávpontokat szűrjük ki, mivel adódhatnak olyan esetek is, amikor például a zaj nagyon közel van a sávponthoz, ugyanolyan vagy akár nagyobb intenzitással. Tehát azzal is foglalkozni kell, hogy ezeket a kívülálló pontokat kiszűrjük. Ehhez egy egydimenziós Kalman szűrőt használtam, a ***filterpy***

könyvtárból. Az állapot váltó mátrix, a mérési zaj, a kovariancia mátrix, a mérési függvény, illetve a feldolgozási zaj értéket a Filterpy dokumentációja [43] alapján határoztam meg. Abban az esetben, ha szaggatott a sávvonal, és két szaggatott vonal között nagy a távolság, akkor előfordulhat, hogy a következő állapot megjóslása nem lesz pontos. Előfordulhat az is, hogy nem minden ablakon belül találunk sávpontokat. Ebben az esetben arról is gondoskodnunk kell, hogy az utolsó jól pozícionált ablak ***y*** koordinátáját elmentsük, hogy ne legyenek érvénytelen koordináták a polinom illesztéskor.

A Kalman szűrő használatával is előfordulhat az, hogy olyan ***x*** koordinátát kapunk, amely túlságosan a kép szélén van. Ebben az esetben meghatározom az adott sávvonalra az átlagos ***x*** koordinátát ott, ahol található pixel, és ezt az értéket átlagolom a Kalman szűrő által meghatározott értékkel, a 4.9-es ábrán látható módon.

```
if len(good_left_inds) > minpix:
    self.left_kalmanFilter.update(rightx_current)
    lkf = int(self.left_kalmanFilter.get_position())

    if 200 < lkf < midpoint:
        leftx_current = lkf
    else:
        maxl = int(np.mean(nonzerox[good_left_inds]))
        avgl = int((lkf + maxl) / 2)
        leftx_current = avgl

    left_end = win_y_high
```

Bal oldali csúszó ablak pozícionálása

```
if len(good_right_inds) > minpix:
    self.right_kalmanFilter.update(leftx_current)
    rkf = int(self.right_kalmanFilter.get_position())
    if midpoint < rkf < 860:
        rightx_current = rkf
    else:
        maxr = int(np.mean(nonzerox[good_right_inds]))
        avgr = int((rkf + maxr) / 2)
        rightx_current = avgr

    right_end = win_y_high
```

Jobb oldali csúszó ablak pozícionálása

ábra 4.9: Bal és jobb cszúszó ablak pozícionálása

A ***validateWindow*** metódus a végén visszaadja a következő jobb, és bal oldali csúszó ablak ***x*** koordinátáját, illetve az aktuális ablakok ***y*** koordinátáját.

Ezt követően a csúszó ablakokon belül lévő pontok tömbjét hozzáadom az előző iterációban detektált pontok listájához. Ezáltal olyan listát kapunk, amelyekben különböző méretű listák vannak. Emiatt az iteráció után ezeket egyesítenünk kell. Ezt a ***numpy.concatenate*** beépített függvénnyel oldottam meg.

Ezután meg kell határoznunk a kirajzolandó sávvonalak polinomjait, amit a ***fitPoly*** metódusban implementáltam. Az egyesített listából azután pedig ki kell nyerni a bal, és a jobb oldali sávpontok ***x*** és ***y*** koordinátáit, amelyek szintén egy listába kerülnek. Ha

ezek a listák nem üresek, akkor mindkét sáv vonalra rá lehet illeszteni egy másodfokú polinomot. Ezt a ***numpy.polyfit*** metódus segítségével teszem meg, amely visszaadja a polinom együtthatóit. Ezekkel az együtthatókkal pedig ki lehet számolni mindkét polinom pontjainak tényleges koordinátáit, amit a 4.10-es ábra mutat.

```
if len(leftx) > 0 and len(lefty) > 0:
    left_fit = np.polyfit(lefty, leftx, 2)
    self.left_fitx = left_fit[0] * self.ploty ** 2 + left_fit[1] * self.ploty + left_fit[2]

if len(rightx) > 0 and len(righty) > 0:
    right_fit = np.polyfit(righty, rightx, 2)
    self.right_fitx = right_fit[0] * self.ploty ** 2 + right_fit[1] * self.ploty + right_fit[2]
```

ábra 4.10: Jobb és bal sáv polinom x koordinátáinak meghatározása

A két polinom együtthatóiból ezután meg lehet határozni a sáv különböző paramétereit, ami a sáv validációjához szükséges. Először meg kell határozni a világtérben levő két sávvonal polinom együtthatóit. Ezt úgy lehet megtenni, hogy a számítási térben meghatározott két polinom pontjainak koordinátáit megszorzom azokkal az értékekkel, amelyek meghatározzák azt, hogy adott pixel hány méternek felel meg a valóságban. Ezek az értékek ***y*** tengelyre tekintve $\frac{30}{720}\frac{méter}{pixel}$, illetve az ***x*** tengelyre tekintve $\frac{5.7}{640}\frac{méter}{pixel}$. Ezen értékek meghatározását szintén Peter Moran implementációjának [42] segítségével határoztam meg annyi különbséggel, hogy az ***x*** tengelyre történő átváltás értékeit megváltoztattam, annak érdekében, hogy a valós magyar lakott területen kívüli autóutak sávszélességét kapjam vissza, ami körülbelül ***2.75*** és ***4*** méter között van a ***11/2001. (III. 13.)*** [44] rendelet szerint.

Ezután ki lehet számolni a két polinom görbületi sugarát, amit a 4.11-es ábrán látható képlet segítségével oldottam meg.

$$R = \frac{\left[1 + \left(y'(x)\right)^2\right]^{\frac{3}{2}}}{\left|y''(x)\right|}.$$

ábra 4.11: Görbület számítás [45]

A sáv szélességének meghatározása pedig az alábbi módon történt: Kiszámítottam a jobb és a bal oldali polinom világtérben lévő ***x*** pontok átlagos távolságát.

A sugarat pedig úgy számítottam ki, hogy elosztottam a jobb és a bal oldali sáv görbületét egymással. Ezzel meg lehet határozni azt, hogy a két sávvonal görbülete, hogy aránylik egymáshoz. Ha túl nagy, vagy túl kicsi számot kapunk, akkor a két sávvonal jelentősen eltér egymástól, ellenkező esetben pedig hasonlóak.

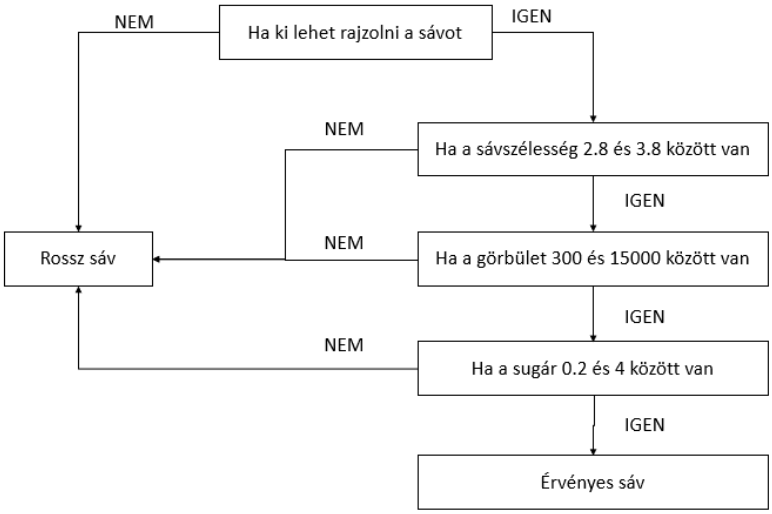
Az autó sávon belüli pozícióját pedig lineáris interpolációval határoztam meg, 4.12-es ábrán látható módon.

```
img_center = self.image.shape[1] / 2
lane_position_prcnt = np.interp(img_center, [self.left_fitx[-1], self.right_fitx[-1]], [0, 1])
lane_position = lane_position_prcnt * self.lane_width
```

ábra 4.12: Sávon belüli pozíció

A lineáris interpoláció lényege az, hogy két pont között lineáris összefüggés van, így, ha van egy harmadik pont, ami ezesetben az ***img_center,*** akkor elég egyik koordinátáját ismerni, és abból meg lehet határozni a másikat. Ebben az esetben az ***img_center*** egy ***x*** koordinátát jelöl, és azt szeretnék meghatározni, hogy ez a [0,1] zárt intervallumon belül hol helyezkedik el. Ez egy százalékos értéket jelöl, így ahhoz, hogy megkapjuk azt, hogy hány méterre vagyunk a sávban a bal oldali sávvonalhoz képest, meg kell szorozzuk a sáv szélességével.

Miután kiszámítottam a sávvonalak paramétereit, akkor ezek alapján már meg lehet határozni, hogy az aktuálisan detektált sáv valós sávot jelöl, vagy sem. Ezt másnéven ***sanity check***-nek nevezik, amit a 4.13-as ábrán látható módon implementáltam.



ábra 4.13: Sanity check lépései

Érvényes sávdetektálás esetén már csak annyi maradt hátra, hogy kirajzoljuk az eredeti képre azt. Először létre kell hozni egy üres képet, melynek a dimenziói megegyeznek a bemeneti kép dimenzióival. Ezt követően a jobb és bal oldali sávpontok ***x***, ***y*** koordinátáit kell egy tömbbe rendezni, majd ezeket egyesíteni. Ezekből az egyesített pontokból lehet majd kirajzolni a sávot jelölő trapézt az elején létrehozott üres képre. Ezután pedig inverz perspektív transzformációval át kell transzformálni a világtérbe, majd egyesíteni ezt az eredeti bemeneti képpel.

Természetesen nem mindig kapunk érvényes sávot. Ebben az esetben is gondoskodni kell arról, hogy legyen valamilyen hibakezelés. Mivel autópályán a sávok nyomvonala nem változik hirtelen, ezért érvényes sáv detektálása esetén elmenthetjük a sáv paramétereit, így hibás detektálás esetén fel lehet használni azt. Természetesen csak abban az esetben, ha a hibás detektálások száma nem érte el a ***20***-at. Ez azt jelenti, hogy körülbelül 1 másodpercig lehet az elmentett sávokra hivatkozni, ezt követően pedig eldobásra kerülnek. Korábbi állapot felhasználása esetén a sáv sárgával rajzolódik ki (4.16-os ábra), ellenkező esetben pedig zölddel (4.15-ös ábra).

Amennyiben a detektálás érvényes sávot észlelt, meg lehet határozni azt, hogy milyen szögben kell elforgatni a kormányt a sáv tartásához, és ezek alapján pedig a ki lehet rajznolni a haladási irányt is, amit David Tian blogja alapján implementáltam [46]. A forgatási szöghöz először meghatároztam az ***x*** tengely menti elmozdulás nagyságát. Ehhez a jobb és a bal oldali sávvonal legtávolabbi pontjainak ***x*** koordinátáit átlagoltam, majd kivontam ebből a kép középpontjának az ***x*** koordinátáját. Ha az így kapott eredmény negatív, akkor balra kell majd tekerni a kormányt, ha pedig pozitív, akkor jobbra. Az ***y*** tengelyi elmozdulás pedig a kép középpontjának az ***y*** koordínátája. Ezek a paraméterek 4.14-es ábrán látható kódban az ***xOffset***, illetve az ***yOffset*** változók. A kormányzási szöghöz pedig meghatároztam ennek a két változó hányadosának arkusz tangesét, amelynek az eredménye radián. Majd végül átváltottam fokba az így kapott eredményt.

```
center = int(self.image.shape[1] / 2)

xLeft = self.left_fitx[0]
xRight = self.right_fitx[0]

xOffset = (xLeft+xRight)/2-center
yOffset = int(self.image.shape[0])

angle_to_radian = math.atan(xOffset / yOffset)
angle_to_deg = int(angle_to_radian * 180.0 / math.pi)
new_steering_angle= angle_to_deg + 90
```

ábra 4.14: Kormányzási szög kiszámítása

A haladási irányt, a 4.15-ös, illetve 4.16-os ábrán is látható piros vonal jelöli. A vonal alsó részének a pontja tulajdonképpen a gépjármű középén helyeszkedik el. A felső részének az ***y*** koordinátája a kép magasságának a fele, az ***x*** koordinátája pedig az alábbi egyenlettel lett kiszámítva:

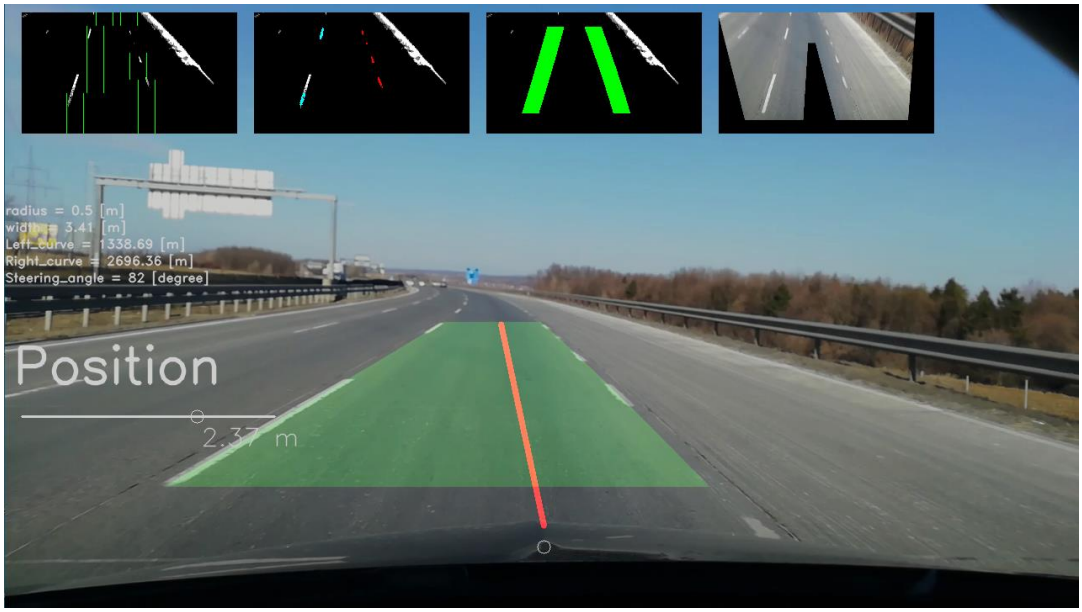
$$x = center_x - \frac{height}{2} / \tan\left(steering\ angle\ in\ radian\right)$$

, ahol a $center_x$ a kép középpontjának az ***x*** koordinátáját jelöli.

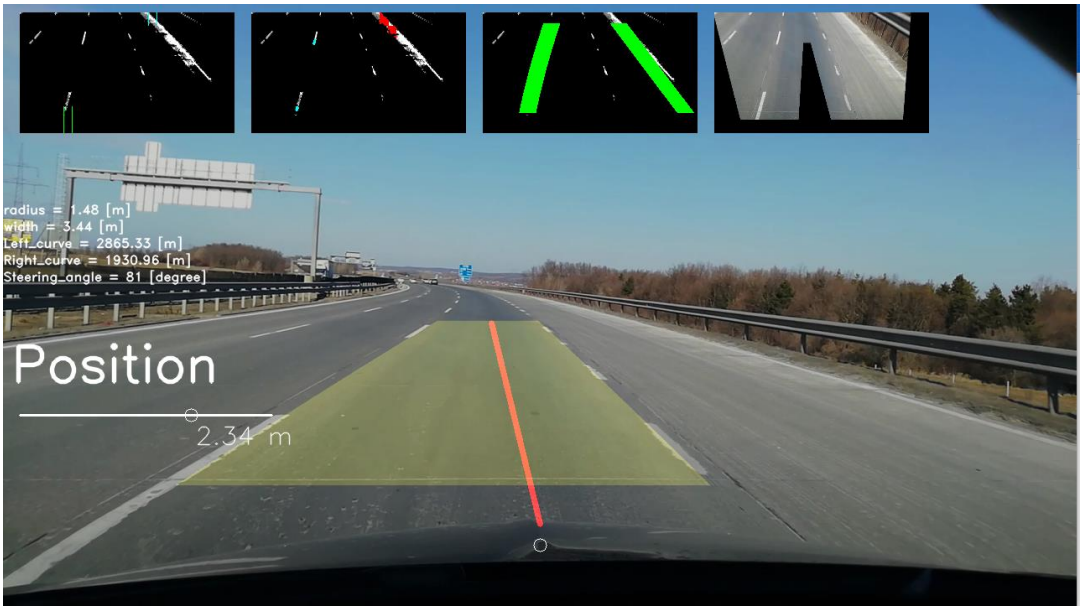
Ez a módszer viszont nem teljesen hibamentes. Előfordulhatnak olyan esetek real-timeban, amikor a kormányzás túl nagy szögben történne meg. Egy autópályán például nem lehet a kormányt túlságosan eltekerni nagy sebességnél, mert balesetet okozhat,

vagy akár az is előfordulhat, hogy a gépjármű egyik sávvonaltól a másikig menne oda-vissza. Tehát arra is figyelni kell, hogy egy új képkockán kiszámított kormányzási szög ne legyen tulságosan eltérő az előzőtől. Való életben is fokozatosan tekerjük a kormányt egy adott szögik, nem pedig hirtelen tekerünk oda. Ennek érdekében először kiszámítom az új, illetve az előző kormányzási szög különbségét. Amennyiben a különbség abszolút értéke nagyobb 3 foknál, akkor az új kormányzási szöget az alábbi módon számítom ki:

$$New\ angle = current\ angle + 3 * angle\ deviation / abs(angle\ deviation)$$



ábra 4.15: Validált sáv



ábra 4.16: Korábbi validált sáv felhasználása

Az 4.15-ös, illetve 4.16-os ábrán látható további négy alkép. Balról jobbra haladva az első a csúszó ablakos keresés vizualizációja, a másodikon a kijelölt bal, illetve jobb sávpontok láthatóak, a harmadikon az ezekre illesztett polinomot lehet látni, és az utolsón pedig a perspektív transzformáció eredményét.

4.2. Közlekedési tábla felismerő modul

4.2.1. Tábladetektálás

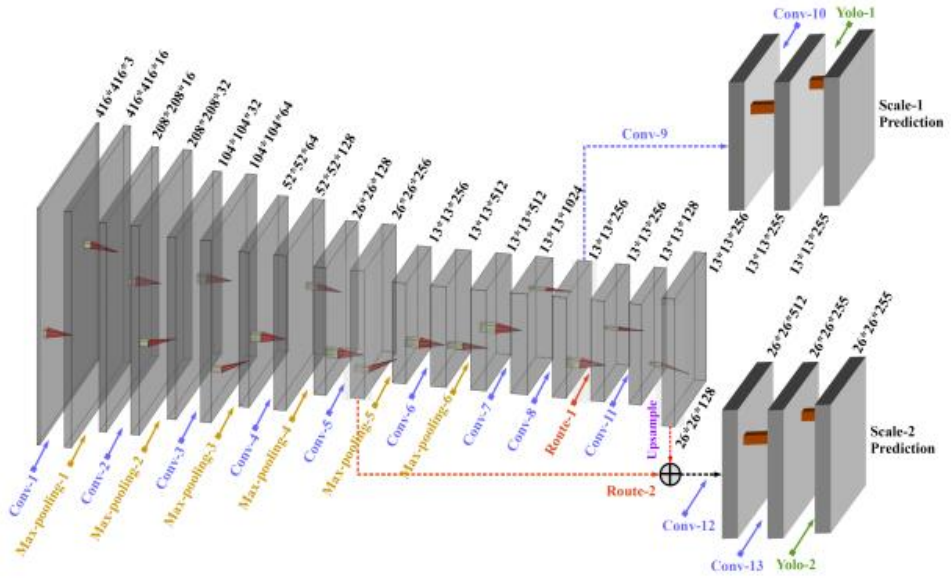
Ez a modul két részből épül fel: közlekedési táblák detektálásból, illetve felismeréséből. Először a detektálás működését szeretném ismertetni. [47]

A detektáláshoz a YOLOv3 algoritmust használtam, és ennek betanításához pedig a GTSDB adatbázist. Ez az adatbázis 900 ppm kiterjesztésű fájlt tartalmaz, illetve egy txt kiterjesztésű annotációs fájlt, amelyben benne vannak a képekhez tartozó kijelölt közlekedési táblák pontjai, illetve az, hogy melyik osztályba tartoznak. Ezen kívül még tartalmaz olyan képeket is, amelyeken nincs közlekedési tábla, de ezeket kihagyom a tanításból. Összesen négyféle osztály van: ***tiltás***, ***veszély***, ***kötelező***, illetve ***egyéb***, és ezek szerint vannak csoportosítva a különböző közlekedési táblák az 4.17-es ábrán látható módon.

Prohibitory	Danger	Mandatory	Other
speed limit; no overtaking; no traffic both ways; no trucks	priority at the next intersection; danger; bend; uneven road; slippery road; road narrows; road crossing; construction; traffic signal; snow; animals	go right; go left; go straight; go right or straight; go left or straight; keep right; keep left; roundabout	restriction ends; priority road; give way; stop; no entry

ábra 4.17: KRESZ táblák csoportosítása [47]

A modellhez a YOLOv3 tiny modellt [48] használnám, mivel ennek a modellnek a tanításához elegendő 2 GB VRAM. A YOLOv3-tiny modell 13 konvolúciós rétegből, 6 maxpool rétegből, illetve 2 yolo rétegből áll. A felépítése az 4.18-as ábrán látható.



ábra 4.18: YOLOv3 tiny modell [48]

Ez a modell egy konfigurációs fájlban van, amit majd a tanításhoz kell felhasználni. Ahogy az 4.18-as ábrán is látható a modell 416x416 pixel méretű képeket vár. A modellen kívül viszont vannak különböző paraméterek is, amelyek a darknet [49] keretrendszeren belüli tanításhoz szükségesek:

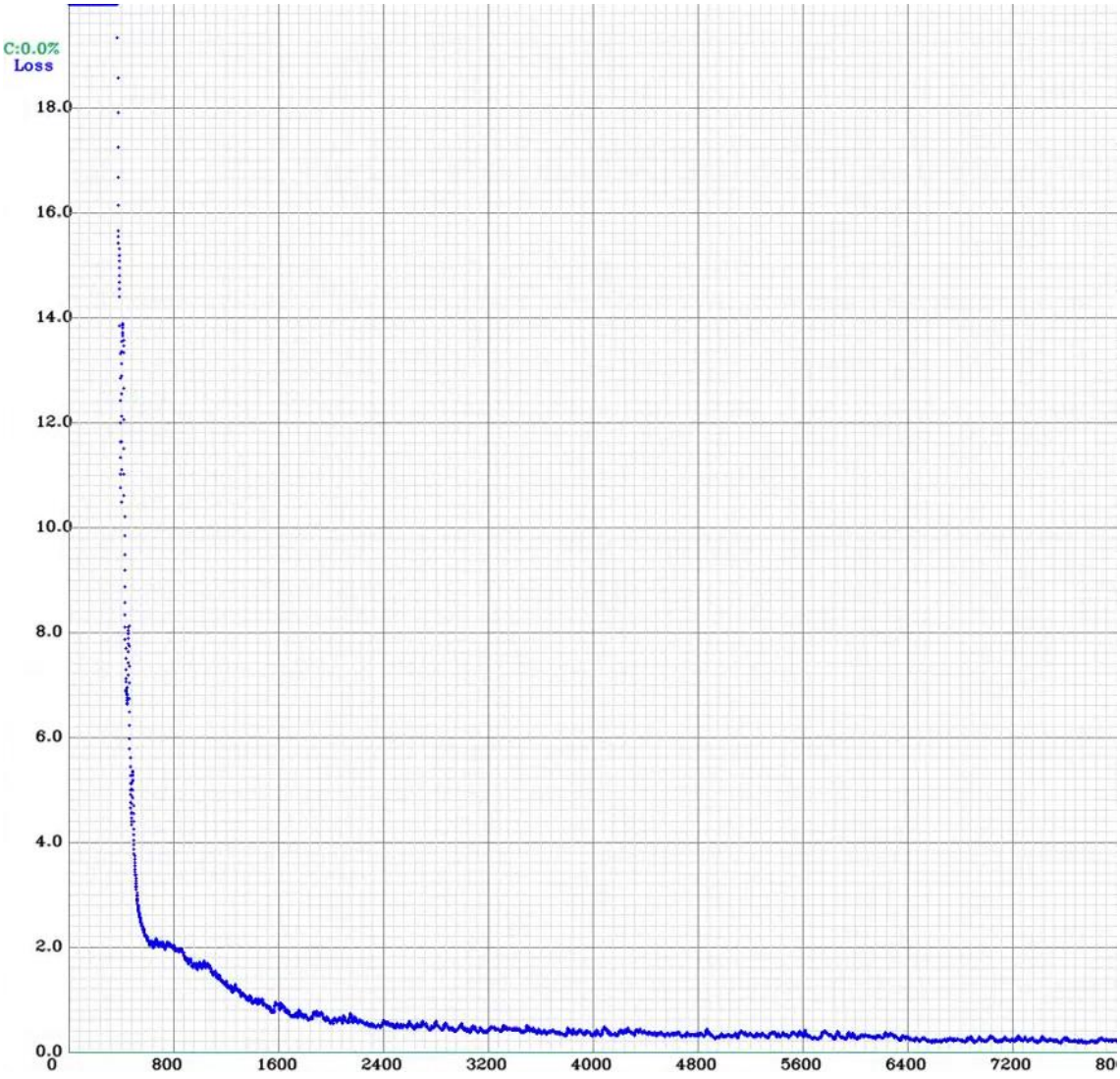
- ***Angle*** – tanítás során ezzel az értékkel forgatja a képeket véletlenszerűen, én ezt 0-ra állítottam.
- ***Saturation*** – ezzel az értékkel változtatja véletlenszerűen a képek szaturációját tanítás közben, 1.5-re állítottam ezt.
- ***Exposure*** – képek világosságát változtatja véletlenszerűen tanítás során, szintén 1.5-re állítottam ezt.
- ***Hue*** – ez pedig a színárnyalatokat változtatja véletlenszerűen a tanítás során, amit 0.1-re állítottam.
- ***Batch***: 64
- ***Subdivisions***: 16, ami azt jelenti, hogy ennyi darab mini batch van 1 sima batch-en belül.
- ***Learning rate***: 0.001
- ***Learning rate decay***: 0.0005, ennyivel fog változni a learning rate az iterációk során
- ***Iterációk száma***: 8000
- ***Steps***: 6400,7200: Az iterációk számának a 80, illetve 90 százaléka

Továbbá a yolo rétegeknél a ***classes*** paramétert ***4***-re állítottam, mivel 4 osztályba vannak besorolva a közlekedési táblák. A két yolo réteg előtti konvolúciós réteg **filter** paraméterét is meg kell változtatni annak érdekében, hogy megfelelően csatlakozzanak. Ez az érték az alábbi egyenlet alapján került kiszámításra:

- filters = (classes + coordinates + 1) * masks

, ahol az osztályok, illetve a koordináták száma ***4***, a maszkok száma pedig ***3***. Maszkok számát a yolo rétegek elején lehet látni, így a ***filters*** paraméterre ***27*** jön ki.

Ezen paramétereket ValentynN. Sichkar [50] Udemy kurzusa alapján határoztam meg.A tanítás során előállt grafikont az 4.19-es ábra mutatja.



ábra 4.19: Tanítás eredménye

Tanítás során mindegy ezredik iterációnál elmentődnek a súlyok, tehát összesen 8 darab súlyfájl lesz. Természetesen nem feltétlen az utolsó súlyfájl lesz a legjobb pontosságú. Ahhoz, hogy megtaláljuk a legjobb pontosságú súly fájlt a 8 közül, meg kell néznünk az úgynevezett mean average precisiont (***mAP***) [47] mindegyik fájlhoz. Ennek kiszámítása úgy történik, hogy először kiszámoljuk az average precisiont mindegyik osztályhoz, majd ezeknek az átlaga fogja meghatározni az ***mAP***-t. Ezt a gyakorlatban ***darknet.exe***

detector map parancssal lehet megvalósítani. Az így létrejött eredményeket a 4.2-es táblázat mutatja.

Súlyfájl	Mean average precision
Yolov3_ts_train_1000	2.23%
Yolov3_ts_train_2000	58.06%
Yolov3_ts_train_3000	59.86%
Yolov3_ts_train_4000	81.68%
Yolov3_ts_train_5000	77.85%
Yolov3_ts_train_6000	79.61%
Yolov3_ts_train_7000	82.56%
Yolov3_ts_train_8000	81.16%

táblázat 4.2: Mean average precision a detektáló tanítása után

A táblázat alapján az utolsó előtti súlyfájl bizonyult a legjobbnak.

4.2.2. Táblafelismerés

A táblafelismerő modellnek a bemenete a detektáló modell kimenete lesz. A felismeréshez is konvolúciós neurális hálózatot használok, melyből 3 fajtát próbáltam ki, felépítésük a 4.4-es táblázatban láthatóak. Mindhárom neurális konvolúciós hálót Valentyn N. Sichkar [51] publikációja, illetve Udemy-s kurzusa [52] alapján határoztam meg, illetve mindhárom CNN modellt kétféle adatbázissal tanítottam be, melyek felépítése az 4.3-as táblázatban láthatóak. Ezen folyamatokat szeretném a következőkben ismertetni.

Adatbázis	Tanító minták száma	Validációs képek	Teszt képek
1. Original dataset	36288	12441	3111
2.Equalized dataset	90601	31063	7766

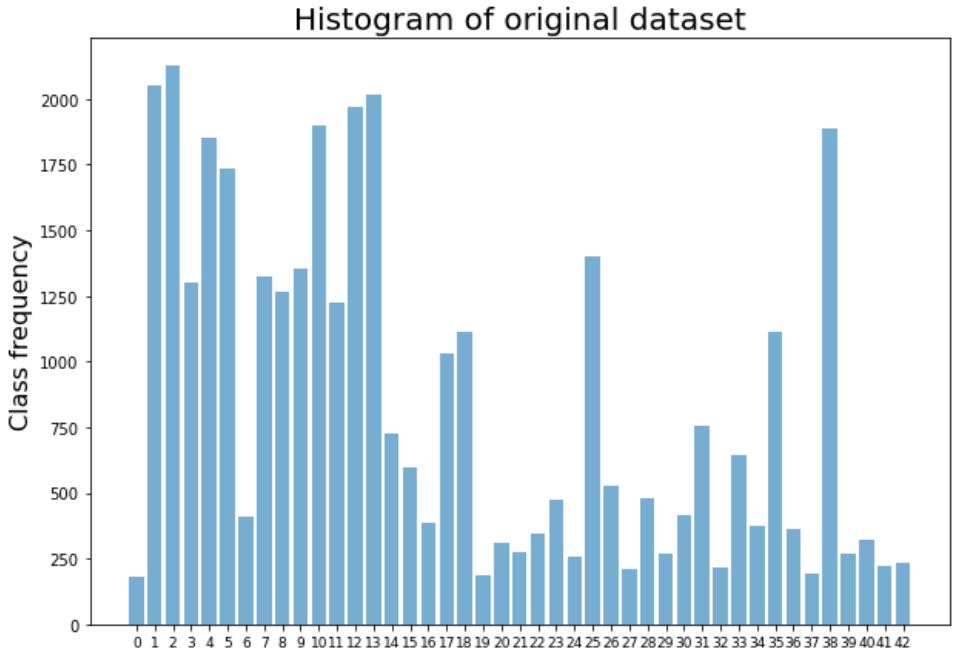
táblázat 4.3: Adatbázisok

1. Modell – RGB képekre	Input -> {8C5-P2} -> {16C5-P2} -> {32C5-P2} -> {64C5-P2} -> 128 -> 43
2.Modell – Szürke képekre	Input -> {128C5-P2-D30} -> {256C5-P2-D30} -> {512C5-P2-D30} -> {1024C5-P2-D30} -> 2048-D30 -> 43
3.Modell – Szürke képekre	Input -> {128C24-P2-D30} -> 256-D30 ->43

táblázat 4.4: CNN modellek

Ahogy a táblázatban is látható, az első modell RGB képekre lesz betanítva, a másik kettő pedig szürkeárnyalatos képekre. A 2. és a 3. modellben látható dropout réteg is, amely a túltanítást tudja enyhíteni.

A GTSRB adatbázis ***36288 darab*** képből áll, amit tanításra fogok felhasználni, illetve ***43 darab*** osztályból, viszont az osztályok közötti eloszlás nem egyenletes, mint ahogy az 4.20-as ábrán is látható. A validációs adathalmaz pedig ***31063 darab*** képből áll.



ábra 4.20: GTSRB tanító adatbázis hisztogram

Az adatbázisban mindegyik osztályhoz tartozik egy excel fájl, amiben benne van mindegyik kép neve az adott osztályban, illetve 4 darab koordináta is, ami a ROI- jelöli, tehát azt a területet, amin belül megtaláljuk az adott táblát. Későbbiekben ezeket ki kell nyerni mindegyik képből, hogy csak a releváns adatok maradjanak meg. Mivel

mindegyik kép más méretű, így a ROI kinyerése után is különböző méretű képeket kapunk, viszont mindegyik modell 48x48-as méretű képeket vár bemenetnek, így muszáj minden képet átméretezni. Ezeket a lépéseket az 4.21-es ábrán látható módon oldottam meg.

```
# get ROI vertices
x_left = dframe.loc[i, 'Roi.X1']
y_left = dframe.loc[i, 'Roi.Y1']
x_right = dframe.loc[i, 'Roi.X2']
y_right = dframe.loc[i, 'Roi.Y2']

# Getting ROI from imgs
roi_img = img[y_left:y_right, x_left:x_right]

roi_img = cv2.resize(roi_img, (48, 48), interpolation=cv2.INTER_CUBIC)
```

ábra 4.21: ROI kinyerése, és képek átméretezése

Miután az összes képet feldolgoztam az alábbi módon, utána különböző előfeldolgozási lépéseket hajtottam végre a szürke, illetve RGB képeken, melyek ***3-3*** fajta tanító aladatbázisba kerültek bele. Először normalizáltam a képeket úgy, hogy mindegyik pixelt elosztottam ***255***-tel, majd meghatároztam az átlagos pixel intenzitást, majd ezt az intenzitást kivontam mindegyik pixelből. Ezt követően pedig meghatároztam ezekből a pixelekből a szórást, mellyel aztán elosztottam mindegyik pixelt.

A második adatbázisnál háromféle módon generáltam képeket: forgatással, fényerő változtatással, illetve perspektív transzformációval. Az utóbbinál pedig kétféle cél és forrás mátrixt határoztam meg, egyik az ***x***, a második az ***y*** tengely körüli transzformációt végzi. Mindhárom traszformációnak az eredménye a 4.22-es ábrán látható.

Forgatásnál meg kellett határozni a képnek a középpontját, illetve egy forgatási szöget, amely véletlenszerűen ***-5*** és ***15*** között van. Ebből a két értékből pedig egy beépített ***opencv*** fügvénnyel meg lehetett határozni a transzformációs mátrixot, amellyel történt maga a forgatás.

A fényerő változtatásnál a bemeneti képet először HSV színtérbe konvertáltam át, mivel itt elég csak a ***V-csatorna*** értékét változtatni, ahhoz hogy a fényerő is változzon. Ennél a módszernél véletlenszerűen sötétítem, illetve világosítom a képet. Sötétítésnél ***2*** és ***6*** között változtatom a ***V-csatorna*** értékét, világosításnál pedig ***50*** és ***75*** között. Amennyiben az így kapott ***V-csatorna*** értéke kisebb nullánál, illetve nagyobb ***255***-nél, akkor ezeket az értékeket kicserélem ezekkel a határértékekre.

A perspektív transzformációnál mindkét forrás mátrix megegyezik, amely pedig a képnek a négy sarkát jelöli. Az ***x*** tengely körüli cél transzformációs mátrixnál pedig a bal felső, illetve a jobb felső sarkok koordinátáit helyeztem beljebb, az ***y*** tengely körüli

cél transzformációs mátrixnál pedig a jobb felső, illetve a jobb alsó sarkokat helyeztem beljebb.

y tengely körüli perspektív transzformáció

x tengely körüli perspektív transzformáció

Kép forgatása

ábra 4.22: Kép augmentációs technikák

Képek generálásánál mindegyik osztályhoz ki kellett számítani, hogy hány darab kép generálása szükséges. Ezt úgy oldottam meg, hogy az aktuális osztályhoz tartozó tanító minták számát kivontam a legtöbb tanító mintával rendelkező osztály tanító minta számából, és ehhez pedig hozzáadtam ***10***-et, így pedig mindegyik osztályhoz ***3010 darab*** kép tartozik. Ezt követően pedig mindegyik ciklusban kiválasztottam egy képet véletlenszerűen, amelynek a fényerejét megváltoztattam, majd pedig a korábban felsorolt módszerek egyikével transzformáltam a képet. Ez a módszer pedig az 4.23-as, illetve a 4.24-es ábrán látható.

```
# Augmenting current class
for j in tqdm(range(number_of_images_to_add)):
    # y array indexes for current class
    image_indexes = np.where(y_train == i)

    # Extracting only array itself
    image_indexes = image_indexes[0]

    # Get random image index
    n = np.random.randint(low=0, high=classesFrequency[i])

    # Getting random image from current class
    random_image = np.copy(x_train[image_indexes[n]])

    # random brightness changing
    random_image = changeBrightness(random_image)
```

ábra 4.23: Véletlenszerűen kiválasztott kép fényerő változtatása

```
# Choosing transformation technique
m = np.random.choice([1, 2, 3])

# Applying rotation around centre point
if m == 1:
    random_image = changeRotation(random_image)

# Applying perspective x axis
elif m == 2:
    random_image = perspectiveChangeAlongX(random_image)

# Applying perspective y axis
elif m == 3:
    random_image = perspectiveChangeAlongY(random_image)

# Appending transformed image into the list
x_temp.append(random_image)
y_temp.append(i)
```

ábra 4.24: Véletlenszerű transzformálási mód kiválasztása

Mindkét tanító adatbázison belül további 6 aladatbázist hoztam létre, 3-3 aladatbázis van az RGB, illetve szürkeárnyalatos képekre is, ezek:

1. Norm_dataset_ts.hdf5
2. Norm_mean_dataset_ts.hdf5
3. Norm_mean_std_dataset_ts.hdf5

Az első aladatbázisnál a pixelek ***255***-tel vannak elosztva (4.25-es ábra), a másodiknál emellett minden pixelből kivontam az átlagos pixelintenzitást (4.26-as ábra), és a harmadiknál pedig kiszámoltam a szórást a második adatbázisból, majd minden pixelt elosztottam ezzel a szórással (4.27-es ábra).

```
x_train_norm=x_train/255.0
x_valid_norm=x_validation/255.0
```

ábra 4.25: Pixel skálázása

```
# mean img calc from training ds
mean_rgb_ds_ts = np.mean(x_train_norm,axis=0)

x_train_norm_mean = x_train_norm-mean_rgb_ds_ts
x_valid_norm_mean = x_valid_norm-mean_rgb_ds_ts
```

ábra 4.26: Átlag pixelintenzitás kivonása

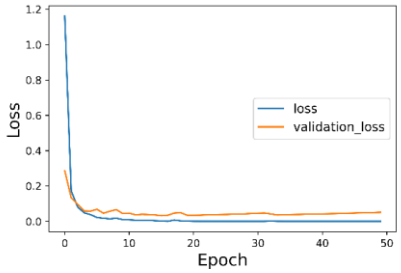
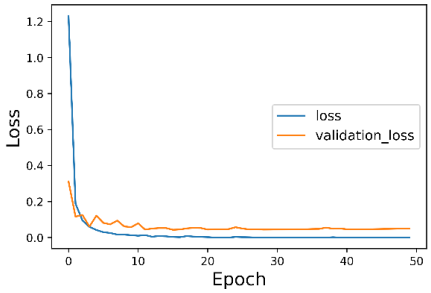
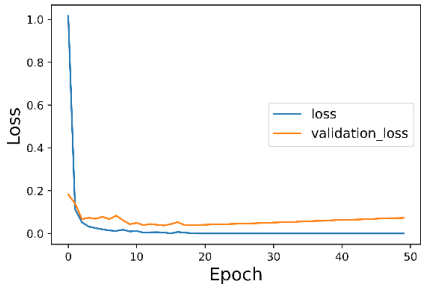
```
#standard deviation
std_rgb_ds_ts = np.std(x_train_norm_mean, axis=0)

x_train_norm_mean_std = x_train_norm_mean / std_rgb_ds_ts
x_valid_norm_mean_std = x_valid_norm_mean / std_rgb_ds_ts
```

ábra 4.27: Szórás kiszámolása

4.2.3. Tanítási eredmények

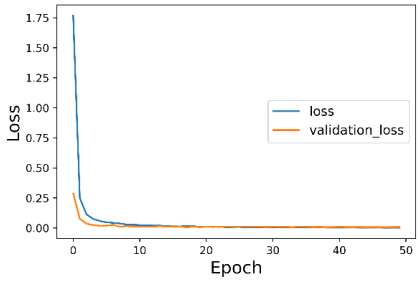
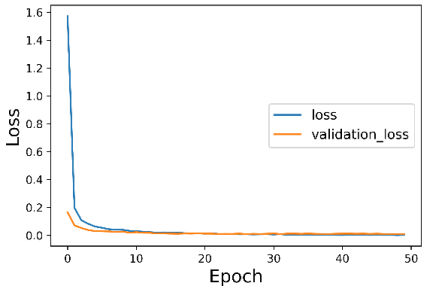
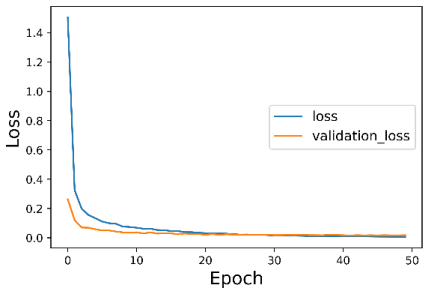
A következőkben a tanítási eredményeket szeretném ismertetni. Először az eredeti adatbázissal történő tanítás eredményeit fogom bemutatni mindhárom CNN modellre, melyek az 4.5-ös, 4.6-os, illetve 4.7-es táblázatban láthatóak.

1. Modell: RGB képekre

Aladatbázis	Losses	Accuracy	Test Accuracy
Norm_dataset		0.9947	0.9952
Norm_mean_dataset		0.9934	0.9926
Norm_mean_std_dataset		0.9954	0.9949

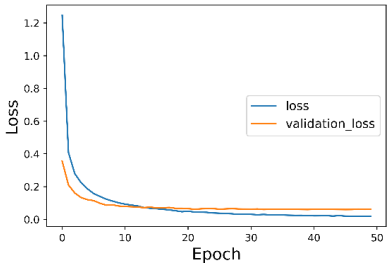
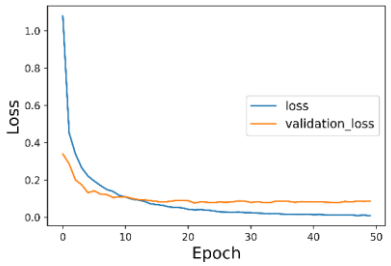
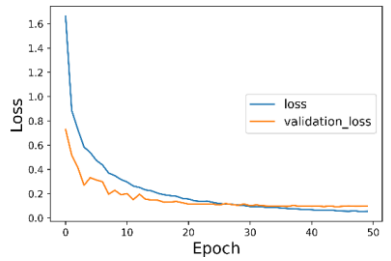
táblázat 4.5: Az 1. Modell tanítási eredményei az eredeti adatbázissal

2. Modell: Szürke képekre

Aladatbázis	Losses	Accuracy	Test Accuracy
Norm_dataset		0.9985	0.9987
Norm_mean_dataset		0.999	0.9984
Norm_mean_std_dataset		0.9965	0.9961

táblázat 4.6. A 2. Modell tanítási eredményei az eredeti adatbázissal

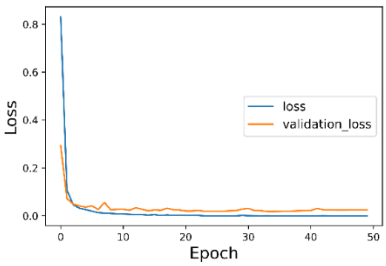
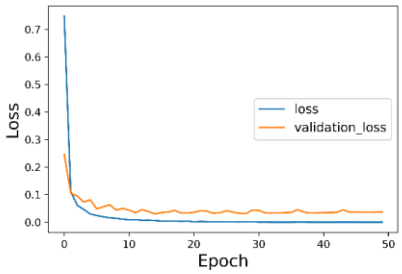
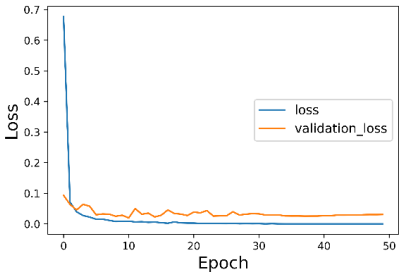
3. Modell: Szürke képekre

Aladatbázis	Losses	Accuracy	Test Accuracy
Norm_dataset		0.9908	0.9926
Norm_mean_dataset		0.9905	0.9916
Norm_mean_std_dataset		0.982	0.9855

táblázat 4.7: A 3. Modell tanítási eredményei az eredeti adatbázissal

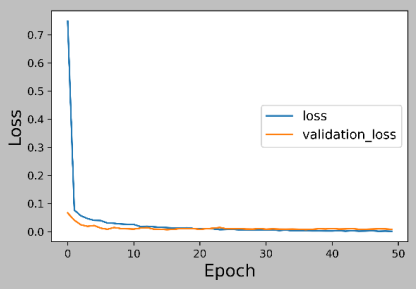
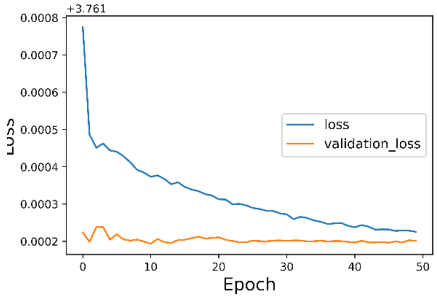
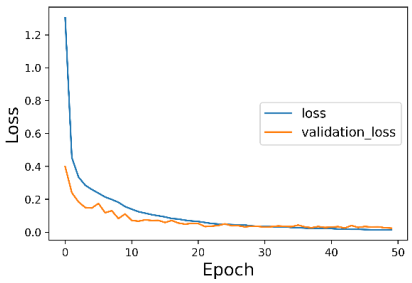
A második, kiegyenlített adatbázissal történő tanítások eredményei pedig a 4.8-as, 4.9-es, 4.10-es táblázatban láthatóak.

1. Modell : RGB képekre

Aladatbázis	Losses	Accuracy	Test Accuracy
Norm_dataset		0.9973	0.9978
Norm_mean_dataset		0.9953	0.9961
Norm_mean_std_dataset		0.9977	0.9986

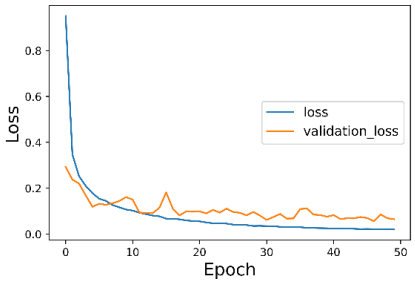
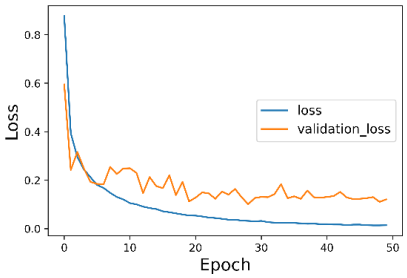
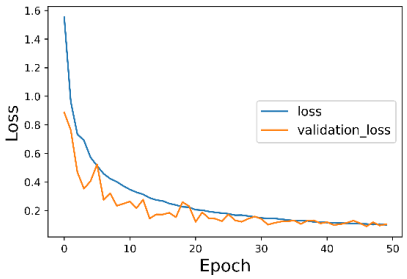
táblázat 4.8: Az 1. Modell tanítási eredményei a kiegyenlített adatbázissal

2. Modell: szürke képekre

Aladatbázis	Losses	Accuracy	Test Accuracy
Norm_dataset		0.9988	0.999
Norm_mean_dataset		0.023	0.02189
Norm_mean_std_dataset		0.9941	0.9948

táblázat 4.9: A 2. Modell tanítási eredményei a kiegyenlített adatbázissal

3. Modell: szürke képekre

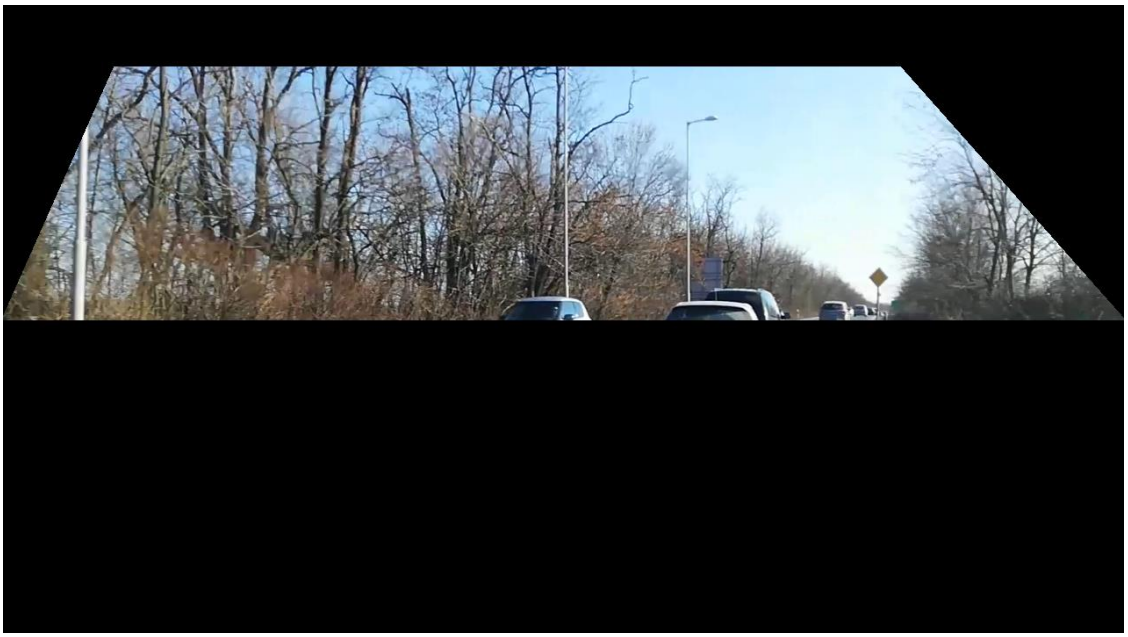
Aladatbázis	Losses	Accuracy	Test Accuracy
Norm_dataset		0.9855	0.9893
Norm_mean_dataset		0.9832	0.9826
Norm_mean_std_dataset		0.9786	0.9818

táblázat 4.10: A 3. Modell tanítási eredményei a kiegyenlített adatbázissal

Ahogy a tanítási képekből is látszik, mindegyik tanításnál ***50*** az epochok száma. Sok modellnél előjön a túltanítás is, amely azt jelenti, hogy a tanító adatokra jól működik a modell, de az általánosító képessége nem jó, tehát még nem látott adatokra nem működik olyan jól. A végső modell kiválasztásánál a túltanítás mértékét, illetve a ***test accuracy***-t vettem figyelembe. Ezek alapján a legjobbnak bizonyult modell a második, augmentált adatbázissal, ezen belül a normalizált szürke képekre betanított 2. modell, ahogy a 4.9-es táblázatban is látható.

4.2.4. Tábladetektálás, és táblafelismerés együttes használata

Legelső lépésben a bemeneti képen meg kellett határozni a ROI-t, hogy csak azokat a területeket nézzük, ahol előfordulhatnak közlekedési táblák. Ez a terület a 4.28-as ábrán látható.



ábra 4.28: KRESZ tábla detektálás ROI

Ezt követően a detektálási, majd a felismerési lépéseket Valentyn N. Sichkar [53] implementációja segítségével oldottam meg.

Ha megvan a ROI, akkor az így kapott képet olyan formává kell alakítani, amit a YOLOv3 modell elvár: RGB, 416x416-os normalizált képet. Majd az így kapott képet át lehet adni a modellnek. Ezeket a 4.29-es ábrán látható módon határoztam meg.

```
# Blob from current frame -> this preprocessing the image to be normalized, resized and converts into RGB
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True, crop=False)

# Forward pass with blob
self.network.setInput(blob)
output_from_network = self.network.forward(self.layers_names_output)
```

ábra 4.29: Előfeldolgozás, majd modellnek átadás

Az ***output_from_network*** változóban benne lesznek a kimeneti rétegek. Mindegyik rétegen belül megtalálhatóak a detektált objektumok. Ezeken végig kell iterálni és ki kell nyerni a határoló négyzetet, a detektálási valószínűséget, illetve a beazonosított osztályt, és ezeket külön tömbbe rakni. Csak azokat az objektumokat vettem figyelembe ennél a lépésnél, amelyeknek a detektálási valószínűsége meghaladta a ***0.6***-et. Mivel sok olyan határoló négyzet lehet, ami keresztezi egymást, vagy ugyanazokat az objektumokat jelölik ki, csak más méretben, így meg kell határozni a legrelevánsabbat

ezek közül. Ezt nevezik ***Non-maxima suppression***-nek, amit az alábbi módon lehet megoldani:

```
results = cv2.dnn.NMSBoxes(bounding_boxes, confidences, self.probability_minimum, self.threshold)
```

A ***self.threshold*** változó értéket ***0.2***-re állítottam, ahogy a [53] -es implementációban van. Ez az érték a non-maximum suppressionnél használt küszöbérték. A ***bounding-boxes*** tömb jelöli a határoló négyzeteket, a ***confidences*** tömb a detektálási valószínűségeket, illetve a ***self.probability_minimum*** meg a minimum detektálási valószínűséget, ami ***0.6***.

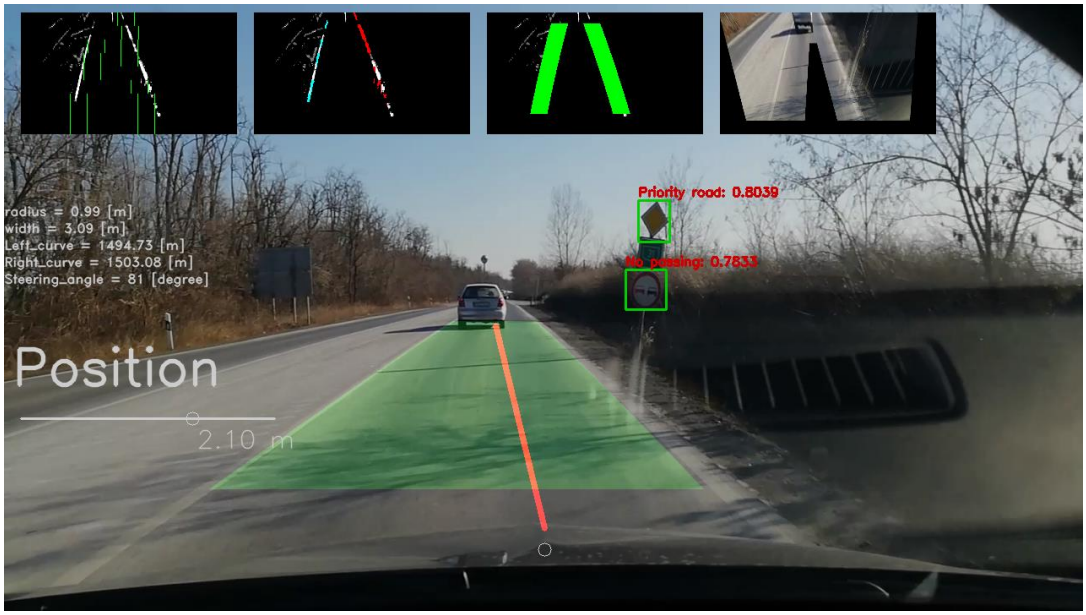
Ezt követően megkapjuk az összes detektált objektum határoló négyzetét, amely alapján ki lehet vágni a képeket, és átadni a KRESZ tábla felismerő modellnek. Az így kapott eredmény a 4.30-as ábrán látható.

ábra 4.30: Táblafelismerés eredmény

A sávdetektáló modul, illetve a közlekedési tábla felismerő modul együttes használatát a 4.31-es ábrán lehet látni, az így kapott eredményt pedig a 4.32-es ábrán lehet látni.

```
while capture.isOpened():
    ret, frame1 = capture.read()
    ret2, frame = capture.read()
    # if frame is read correctly ret is True
    frame =cv.resize(frame,(1280,720))
    frame =SignDetector.DetectSign(frame)
    frame = Process_adv(frame)
```

ábra 4.31: Sávdetektáló, illetve táblafelismerő modul használata



ábra 4.32: Sávdetektálás, KRESZ tábla felismerés eredménye

5. EREDMÉNYEK KIÉRTÉKELÉSE

Ebben a fejezetben szeretném összegezni, hogy milyen eredményeket sikerült elérni.

5.1. Sávdetektálás

Ebbben a modulban sikerült különböző képfeldolgozási lépésekkel detektálni egy adott sávot, megállapítani a sáv különböző paramétereit: szélesség, görbület, sugár. Továbbá sikerült megállapítani azt is, hogy milyen szögben kell eltekerni a kormány a sáv tartása érdekében.

A szakdolgozatomban bemutatott sávdetektáló leginkább autópályákra alkalmas, ahol az útnak a görbülete nem túl nagy, illetve az útnak a minősége is elfogadható. Továbbá az is számít, hogy milyen környezeti viszonyok között zajlik a detektálás. A teszt videó jó láthatósági körülmények között lett rögzítve az M0-ás autóúton, ahol az útnak a görbülete sem túl nagy, illetve a sávvonalak jól láthatóak. Ebből a videóból ***900 darab*** képet vizsgáltam. A teszt eredményei az 5.1-es táblázatban láthatóak. Fontos kiemelni, hogy a kombinált módszernél látható eredmények nem a felette lévő History, és Kalman szűrő módszerek összegzése, mivel a Kalman szűrő által többször kerülnek validált sávok a History-ba, így olyan detektálások is validak lesznek, melyek külön a History-val, vagy külön a Kalman szűrővel nem lennének, illetve az is előfordulhat, hogy a kombinált módszer invalid detektálást hajt végre, de lehet hogy a Kalman, vagy a History módszer külön validnak tekintette azt. Tehát úgy is lehet fogalmazni, hogy a History, illetve a Kalman szűrő eredményei nem a Kombinált módszer eredményeinek a részhalmaza.

Módszer	**Valid detektálások száma**	**Invalid detektálások száma**	**Pontosság**
Kalman szűrő	258	642	28.67%
History	492	408	54.67%
Kombinált	766	134	85.11%

táblázat 5.1: Autópályán elért teszteredmények

A tesztelés eredményeiből látható, hogy a Kalman szűrő, illetve a History adatok felhasználása jelentősen növeli a detektálás pontosságát. Tesztelési eredményekből az is megfigyelhető, hogy a kombinált módszernél a Kalman szűrő javítja a History módszert. Ennél a példánál azt lehet észrevenni, hogy ***+274 darab*** detektálás lett valid a History módszerhez képest, a Kalman szűrővel kombinálva. Ezek közül pedig ***16 darab*** képen nem detektált sem a History, sem a Kalman szűrő valid sávot, de a kombinált módszer igen.

Városi úton a detektálás pontossága alacsonyabb, mivel több tényező befolyásolhatja a sávok láthatóságát. Gyakrabban fordulhatnak elő például fák, épületek, melyeknek az árnyéka eltakarhatja oly mértékben a sávvonalak egyes részeit, hogy küszöbölésnél nem lesznek kiválasztva. Továbbá az útnak a görbülete is jelentős mértékben változhat, ami szintjén rontja a detektálás minőségét. Az ehhez készült teszt videóból, ami a 31-es főúton készült Maglód és Budapest között, ***1374 darab*** képet vizsgáltam. A tesztelés eredményei az 5.2-es táblázatban láthatóak.

Módszer	**Valid detektálások száma**	**Invalid detektálások száma**	**Pontosság**
Kombinált	933	441	67.9%

táblázat 5.2: Városi úton elért teszteredmény

A két teszt során elért átlagos pontosság így ***76.51%***. A két tesztvideón látható útvonal között számos különbség van. A 31-es főúton a sávvonalak felfestése nem túl jó minőségű, illetve sok fa is van, amelyek árnyékolják azt. Továbbá van egy kereszteződés is Pécel felé, ahol a sávvonalak adott távolságig megszakadnak, illetve két élesebb kanyar Maglód elején. Ezek a tényezők mind rontják a detektálás minőségét. Ezzel szemben az M0-as autópályán a sávvonalak felfestései sokkal jobb minőségűek, szélesebbek a sávok, és kevesebb olyan tényező van, ami ronthatná a sávvonalak láthatóságát.

A [2]-ben bemutatott módszernél neurális hálózat végzi a detektálást, és a pontosság ***96.4%,*** szóval jóval magasabb az általám bemutattot módszernél. Továbbá az is megfigyelhető, hogy mivel neurális hálózatot használtak, és a tanító mintát úgy címkézték fel, hogy az eltakart sávvonalakat is bejelölték, így rosszabb környezeti, illetve útviszonyok között is képes rendesen detektálni a sávokat. A másik lényeges különbség az, hogy az általam bemutatott módszer mindig csak az aktuális sávot képes detektálni, addig a publikációban látható megoldás az összes sávot képes detektálni egy úton.

A [3]-ban látható megoldásnál modell illetve tulajdonságon alapuló számítások segítségével detektálják a sávokat. Így egy sokkal robosztusabb rendszert hoztak létre, amely nem annyira érzékeny a zajokra, illetve a rossz láthatósági viszonyokra. Ez a megoldás ***92.3%*** pontosságú, mely szintén jóval magasabb az általam bemutatottnál.

Mindkét publikációban bemutatott megoldás pontossága növelhető lenne history adatok segítségével. Amennyiben a detektálás nem volt pontos, akkor már a korábban jól detektált sáv paramétereit fel lehetne használni az aktuális sáv kirajzolásához, de csak abban az esetben ha history adatok frissek. A 5.3-as táblázatban láthatóak összesítve a különböző megoldások pontosságai.

Megoldás	Pontosság
[2]	96.4%
[3]	92.3%
Saját	76.51%

táblázat 5.3: Sávdetektáló megoldások pontosságai

5.2. Táblafelismerés

Ebben a modulban sikerült különböző közlekedési táblákat detektálni egy képen YOLOv3 algoritmust használva, ami a Darkenettel lett betanítva GTSDB adatbázis alapján. Az így kapott detektált táblákat pedig továbbítottam a tábla felismerő CNN felé, ami így klasszifikálni tudta a bejövő táblákat. A táblafelismerőnél 3 fajta CNN modellt próbáltunk ki, melyek két fajta adatbázissal lettek betanítva: az eredeti GTSRB adatbázissal, illetve a kiegyenlített GTSRB adatbázissal. A tanítás eredményei alapján megállapítottam, hogy melyik modell, illetve melyik adatbázis bizonyolt a legjobbnak. A tesztelés során ***108 darab*** képet vizsgáltam, melyeken összesen ***5*** fajta tábla található. A 5.4-es táblázat mutatja a teszt során elért eredményeket.

Teszt képek száma	Különböző táblák száma	Detektálások száma	Helyes detektálások száma	Helyes felismerések száma	Helytelen felismerések száma
108	4	146	105	98	7

táblázat 5.4: Tábladetektálás, és táblafelismerés kiértékelése

A fentebb látható 5.4-es táblázat alapján megállapítható, hogy a detektálás pontossága ***71.92%,*** a felismerés pontossága pedig ***93.33%.*** A detektálás pontosságára jelentősen hatással vannak a különböző környezeti változók. Rossz látási viszonyok esetén a közelebb lévő táblák detektálása is problémás lehet.

A [4]-ben látható megoldásban a detektáló modulban meghatározták mindegyik osztályhoz tartozó pontosságot: ***99.27%*** a Tiltó táblákra, ***96.74%*** a Kötelező táblákra, illetve ***97.13%*** Veszélyt jelző táblákra. Az átlagos pontosság ebből adódóan pedig ***97.72%*** lett, ami jóval magasabb az általam bemutatott megoldásnál. A felismerő modul pedig ***97.75%*** pontosságú, ami nem tér el tulságosan az általam bemutatott megoldástól. Továbbá a szín alapú szegmentációnak köszönhetően sokkal robosztusabb a rendszer a környezeti változásokkal szemben is detektálás során.

A [8]-ban látható módszernél a detektálás pontossága ***90.13%,*** ami kicsivel rosszabb a [4]-ben bemutatott módszernél, a felismerés pontossága pedig ***97.43%.*** Jól látható, hogy a [4]-ben elért pontosság nagyobb, mivel a szín alapú szegmentációt követően a szülő osztályba való bekategorizálás során is neurális hálózatot, pontosabban SVM-et használtak, míg [8]-ban csak invariáns geometriai momemntumon alapuló módszert. Továbbá az is megfigyelhető, hogy a klasszifikáció pontossága is nagyobb, mivel minden szülőosztályon belül egy külön CNN modell végezte a klasszifikációt. Az 5.5-ös táblázatban látható összefoglalva a különböző megoldások pontosságai.

Megoldás	**Detektálás**	**Felismerés**
[4]	97.72%	97.75%
[8]	90.13%	97.43%
Saját	71.92%	93.33%

táblázat 5.5: Detektáló, és felismerő megoldások pontossága

6. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

A sáv detektáló modulban felmerülhet továbbfejlesztési lehetőségként az, hogy nem csak az aktuális, középső sávot detektáljok, hanem a szomszédos sávokat is, ahogy Junhwa Hur, Seung-Nam Kang, illetve Seung-Woo Seo publikációjában is látható [54]. Ez lehetővé tenné az autonóm sávváltást is, amiről korábban is írtam a 3.3.1-es fejezetben.

Ahogy korábban is írtam, kétféle sávváltás van: a tetszés szerinti sávváltás, illetve a kötelező sávváltás. Mindkét esetben 3 féle döntés születhet: vagy megáll a jármű, amennyiben a sávváltás nem végezhető el biztonságosan, vagy követi az előtte lévő járműt, amíg a feltételek nem adottak a sávváltáshoz, vagy pedig ha minden adott, akkor sávot vált a jármű.

A sávváltás megvalósításához egy path planning algoritmusra is szükség van, mely egy biztonságos utat határoz meg az aktuális és a szomszédos sáv között.

A másik továbbfejlesztési lehetőseg a közlekedésben résztvevő többi jármű detektálása, ahogy Margrit Betke, Esin Haritaoglu, illetve Larry S. Davis publikációjában is látható [55]. Ez egy elengedhetetlen funkció a biztonságos közlekedéshez, és az autonóm sávváltáshoz is, hiszen csak abban az esetben lehet végrehajtani a sávváltást, ha semmilyen jármű nem akadályozza azt.

Továbbá a sáv detektálásnak a pontosságát is lehetne növelni neurális hálózattal, ahogy a [2]-es publikációban is szerepel, így minden környezeben képes lenne sávokat detektálni, nem csak autópályán.

A közlekedési táblák detektálási pontosságát is lehetne növelni szín alapú szegmentációval, ahogy a [4]-es publikációban is látható, illetve a felismerés pontosságát is úgy, hogy mindegyik szülő osztályhoz külön CNN modellt tanítunk be. A szülő osztályba való kategorizálásért pedig külön neurális hálózat felelne.

Ezenkívül a közlekedési lámpákat is lehetne detektálni, ahogy a [56]-es publikációban is látható. Ebben a megoldásban a bejövő képet először RGB-ből normalizált RGB-be konvertálják, hogy robosztusabb legyen a különböző környezeti változásokra. Ezt követően ki kell nyerni a megfelelő ROI-kat. Ehhez háromféle szűrőt használtak az RGB értékekre, ahogy a 6.1-es ábrán is látható.

$$(R > 200 \text{ and } G < 150 \text{ and } B < 150)$$
$$\text{or } (R > 200 \text{ and } G > 150 \text{ and } B < 150)$$
$$\text{or } (R < 150 \text{ and } G > 240 \text{ and } B > 220)$$

ábra 6.1: RGB szűrők

Ezután Sobel éldetektálót használva kijelölik az így kapott területeket. Ha ez megvan, akkor Hough transzformáció segítségével kijelölik a kör alakű részeket, és meghatározzák a körön belül lévő színt, ezzel a közlekedési lámpa színét.

Összesítve, ezen fejlesztések által a jármű képes lenne sávdetektálásra, sávváltása, közlekedési táblák, és lámpák felismerésére, illetve más járművek detektálására is, melyek jelentősen növelik a jármű autonomitását is és elengedhetetlenek a biztonságos közlekedéshez. Természetesen ehhez nem csak kamera által befogadott információkat kell feldolgozni, hanem különböző szenzorok által bejövő adatokat is. Ilyen például a lidar szenszor, melyről a 3.3.2-es fejezetben írok bővebben.

7. FELHASZNÁLÓI LEÍRÁS

Ebben a fejezetben szeretném leírni a projekt futtatásához szükséges lépéseket. A projekthez ***PyCharm*** fejlesztői környezet szükséges, illetve ***Anoconda Navigator***. A szükséges környezetet az Anaconda Navigator segítségével lehet betölteni. A környezetet leíró fájlt a mellékletben csatoltom ***GPU_ENV.yml*** néven. A környezetben szerepel a tensorflow gpu könyvtár is, melyhez szükséges követelmények a tensorflow hivatalos oldalán találhatóak [57]. Amennyiben minden szükséges eszköz telepítve van és a környezet is működik, akkor lehet futtatni a projektetet (***Lane_detection_projektmunka***) a ***main.py*** elindításával. A tábladetektáló, illetve táblafelismerő modulhoz szükséges CNN modell, illetve szükséges súly és konfigurációs fájlok a projektkönyvtár ***'ts'*** elnevezésú almappájában találhatóak.

A táblafelismerő tanítása ***Jupyter Lab*** segítségével történt, melyet szintén a ***GPU_ENV*** környezet szükséges. A ***TSR_projektmunka*** projektet abba a mappába kell helyezni, ahonnan kiválasztjuk a ***GPU_ENV*** környezetet az ***Anaconda Navigator***-ben. A táblafelismerő projekthez szükséges letölteni a GTSRB adatbázist [58], amit a projektkönyvtárba kell elhelyezni ***'GTSRB'*** elnevezésű mappában. Ennek a könyvtárnak tartalmaznia kell a ***GTSRB_Final_Test_Images.zip***, illetve a ***GTSRB_Final_Training_Images.zip*** állományban lévő fájlokat. Elegendő a két tömörített állományt csak kicsomagolni. Ezt követően a ***GTSRB_Final_Test_GT.zip*** állomány tartalmát a ***GTSDB/Final_Test/Images*** mappába kell elhelyezni. A projektmappának továbbá tartalmaznia kell egy ***'ts'*** nevezetű almappát, melyben két további mappa van: ***'aug'***, illetve ***'orig'***. Ide fognak kerülni az előfeldolgozás, illetve a tanítás eredményei. A GTSRB adatbázis előfeldolgozásához a ***preprocess_tsr_dataset.py*** fájlt kell elindítani. A tanítás elkezdéséhez a ***Build_Train_TSR.ipynb*** fájlt kell elindítani. A kód elején van három változó, amelyet a tanítás előtt be kell állítani. Ezek a változók a ***folder***, ***database***, és ***subdatabase***. A ***folder*** azt mondja meg, hogy éppen melyik adatbázis almappájából kell dolgozni, a ***database*** pedig, hogy az augmentált, vagy az eredeti adatbázis van használva, a ***subdatabase*** pedig azt mondja meg, hogy melyik aladatbázissal történjen a tanítás. Mindhárom érték lehetséges értékei a változó nevek után vannak írva. Továbbá van egy ***beist_weights_filepath*** mindhárom tanítás előtt. Ez a fájl az elmentett súlyfájl nevét adja meg.

A tábladetektáló modell betanításához szükséges a ***Darknet keretrendszer***, illetve a ***YOLOv3*** rendszer is. Mindkettő telepítéséről a Darknet hivatalos weboldalán lehet bővebb információt találni [49]. Darknet tanításához szükséges két konfigurációs fájl, egy tanításhoz, egy pedig a teszteléshez. Az alapja a YOLOv3-tiny modell, mely konfigurálásáról korábban írtam. A tanításhoz szükséges konfigurációs fájlokat a mellékletben csatoltam ***yolov3_ts_test.cfg***, illetve ***yolov3_ts_train.cfg*** néven.

A GTSDB tanító adatbázishoz [58] a ***FullIJCNN2013.zip*** állományt kell letölteni és behelyezni a ***TSD_projektmunka*** projektbe. A projektkönvytáron belül van egy

GTSDB elnevezésű almappa, ide kell bemásolni a ***FullIJCNN2013.zip*** állományon belül lévő ***FullIJCNN2013*** mappa tartalmát. Ezt követően le kell futtatni a ***prepare_tsd_dataset.py*** fájlt az adatbázis előfeldolgozásához. Ebben a fájlban található egy változó ***(full_path)***, ami a projekt teljes elérési útvonalát jelenti. Ezt mindenképpen meg kell változtatni a felhasználó egyéni útvonalára. Ennek eredményeképpen létrejönnek a tanításhoz szükséges fájlok. Ezek ***test.txt***, ***train.txt***, ***classes.names***, illetve ***ts_data.data***. Ezek közül a ***classes.names***, illetve a ***ts_data.data*** fájlt át kell átmásolni a ***darknet-master/build/darknet/x64/cfg*** mappába. Illetve ide kell a ***yolov3_ts_test.cfg***-t és a ***yolov3_ts_train.cfg***-t is bemásolni. Miután ezek megvannak ***darknet-master/build/darknet/x64*** mappában létre kell hozni egy ***'weights'*** mappát is, ide fog bekerülni egy előre betanított modell, a ***darknet53.conv.74*** fájl. Ezt a fájlt szintén a darknet hivatalos weboldaláról lehet elérni. Ezután minden készen áll a tanítás megkezdéséhez, amit a ***'darknet.exe detector train cfg\ts_data.data cfg\yolov3_ts_train.cfg weights\darknet53.conv.74'*** paranccsal lehet elindítani.

8. ÖSSZEFOGLALÁS

A szakdolgozatom célja egy metodológia létrehozása volt sávok detektálására, illetve közlekedési táblák felismerésére.

A sávdetektáló modulnál különböző képfeldogozói algoritmusok voltak használva. Először meg kellett határozni a ROI-t a bemeneti képen, hogy csak a lényeges területre fókuszáljunk. Ezt követően binarizálni kellett, melyhez HLS, illetve HSV küszöbölést alkalmaztam. Ezután perspektív transzformációval olyan képet hoztam létre, melyen könyebben lehet detektálni a sáv pixeleket, amit pedig a csúszó ablakos kereséssel oldottam meg. Ebben az algoritmusban egydimenziós Kalman szűrő segítségével pozicionáltam az ablakokat, hogy a zajokat a lehető legjobban ki tudjam szűrni. A metodológia végén két polinóm készült a sáv pixelek felhasználásával, amelyeket inverz perspektív transzformációval az eredeti képre rajzoltam. A pontosság növelése érdekében a sávokat kirajzolás előtt validálom, illetve a validált sáv paramétereit elmentem későbbi felhasználásra, amennyiben bizonyos feltételek teljesülnek.

A tábladetektáláshoz YOLOv3 CNN modellt használtam, amit a GTSDB adatbázissal tanítottam be a darknet segítségével. A táblafelismeréshez három különöző CNN modellt próbáltam ki, melyet két különböző adatbázissal tanítottam be. A tanítás eredményei alapján meghatároztam melyik tanító adatbázis, illetve melyik modell bizonyult a legjobbnak.

A képek augmentációját három különböző algoritmus segítségével oldottam meg. Első a perspektív transzformáció volt, a második a képeknek a forgatása, a harmadik pedig a képek fényességének változtatása.

A metodológiákat sikerült kiértékeljem, illetve összehasonlítanom más hasonló megoldásokhoz, továbbá az előző fejezetben pedig a továbbfejelsztési lehetőségeken mentem végig.

9. SUMMARY

The aim of this project was to develop a method for detecting lane lines and to detecting traffic signs as well.

For the lane detection, various image processing techniques were used. Firstly, the ROI needed to be extracted from the image, then binarized using HLS and HSV thresholding. After that the image transformed using perspective transformations, to get a bird's eye view of the road, so the lane lines appear to be parallel, which helps the sliding window search algorithm to detect lane pixels. To filter out noisy pixels from the image, a one-dimensional Kalman filter was implemented. Also, some validation and fallback techniques were implemented to improve accuracy. In the end two polynomial were created of the pixels and warped back to the original image.

For the traffic sign detection, a YOLOv3 CNN model was used, which was trained with the GTSDB database. For the traffic sign classification 3 different models were trained with 2 different databases. The first one was the original GTSRB database, and the second one was the equalized database. These models were evaluated, and the best one was selected according to test accuracy.

For the image augmentation 3 different techniques were used to generate images. The first one was the perspective transformation, the second one was image rotation, and the last one was brightness changing.

The methodologies were evaluated, and compared to similar solutions, and further possible improvements were introduced.

10.IRODALOMJEGYZÉK

[1] https://www.nhtsa.gov/technology-innovation/automated-vehicles, utoljára megtekintve: 2021.04.11

[2] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans and L. V. Gool, "Towards End-to-End Lane Detection: an Instance Segmentation Approach," 2018 IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 286-291, doi: 10.1109/IVS.2018.8500547.

[3] Q. Lin, Y. Han and H. Hahn, "Real-Time Lane Departure Detection Based on Extended Edge-Linking Algorithm," 2010 Second International Conference on Computer Research and Development, 2010, pp. 725-730, doi: 10.1109/ICCRD.2010.166.

[4] Y. Yang, H. Luo, H. Xu and F. Wu, "Towards Real-Time Traffic Sign Detection and Classification," in IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 7, pp. 2022-2031, July 2016, doi: 10.1109/TITS.2015.2482461.

[5] T. S. Yu-Ichi Ohta and T. Kanade, "Color information for region segmentation", Comput. Graph. Image Process., vol. 13, pp. 222-241, 1980.

[6] J. Matas, O. Chum, M. Urban and T. Pajdla, "Robust wide-baseline stereo from maximally stable extremal regions", Image Vis. Comput., vol. 22, no. 10, pp. 761-767, 2004.

[7] Suthaharan, S. (2016). Support Vector Machine. In: Machine Learning Models and Algorithms for Big Data Classification. Integrated Series in Information Systems, vol 36. Springer, Boston, MA. https://doi.org/10.1007/978-1-4899-7641-3_9

[8] Ayoub Ellahyani, Mohamed El Ansari, Ilyas El Jaafari, Traffic sign detection and recognition based on random forests, Applied Soft Computing,Volume 46,2016,Pages 805-815, ISSN 1568-4946, Available: https://doi.org/10.1016/j.asoc.2015.12.041.

[9] Ming-Kuei Hu, "Visual pattern recognition by moment invariants," in *IRE Transactions on Information Theory*, vol. 8, no. 2, pp. 179-187, February 1962, doi: 10.1109/TIT.1962.1057692.

[10] Breiman, L. Random Forests. Machine Learning **45**, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[11] Mohan, Vijayarani. (2013). Performance Analysis of Canny and Sobel Edge Detection Algorithms in Image Mining. International Journal of Innovative Research in Computer and Communication Engineering. 1760-1767.

[12] Lijun Ding, Ardeshir Goshtasby,On the Canny edge detector,Pattern Recognition,Volume 34, Issue 3,2001,Pages 721-725,ISSN 0031-3203, Available: https://www.sciencedirect.com/science/article/pii/S0031320300000236

[13] Hoang Nhat-Duc, Quoc-Lam Nguyen, Van-Duc Tran, Automatic recognition of asphalt pavement cracks using metaheuristic optimized edge detection algorithms and convolution neural network, Automation in Construction, Volume 94,2018, Pages 203-213, ISSN 0926-5805, Available: https://www.sciencedirect.com/science/article/pii/S0926580518300499

[14] Mamta Juneja, Parvinder Singh Sandhu, Performance Evaluation of Edge Detection Techniques for Images in Spatial Domain, International Journal of Computer Theory and Engineering, Vol. 1, No. 5, 2009 1793-8201

[15] Nazma Nausheen, Ayan Seal, Pritee Khanna, Santanu Halder, A FPGA based implementation of Sobel edge detection, Microprocessors and Microsystems, Volume 56,2018, Pages 84-91, ISSN 0141-9331, Available: https://www.sciencedirect.com/science/article/pii/S0141933116302289

[16] G. Ravivarma, K. Gavaskar, D. Malathi, K.G. Asha, B. Ashok, S. Aarthi, Implementation of Sobel operator based image edge detection on FPGA, Materials Today: Proceedings,2020, ISSN 2214-7853, Available: https://www.sciencedirect.com/science/article/pii/S2214785320384625

[17] Şaban Öztürk, Bayram Akdemir, Comparison of Edge Detection Algorithms for Texture Analysis on Glass Production, Procedia - Social and Behavioral Sciences, Volume 195,2015, Pages 2675-2682, ISSN 1877-0428, Available: https://www.sciencedirect.com/science/article/pii/S1877042815039567

[18] Dagao Duan, Meng Xie, Qian Mo, Zhongming Han and Yueliang Wan, "An improved Hough transform for line detection," 2010 International Conference on Computer Application and System Modeling (ICCASM 2010), 2010, pp. V2-354-V2-357, doi: 10.1109/ICCASM.2010.5620827.

[19] Filippus S. Roux,Encyclopedia of Modern Optics (Second Edition),Coordinate Transformations and the Hough Transform,Elsevier,2018, ISBN 9780128149829, Available: https://www.sciencedirect.com/science/article/pii/B9780128035818093772

[20] Ismaïl El Hajjouji, Salah Mars, Zakariae Asrih, Aimad El Mourabit, A novel FPGA implementation of Hough Transform for straight lane detection, Engineering Science and Technology, an International Journal, Volume 23, Issue 2, 2020, ISSN 2215-0986, Available: https://www.sciencedirect.com/science/article/pii/S2215098618314782

[21] Raja Muthalagu, Anudeepsekhar Bolimera, V. Kalaichelvi, Lane detection technique based on perspective transformation and histogram analysis for self-driving cars, Computers & Electrical Engineering, Volume 85,2020,106653,ISSN 0045-7906,Available: https://www.sciencedirect.com/science/article/pii/S0045790620305085

[22] K. Dinakaran, A. Stephen Sagayaraj, S.K. Kabilesh, T. Mani, A. Anandkumar, Gokul Chandrasekaran, Advanced lane detection technique for structural highway based on computer vision algorithm, Materials Today: Proceedings,2020, ISSN 2214-7853, Available: https://www.sciencedirect.com/science/article/pii/S2214785320373302

[23] Raja Muthalagu, Anudeepsekhar Bolimera, V. Kalaichelvi, nLane detection technique based on perspective transformation and histogram analysis for self-driving cars, Computers & Electrical Engineering, Volume 85,2020,106653, ISSN 0045-7906, Available: https://www.sciencedirect.com/science/article/pii/S0045790620305085

[24] https://www.kalmanfilter.net/alphabeta.html,utoljára megtekintve, utoljára megtekintve: 2022.04.18

[25] N. Buduma and N. Locascio, Fundamentals of deep learning: Designing nextgeneration machine intelligence algorithms. " O'Reilly Media, Inc.", 2017.

[26] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, Roberto Prevete, A survey on modern trainable activation functions, Neural Networks,Volume 138,2021, Pages 14-32,ISSN 0893-6080, Available: https://www.sciencedirect.com/science/article/pii/S0893608021000344

[27] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning, ser. Adaptive computation and machine learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[28] Jonas Teuwen, Nikita Moriakov, Chapter 20 - Convolutional neural networks, Editor(s): S. Kevin Zhou, Daniel Rueckert, Gabor Fichtinger, In The Elsevier and MICCAI Society Book Series, Handbook of Medical Image Computing and Computer Assisted Intervention, Academic Press, 2020, Pages 481-501, ISBN 9780128161760, Available: https://www.sciencedirect.com/science/article/pii/B9780128161760000259

[29] K. Kranthi Kumar, M. Dileep Kumar, Ch. Samsonu, K. Vamshi Krishna,Role of convolutional neural networks for any real time image classification, recognition and analysis, Materials Today: Proceedings,2021,ISSN 2214-7853,Available: https://www.sciencedirect.com/science/article/pii/S2214785321012682

[30] Ho, M.L., Chan, P.T. and Rad, A.B. (2009), Lane change algorithm for autonomous vehicles via virtual curvature method. J. Adv. Transp., 43: 47-70. Available: https://doi.org/10.1002/atr.5670430104

[31] Xing, Shiyu and Jakiela, Mark, "Lane Change Strategy for Autonomous Vehicle" (2018). Mechanical Engineering and Materials Science Independent Study. 61. *https://openscholarship.wustl.edu/mems500/61*

[32] Hai Wang, Xinyu Lou, Yingfeng Cai, Yicheng Li, Long Chen, "Real-Time Vehicle Detection Algorithm Based on Vision and Lidar Point Cloud Fusion", Journal of

Sensors, vol. 2019, Article ID 8473980, 9 pages, 2019. Available: https://doi.org/10.1155/2019/8473980

[33] Da Yang, Shiyu Zheng, Cheng Wen, Peter J. Jin, Bin Ran, A dynamic lane-changing trajectory planning model for automated vehicles, Transportation Research Part C: Emerging Technologies, Volume 95, 2018, Pages 228-247,ISSN 0968-090X, Available: https://www.sciencedirect.com/science/article/pii/S0968090X18305898

[34] SHAIKH, Er Atique; DHALE, Atul. AGV path planning and obstacle avoidance using Dijkstra's algorithm. International Journal of Application or Innovation in Engineering & Management (IJAIEM), 2013, 2.6: 77-83.

[35] Qiu-Yan Pei, Li-Juan Hao, Chun-Hua Chen, Xiao-Lei Zheng, Tao He, Minimum collective dose based optimal evacuation path-planning method under nuclear accidents, Annals of Nuclear Energy, Volume 147, 2020, 107644, ISSN 0306-4549, Available: https://www.sciencedirect.com/science/article/pii/S030645492030342X

[36] Stephen Balakirsky, Chris Scrapper, Knowledge representation and planning for on-road driving, Robotics and Autonomous Systems, Volume 49, Issues 1–2, 2004, Pages 57-66, ISSN 0921-8890, Available: https://www.sciencedirect.com/science/article/pii/S0921889004001368)

[37] A.R. Soltani, H. Tawfik, J.Y. Goulermas, T. Fernando, Path planning in construction sites: performance evaluation of the Dijkstra, A∗, and GA search algorithms, Advanced Engineering Informatics,Volume 16, Issue 4,2002, Pages 291-303,ISSN 1474-0346, Available: https://www.sciencedirect.com/science/article/pii/S1474034603000181

[38] Adem Tuncer, Mehmet Yildirim, Dynamic path planning of mobile robots with improved genetic algorithm, Computers & Electrical Engineering,Volume 38, Issue 6, 2012, Pages 1564-1572,ISSN 0045-7906, Available: https://www.sciencedirect.com/science/article/pii/S0045790612001255

[39] M.A. Porta Garcia, Oscar Montiel, Oscar Castillo, Roberto Sepúlveda, Patricia Melin, Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation, Applied Soft Computing, Volume 9, Issue 3, 2009, Pages 1102-1110, ISSN 1568-4946, Available: https://www.sciencedirect.com/science/article/pii/S1568494609000349

[40] Imen Châari, Anis Koubâa, Hachemi Bennaceur, Adel Ammar, Sahar Trigui, Mohamed Tounsi, Elhadi Shakshuki, Habib Youssef, On the Adequacy of Tabu Search for Global Robot Path Planning Problem in Grid Environments, Procedia Computer Science, Volume 32, 2014, Pages 604-613, ISSN 1877-0509, Available: https://www.sciencedirect.com/science/article/pii/S1877050914006668

[41] P. Ganesan, V. Rajini, B. S. Sathish and K. B. Shaik, "HSV color space based segmentation of region of interest in satellite images," *2014 International Conference*

on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), 2014, pp. 101-105, doi: 10.1109/ICCICCT.2014.6992938.

[42] Filterpy documentation 1.4.4, Available: https://filterpy.readthedocs.io/_/downloads/en/stable/pdf/ ,utoljára megtekintve: 2022.04.02

[43] https://github.com/peter-moran/highway-lane-tracker, utoljára megtekintve: 2022.04.02

[44] https://net.jogtar.hu/jogszabaly?docid=a0100011.kov, utoljára megtekintve: 2022.04.02

[45] https://math24.net/curvature-radius.html, utoljára megtekintve: 2022.04.02

[46]https://towardsdatascience.com/deeppicar-part-4-lane-following-via-opencv-7[37dd9e47c96, utoljára megtekintve: 2022.05.10

[47] Sichkar V.N., Kolyubin S.A. Real time detection and classification of traffic signs based on YOLO version 3 algorithm. Scientific and Technical Journal of Information Technologies, Mechanics and Optics, 2020, vol. 20, no. 3, pp. 418–424 (in English). doi: 10.17586/2226-1494-2020-20-3-418-424

[48] Li, Tao & Ma, Yitao & Endoh, Tetsuo. (2020). A Systematic Study of Tiny YOLO3 Inference: Toward Compact Brainware Processor with Less Memory and Logic Gate. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3013934.

[49] https://pjreddie.com/darknet/, utoljára megtekintve: 2022.04.18

[50] https://www.udemy.com/course/training-yolo-v3-for-objects-detection-with-custom-data/, utoljára megtekintve: 2022.04.18

[51] V. N. Sichkar and A. V. Lyamin, "Design of Deep CNN Model for Effective Traffic Signs Recognition," 2021 International Russian Automation Conference (RusAutoCon), 2021, pp. 367-373, doi: 10.1109/RusAutoCon52004.2021.9537445.

[52] https://www.udemy.com/course/convolutional-neural-networks-for-image-classification/, utoljára megtekintve: 2022.04.18

[53] https://www.kaggle.com/code/valentynsichkar/traffic-signs-detection-by-yolo-v3-opencv-keras/notebook, utoljára megtekintve: 2022.04.18

[54] J. Hur, S. -N. Kang and S. -W. Seo, "Multi-lane detection in urban driving environments using conditional random fields," 2013 IEEE Intelligent Vehicles Symposium (IV), 2013, pp. 1297-1302, doi: 10.1109/IVS.2013.6629645.

[55] Betke, M., Haritaoglu, E. & Davis, L. Real-time multiple vehicle detection and tracking from a moving vehicle. Machine Vision and Applications 12, 69–83 (2000). https://doi.org/10.1007/s001380050126

[56] Masako Omachi and Shinichiro Omachi, "Traffic light detection with color and edge information," *2009 2nd IEEE International Conference on Computer Science and Information Technology*, 2009, pp. 284-287, doi: 10.1109/ICCSIT.2009.5234518.

[57] https://www.tensorflow.org/install/pip, utoljára megtekintve: 2022.04.18

[58] https://benchmark.ini.rub.de/, utoljára megtekintve 2022.04.18