GÉPI LÁTÁS

TKNB_INTM038

KÖZLEKEDÉSI TÁBLA FELISMERŐ ALKALMAZÁS

CSIZI SZEBASZTIÁN MÁRK

VUADF2

2019/20/2

Tartalom

1 Bevezetés

1.1 Feladatleírás

A féléves feladatom egy közlekedési tábla felismerő alkalmazás elkészítése volt. A program képes detektálni és felismerni 43 különböző KRESZ táblát. A megoldásomban több kép feldolgozási eljárást használok, mint a küszöbölés, detektálása, CNN. A dokumentumban részletezni fogom megoldásom elméleti hátterét, programom felépítését, és használatát.

1.2 Közlekedési táblák

A KRESZ táblák olyan objektumok, amelyek jól megfogalmazható szabályokkal rendelkeznek. A táblák esetében ilyenek a színek és a formák. Ezek a tulajdonságaik az esetek többségében jól kiemelik őket a környezetükből. Az alakzatok a: kör, oktagon, rombusz, háromszög, négyzet. A színek pedig a: vörös, kék és sárga. Ezek figyelembevételével a detektálás megadhatja a számunkra fontos részeit a képnek (ROI: Region Of Interest).

1.3 Potenciális kihívások

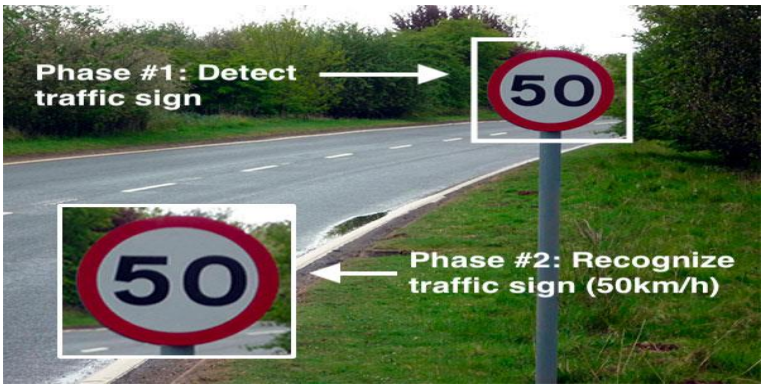
A táblák felismerését, rengeteg tényező befolyásolhatja:

- Táblák megrongálódása.
- Táblákhoz hasonló színek, vagy formák. (épületek, plakátok)
- Környezeti hatások.
- Különböző fényviszonyok.

1.ábra: kihívások

1.4 Objektumok felismerése

Objektumok felismerésének három fő lépése van. A kép előfeldolgozása, detektálás és a felismerés. Sok esetben az első kettőt egy lépésként értelmezik.



2.ábra: detektálás és felismerés

1.5 Előfeldolgozás

A képek eltérő minősége miatt érdemes előfeldolgozni őket. Ilyenek a zajszűrés vagy a méretezés. A méretezés során egyes pixelek értéke nem egyértelmű, ezért érdemes egy olyan interpolációs függvény használata, ami a kép minőségét kevésbé rontja. Ez a függvény egy feltételezés arra vonatkozóan, hogy bizonytalan pixeleknek milyen színe lehetett. A zajszűrésnek sok fajtája ismert, ahol a kis kiterjedésű impulzus szerű részeit a képnek próbáljuk csökkenteni.

1.6 Detektálás

Detektálás célja, hogy a potenciális objektumokat szétválasszuk azok környezetétől. Esetünkbe ez két fő lépésből áll.

1.6.1 Színek intervallumokra osztása

1.6.1.1 Küszöbölés

Küszöbölés alatt azt a folyamatot értjük amikor intenzitástartományt intervallumokra osztjuk, és az egyes intervallumokhoz 1-es vagy egy 0 értéket rendelünk, végeredményül pedig egy bináris képet kapunk.

1.6.1.2 HSV

Sok TSR rendszer HSV (H(Hue)-színárnyalat, S(Saturation)-színtelítettség, V(Value)-világosság) színtérrel végzi el a küszöbölést. Ezt a színteret kifejezetten a képfeldolgozáshoz készítették el. Az RGB és más színterekhez képest sokkal gyorsabb a detektálási sebessége, ami a kisseb számítási komplexitásnak köszönhető. Továbbá a fényviszonyok kevésbé vannak hatással rá. [2][3]

1.6.2 2. Kép formákra osztása

A színekkel való detektálásra a fényviszonyok hatással vannak. Ebben az esetben érdemes a képet formákra osztani és külön-külön elemezni őket, hogy megfelelnek-e a keresendő

objektum paramétereinek. A feladatom megoldásában a már színekkel detektált formák transzformálása, elemzése zajlik.

1.6.2.1 Kontúrok detektálása

A körvonalakat detektálása a bináris képeken, lehetővé teszik a formák és méretük megismerését.

1.6.2.2 Morfológia

Kép alakjának és struktúrájának olyan elemzése, ahol képet összehasonlítjuk egy strukturáló elemmel. A strukturáló elem ütköztetésével az objektumokat a kívánt alakra hozzuk. Két nagyobb fajtája van erózió és dilatáció.

1.6.2.3 Gépi tanulás

Rengeteg gépi tanulási módszer létezik: mint a Tartóvektor-gép (SVM: Support Vector Machine) és a konvolúciós neurális hálózat (CNN: Convolution Neural Network). Általában ezeket használják táblák detektáláshoz és vagy a felismeréshez. [3]

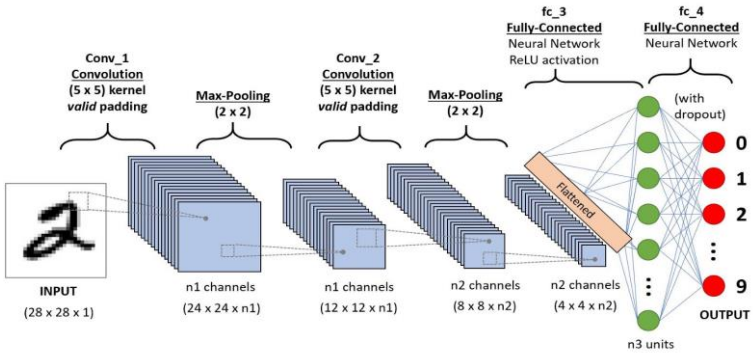
1.6.2.4 CNN

A CNN egy matematikai modell, amivel szimulálni akarjuk a biológiai neurális hálózattokat. A CNN tervezésekor az agy vizuális kérgének utánozása volt a cél. Ennek a kéregnek a sejtjei azok, amelyek felelősek a fény érzékeléséért. Egy neurális hálózatok bementére érkező adatokat manipulálja „súlyok" beállításával, amelyek feltevések arra, hogy bemeneti jellemzők hogyan kapcsolódnak egymáshoz és az objektumok osztályaihoz. [1]

A hálózat kiképzésekor a súlyok értékeit beállítják. A súlyokat a konvolúció nevezetű matematikai művelet hozza létre, ezek a súlyok lénygében a kép egyes részeinek ábrázolása. A kernel vagy szűrő ezen súlyok összesége. A létrehozott szűrő kisebb, mint a teljes bemeneti kép, csak a kép egy részét fedi le. A szűrőben szereplő értékeket megszorozzuk a képen szereplő értékekkel. A szűrőt ezután áthelyezik a kép új részének ábrázolására, és a folyamatot addig ismétlik, amíg a teljes képet le nem fedi. Az egész bemeneti kép körül mozgatott szűrők kimenete egy kétdimenziós tömb, amely a teljes képet ábrázolja. Ezt a tömböt „feature map" - nek hívják.[1]

A CNN folyamatosan fejlődik az adatok növekedésével, mivel képesek alkalmazkodni a megtanult tulajdonságokhoz és saját szűrőket fejleszteni. A "hideg indításkor" (cold start) ezeket a szűrőket kézzel kel beállítani.[6]

Tehát ez az algoritmus képes egy bemeneti kép különböző tulajdonságainak megtanulására, de sok más területen is alkalmazzák. Ez lehetővé teszi a fontosabb objektumok megjegyzését és azok felismerését. Egyik előnye a többi gépi tanulás hoz képest az, hogy az adatokat önmagukban előfeldolgozza. Így csökkentve az erőforrás igényeket. [6]



3.ábra: tipikus CNN felépítés

Az egyszerű CNN egy rétegkészletből áll. A bejövő adatok keresztül haladnak differenciálható függvényeken a rétegekben, amíg el nem érjük a kimeneti értékeket. A normál CNN háromféle réteggel rendelkezik a hálózat felépítéséhez: Convolutional Layer, Pooling layer, és Fully-Connected Layer.

Convolutional Layer: Meghatározott számú konvolúciós szűrőt alkalmaznak a képekre. Matematikai műveletekkel a szűrők egyetlen értéket állítanak elő a kimeneti feature map-en.

Pooling layer: Ezek a szűrők alakítják át Convolutional Layer által kivont képadatokat. Csökkentik a features map dimenzióját feldolgozás redukálásának érdekében.

Fully-Connected Layer: Osztályozzák a mintákat a korábbi rétegek munkája alapján. Itt minden réteg csomópontja kapcsolódik az előző réteg csomópontjához.[3]

2 Megoldásom

2.1 Program felépítése

A közlekedési táblák felismeréséhez két kódot használtam fel. Az első egy olyan python kód (train.py) amely megtanítja egy kész adatbázis elemeinek felismerésére (GTSRB). A kódot elemeztem, de mivel tesztelni és lefuttatni nem tudtam ezért a végeredményül kapott h5 súly fájlt használtam a felismeréshez. A második python fájl (TSR.py) a kapott képeken detektálja, majd a korábban edzett CNN-el felismerni azokat.

CNN modell elkészítése (train.py):

- Adatbázis betöltése és előfeldolgozása
- CNN modell felépítése
- Edzés (train)
- Modell tesztelése a teszt adatkészlettel

Detektálás és felismerés (TSR.py)

Detektálás:

- Kép betöltése.
- A képet RGB színtérről HSV színtérre alakítása.

- Küszöbölés.
- Morfológiai nyitás.
- Kontúrok detektálása.
- Méret és formák értelmezése. Ez alapján a potenciális részek kiválasztása.

Felismerés:

- Osztályozás a korábban készített CNN-nel.

Eredmények rajzolása a képre és megjelenítése.

2.2 Adatbázis

A German Traffic Sign Recognition Benchmark egy olyan adatbázis, ami 43 különböző német közlekedési tábla képét tartalmazza különböző db számban. Tartalmazza továbbá a címkéket és a teszteléshez használt képeket.

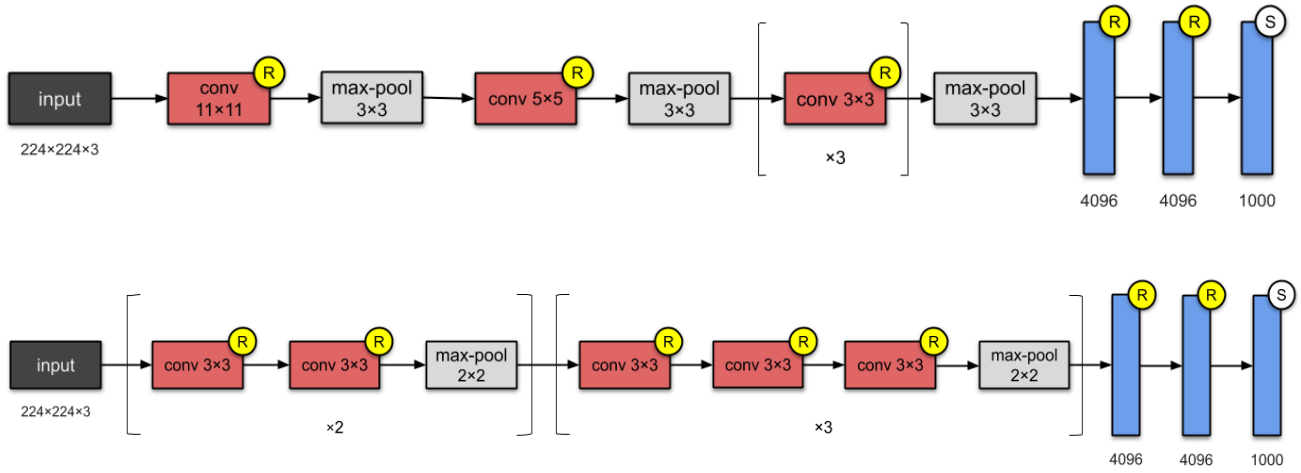
2.3 CNN

2.3.1 Adatbázis betöltése és előfeldolgozása

A 43 osztály betöltése OS segítségével. Átméretezés 30*30-as méretre. A PIL könyvtárral egy tömbbé alakítása a képeknek. Címkék eltárolása. Utána sklearn csomagot train_test_split() metódusával választja szét a train és a test adatokat. A keras.utils csomag to_categorical függvényével az y_test-ben és a t_train-ben lévő címkék átalakítása egy ”forró” kóddá.

2.3.2 CNN modell felépítése

A modell felépítése hasonlít az AlexNet modellre, amit Alex Krizhevsky tervezett és publikált Ilya Sutskever-el és Geoffrey Hinton-al 2012-ben Ez az architektúra 8 réteget használ -5 kovolúciós és 3 teljesen csatlakoztatok. Továbbá merít a 2014-ben publikált Karen Simonyan, és Andrew Zisserman által publikált VGG-16-ból.



4.ábra: fent AlexNet, lent VGG-16

A használt modell felépítése pedig a következő:

- Convolution (konvolúciós) réteg 5x5 kernel, és 32 szűrő. (ReLU)
- Convolution (konvolúciós) réteg 5x5 kernel, és 32 szűrő. (ReLU)
- Max pool réteg 2x2 kernel
- Dropout
- Convolution réteg 5x5 kernel, 64 szűrő. (ReLU)
- Convolution réteg 5x5 kernel, 64 szűrő. (ReLU)
- Max pool réteg 2x2 kernel
- Dropout
- Flatten
- Dense (ReLU)
- Dropout
- Dense (softmax)

Az **INPUT** réteg a bemeneti képet kétdimenziós pixelérték-tömbként tartja. Első conv réteg.
A **CONV** réteg kiszámítja a pontértékeket a kernel és a kapott kép altömbje között, ami pontosan akkora, mint a kernel. Ezután összeadja a ponttermékből származó összes értéket, és ez lesz a kimeneti kép egyetlen pixelértéke. Ezt a folyamatot addig ismételjük, amíg a teljes bemeneti kép nem lesz lefedve.
A **ReLU** réteg egy aktiválási függvényt alkalmaz (max. 0, x) a kimeneti kép összes pixelértékére.
A **POOL** réteg mintavételt vesz egy kép szélessége és magassága mentén, ami csökkenti a kép méretét.
Az **FC** (Fully-Connected) réteg kiszámítja az osztályozási pontszámot az összes osztályozási kategóriához.

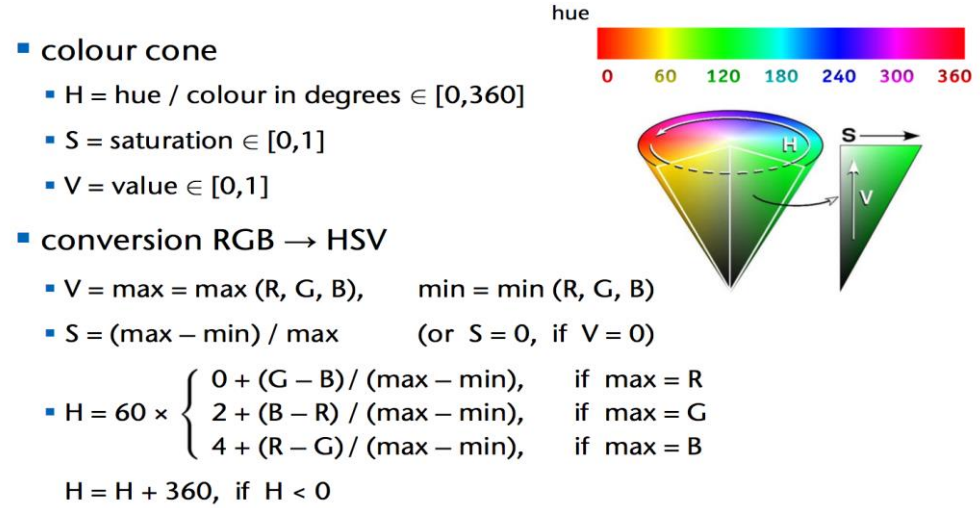
2.3.3 Edzés (train) és tesztelés

A model.fit() függvénnyel edzi meg a modellt, aminek az eredménye 95%-os hatékonyság.
A végén 30 × 30 pixelre méretezi a képeket, és egy numpy tömbbé alakítja azokat. Ez tartalmazza a kép összes adatát.

2.4 Detektálás

2.4.1 A képet RGB/BGR színtérről HSV színtérre alakítása

Ahogyan korábban szó volt róla, érdemes a színteret HSV-re alakítani.



5.ábra: HSV értelmezése

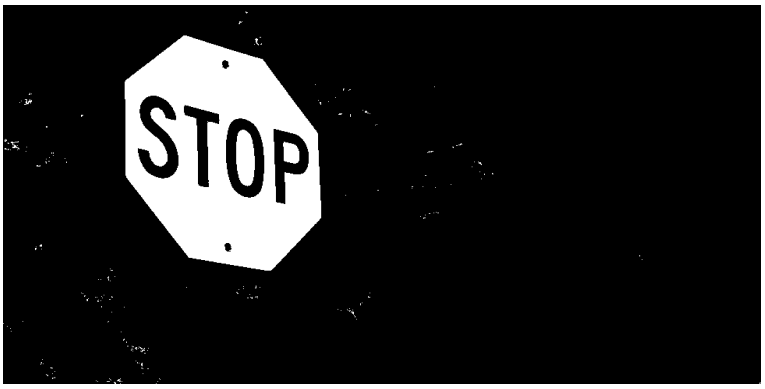
6.ábra: RGB és HSV

2.4.2 Küszöbölés

A küszöböléshez a H és S értékeket használják, mivel a V nem hordoz információt a színről. Ahhoz, hogy a meghatározott színek legyenek láthatóak a képen szükség van a megfelelő küszöbérték meghatározására. Ezeket a táblázatban láthatjuk **1.táblázatban**. Azok az értéket, amik beleesnek ezekbe a tartományokba egyesek lesznek, amik nem azok pedig nullák. Érdemes a küszöbölés elött zajszűrést végezni, amit átlagoló szűréssel old meg a program.

1.táblázat: H és S küszöbértékek

	H	**S**
Vörös	0-12 és 160-210	132-255
Kék	100-140	100-255
Sárga	15-25	110-255

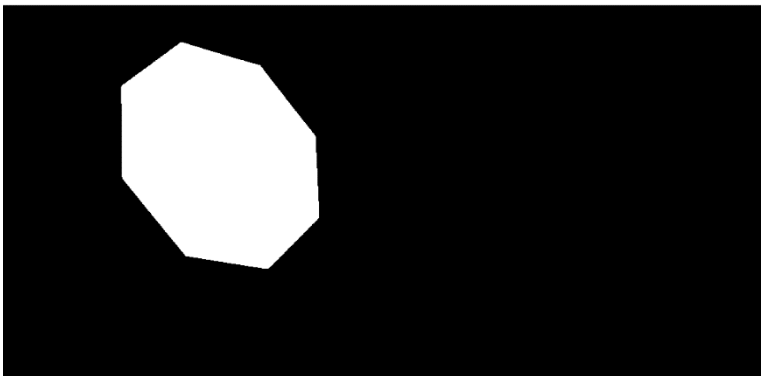


7.ábra**:** küszöbölés

2.4.3 Morfológia

A képen sok zaj is megjelenhet a küszöbölés során, amik megfelelnek a kívánt szín értékeknek, de program szempontjából feleslegesek.

1. Nyitás: Bináris zajok eltávolítása.
2. Zárás: Megmaradt foltok konvexitási problémáinak javítása.
3. Erózió: Objektumok határainak szétválasztása.



8.ábra: zajszűrés

2.4.4 Formák geometriai tulajdonságai

A közlekedési táblák jól felismerhető szabályos formák, ezért bizonyos tűrési határok keretein belül meg kell felelniük az ilyen formákra jellemző matematikai törvényszerűségeknek. Ilyenek a kompaktság, cirkularitás, rektangularitás, oldalarányok (kresz táblák esetén egyhez közelít). A **2.táblázatban** ezen geometria szabályok képletei és az általam megadott tűréshatárok láthatóak.

- T: Test területe.
- K: Test kerülete.
- T_min: A minimális területű befoglaló téglalap területe
- W: T_min szélessége.
- H: T_min magassága

2.táblázatban: Formák szűrése

Név	Képlet	Tűréshatárok
Kompaktság	$\frac{K^2}{T}$	13<=x<=25
Cirkularitás	$\frac{T}{K^2}$	0.06<=x<=0.07
Rektangularitás	$\frac{T}{T\_min}$	0.85<=x<=1.1
Oldalarány	$\frac{W}{H}$	0.70<=x<=1.1

2.4.5 Felismerés

Az eddigi szűréseken átment területei a képnek, levágásra kerülnek. Ezeket 30*30 pixelre méretezi és osztályozza a korábban készített CNN-el.

3 Tesztellés

3.1 Detektálás teszt:

A detektáló algoritmusomat 10 osztályra teszteltem. A **2.táblázatban** a GTSRB Train mappa: 0,5,11,12,13,14,15,18,33,34 osztályainak teszt eredményei láthatók. A **3.táblázatban** azzal javítottam az eredményeket, hogy 70*70-es képeknél kissebeket nem teszteltem.

3.táblázat: Detektálás eredményei

Tábla	**Összes tábla (db)**	**Helyes (db)**	**Helytelen (db)**	**Pontosság (%)**
Maximális sebesség 20 km/h	210	66	144	31.43
Maximális sebesség 80 km/h	1860	315	1545	16.94
Állj! Elsőbbségadás kötelező	780	246	534	31.54
Elsőbbségadás kötelező	2160	450	1710	20.83
Mindkét irányból behajtani tilos	630	155	475	24.60
Egyéb veszély	1200	262	938	21.83
Út kereszteződés (alá. úttal)	1320	346	974	26.21
Kötelező haladási irány (meg. bal)	420	82	338	19.52
Kötelező haladási irány (meg. jobb)	689	108	581	15.67
Főútvonal	2100	501	1599	23.86
Összes	11369	2531	8838	22.26

4.táblázat: Javított detektálás eredményei

Tábla	**Összes tábla (db)**	**Helyes (db)**	**Helytelen (db)**	**Pontosság (%)**
Maximális sebesség 20 km/h	46	24	22	52.17

Maximális sebesség 80 km/h	118	50	68	42.37
Állj! Elsőbbségadás kötelező	237	99	138	41.77
Elsőbbségadás kötelező	504	111	393	22.02
Mindkét irányból behajtani tilos	69	17	52	24.64
Egyéb veszély	340	109	231	32.06
Út kereszteződés (alá. úttal)	359	113	246	31.48
Kötelező haladási irány (meg. bal)	49	6	43	12.24
Kötelező haladási irány (meg. jobb)	123	20	103	16.26
Főútvonal	400	97	303	24.25
Összes	2245	646	1599	28.78

3.2 Osztályozás teszt:

A teljes 43 osztályra teszteltem a CNN modellt. Itt 10 nagyobb mértékben eltérő osztály eredményét és a teljes teszt eredményt tüntetem fel.

5.táblázat: Osztályzás eredményei

Tábla	Összes tábla (db)	Helyes (db)	Helytelen (db)	Pontosság (%)
Maximális sebesség 20 km/h	210	208	2	99.05
Elsőbbségadás kötelező	2160	2159	1	99.95
Állj! Elsőbbségadás kötelező	780	780	0	100
Főútvonal	2100	2094	6	99.71
Mindkét irányból behajtani tilos	630	627	3	99.52
Egyéb veszély	1200	1199	1	99.92
Körforgalom	360	359	1	99.72
Kötelező haladási irány egyenesen	1200	1199	1	99.92
Közlekedési lámpa	600	476	124	79.33
Előzni tilos!	1470	1468	2	99.86
…	…	…	…	…
Összes	39209	38887	322	99.18

3.2.1 Osztályozás és tesztelés:

ND: Nem detektált
DNF: Detektált, de nem felismert
DF: Detektált és felismert

9.táblázat: Detektálás és felismerés eredményei

Tábla	Összes	ND	DNF	DF
Stop	20	8	2	10
Főút	20	12	8	0
Elsőbbség köt.	20	15	0	5
Körforgalom	20	18	1	1
Vegyes	20	4	12	4
Összes	100	57	23	20

3.3 A rendszer határainak megállapítása:

Értelemszerűen a detektálás adja meg a rendszer pontosságát mivel, ha az nem megfelelő, akkor a felismerés semmit nem tud tenni. A rendszer az átméretezéseket nem kezeli jól ezért a kis képek nagyobbá tételével sem tudja detektálni a kisebb táblákat, ami részben jó mivel ezek egy valós képen a távolságot jelentenék. A nagyon távoli táblákkal pedig nem is érdeke a TSR rendszereknek detektálnia. A valós környezetben a rendszer nem működik nagy pontossággal.

4 Használati útmutató

1. A train.py cmd-ből való futtatása előtt szükséges, az adatbázis kicsomagolása és annak egy mappában történő elhelyezése.
2. A TSR.py és a cnn-ből elkészített traffic_classifier.h5 fájl egy mappában valós elhelyezése.
3. A cmd-ből futtassuk a TSR-t a következő paranccsal: TSR.py --image fájl útvonala. Egy lehetséges példa TSR.py --image /test/1.png.
4. A GTSRB_test.py külön teszteli a detektálást és a felismerést. Futtatás előtt szükséges a Train mappa és kód egy helyen történő elhelyezése.
5. A My_data_test.py a detektálást és a tesztet egyszerre végzi a My_data_inputs-ból. A kimenetet nem tudtam egyből My_data_outputs-ba menteni, de a jelenlegi kimeneteket ott tárolom.

Szükséges könyvtárak:

- Pillow 7.0.0
- tensorflow 2.0.0rc1
- sklearn: 0.0
- pandas: 1.0.3
- opencv-python: 4.2.0.34
- numpy: 1.18.2
- matplotlib: 3.2.1
- Keras: 2.3.1

5 Felhasznált irodalom

A felhasznált irodalom a dokumentumban számokkal megadott irodalom nevű mappában találhatók. Továbbá felhasználatam:

[4]https://www.pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/

[5]https://www.pyimagesearch.com/2018/09/10/keras-tutorial-how-to-get-started-with-keras-deep-learning-and-python/

[6]https://data-flair.training/blogs/python-project-traffic-signs-recognition/

[7]https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[8]https://regi.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0052_13_digitalis_kepelemzes_alapveto_algoritmusai/index.html

Órai jegyzeteket