Gépi látás

GKNB_INTM038

Jelzőtáblák felismerése

Török Kristóf

UWQK95

Győr, 2020/21/1

Tartalomjegyzék

1 Bevezetés 2
2 Elméleti háttér áttekintése 3
2.1 Jelzőtáblák felismeréséről általánosan 3
2.2 Különböző táblák, különböző országok 3
2.3 Gépi tanulás 3
2.4 Neurális hálózatok 3
2.5 Konvolúciós neurális hálózatok 4
3 A feladat megvalósítási terve kivitelezése 5
3.1 Ötletek, alaptervek 5
3.2 Adatbázis 5
3.3 Fejlesztői környezet, programnyelv, programcsomagok 5
3.4 Konvolúciós neurális hálózat programozása jelzőtáblák felismeréséhez 5
3.5 Grafikus felhasználói felület 6
4 A szoftver tesztelése 8
4.1 Teszttervezés 8
4.2 Tesztkészletek 8
4.3 Tesztek előkészítése, automatizálása, tesztelő algoritmus 8
4.4 Teszteredmények 9
4.5 Összegzés 10
5 Felhasználói leírás 11
5.1 A program telepítése 11
5.2 Felhasználói felület használata 11
5.3 Tesztprogram használata 11
5.4 Összegzés 11
6 Felhasznált irodalom 12

1 Bevezetés

Manapság az autóbalesetek nagy százaléka azért történik meg, mivel az autóvezetők valamilyen külső/belső behatás következtében figyelmen kívül hagyják az út menti jelzőtáblákat. Az ilyen mulasztások miatt bekövetkezett károk sokszor komoly anyagi terhet jelentenek, rosszabb esetben, mint ismeretes az emberi élet nem pótolható. Ezen esetek megelőzése érdekében kezdtek el az autógyártók különböző technológiákat kifejleszteni. Ezen technológiák közé tartozik a jelzőtábla felismerő rendszerek is, amelyek ma már a legtöbb autóban megtalálhatóak és fontos részét képezik a fejlett vezetéstámogató rendszereknek (Advanced Driver Assistance System – ADAS).

A beadandó dolgozat témája, amit már az előző bekezdés és a dolgozat címe is mutatja a jelzőtáblák felismerése. A dolgozat elkészítése során célom egy olyan program elkészítése, amely képes, különböző utcai képeken vagy különböző táblákról készült fotókon felismerni a képeken található táblákat.

A dolgozat elkészítése során először szükség van az elméleti háttér áttekintésére, amely a következő bekezdésben fog történni. Itt röviden szót ejtek a jelzőtáblák felismerésének történetéről és az első lépésekről, amelyek nagyban befolyásolták ennek a technológiának a kialakulását. Ebben a bekezdésben bővebben ki fogok térni a neurális hálózatokra, mivel a megoldás a mesterséges intelligencia alapjaira fog épülni. Miután az elméleti háttér áttekintésével végeztünk, áttérek a szoftvertervezés fázisaira, itt fontos leszögezni, hogy egy a mesterséges intelligencia által támogatott megoldást fogok megvalósítani, részletezni. A szoftvertervezés után kerül sor a konkrét szoftver megvalósítására és ennek a folyamatnak a részletezésére. Ezután következik a szoftver tesztelése, ahol a betanított neurális hálót fogom tesztelni egy nagyobb adathalmazon (megközelítőleg 100-150 db kép). Majd a feladat végén kerül sor a felhasználói leírás elkészítésre, amelynek segítségével az egyszerű felhasználók is képesek lesznek a program használatára.

2 Elméleti háttér áttekintése

2.1 Jelzőtáblák felismeréséről általánosan

A jelzőtáblafelismerés (TSR) olyan technológia, amellyel a jármű képes felismerni az út mentén elhelyezett közlekedési táblákat pl.: Sebesség korlátozó táblák. ADAS funkciók része. Ezt a technológiát különféle autóipari beszállítók fejlesztik, akik képfeldolgozási technikákat alkalmaznak a közlekedési táblák detektálására. A detektálási módszerek között szín-, alak-, és tanulásalapú módszereket különböztetünk meg. [1]

2.2 Különböző táblák, különböző országok

A közúti jelzőtáblákról szóló bécsi egyezmény segítségével 1968-ban sikerült egységesíteni a közlekedési táblákat a különböző országokban. Mintegy 52 ország írta alá a szerződést, amelyek között 31 európai ország szerepel. Ezen egyezmény nagy áttörést jelentett a világszerte használható közlekedési táblákat felismerő rendszerek fejlesztésében. [1]

2.3 Gépi tanulás

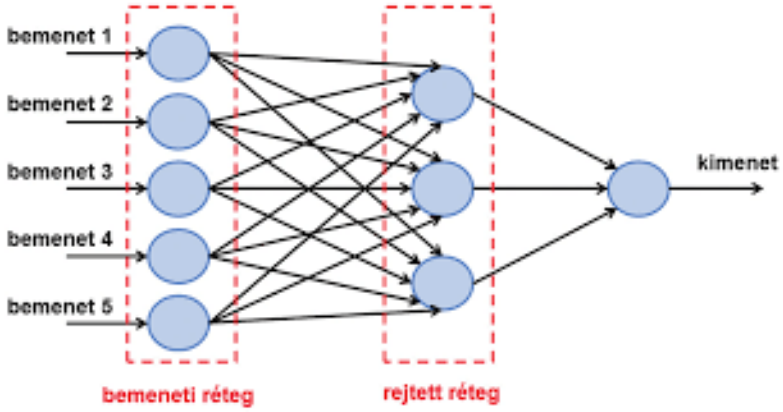
A gépi látás a modern informatikai megoldások közé sorolható, melynek célja az emberi gondolkodás és felismerés lemodellezése számítógépek segítségével. A gépi tanulás szorosan kapcsolódik a robotika tudományához is. Ez viszont természetesnek vehető, mivel emberi tulajdonságokkal próbálunk felruházni gépeket, és látható, hogy ők is képesek tévedésekre, mellélövésekre. A jövőben a gépi látás fő iránya az lehet, hogy olyan gépeket hozzunk létre, amelyek az embereket segítik és önálló gondolkodással rendelkeznek, tehát önálló döntéseket legyenek képesek meghozni. Ez nagy segítség lehet az emberek számára például a háztartásokban vagy más területeken, ezért a témán belül fontos kérdésnek számít az ember és gép együttműködése. [2]

Véleményem szerint a gépi látás a jövőben az informatika egyik meghatározó, ha nem a legmeghatározóbb ága lehet és remélhetőleg nem történek meg az, amit sok filmben már láthattunk, hogy a gépek az emberek ellen fordulnak. Az általam készített projektben például egy jó és hasznos megoldásra használom és úgy érzem, hogy világméretűleg is ennek kellene lennie a fő iránynak a jövőben.

2.4 Neurális hálózatok

A neurális hálózat a mesterséges intelligencia megoldásokhoz szorosan kapcsolódó fogalom. A gépi tanulás során elengedhetetlenné vált napjainkban. A mesterséges neurális hálózat az idegrendszer felépítése és működése analógiájára kialakított számítási mechanizmus. Hiszen a fő cél nem elvi, hanem ténylegesen működő modell létrehozása. Ezt pedig tipikusan valamilyen elektronikai eszközzel és valamilyen tudományos eljárással lehet elérni. Tehát a biológiai elvek alapján megalkottak bizonyos matematikai jellegű modelleket. Ezeket elméleti matematikai módszerekkel pontosították, alkalmazott matematikai módszerekkel számításokra alkalmassá tették, majd számítógépen realizálták. Azonban a matematikai módszerek mellett sokszor heurisztikus meggondolásokra és számítógépes kísérletezésre is szükség van. Egy neurális hálózatot érdemes úgy felfogni, hogy nem kívánja a jelenséget modellezni, arra törvényszerűségeket megállapítani, hanem a jelenséget fekete dobozként kezeli, csak a bemenő (input) és a kimenő (output) adatokat tekinti. A jó neurális hálózat olyan, hogy ugyanarra az inputra hasonló outputot ad, mint a vizsgált jelenség. De a fekete dobozban működő mechanizmust nem tárja fel, maga a neurális hálózat pedig nem hasonlít a jelenségre. A jelenségek fekete dobozként való kezelése a neurális hálózatoknak részben hátrányuk, de részben előnyük is. Hiszen működésükhöz csak adatokra van szükség. [3]

A neurális hálózatok lehet egyszerűek és bonyolultak is, amelyek számos rejtett réteggel rendelkezhetnek, ezek a rejtett rétegek nagyon fontosak lehetnek a gépi tanulásban.



1. ábra Példa neurális háló
Forrás: http://real.mtak.hu/88740/1/Tolcsvai-nyomda_156_laki.pdf
(Megtekintés napja: 2021. január 2.)

A neurális hálózat felépítéséről elmondható, hogy rendelkezik bemeneti adatokkal, amelyek a rejtett rétegen/rétegeken áthaladva fognak kimeneti eredményt adni. Minél több rejtett réteggel rendelkezik a neurális háló annál pontosabb eredményeket kapunk.

2.5 Konvolúciós neurális hálózatok

Ennek a bekezdésnek a forrása [4].

A klasszikus képfeldolgozásnál alapművelet a konvolúció. Alkalmazható zajszűrésre, alacsony képi jellemzők kiemelésére, összetett objektumok kiemelésére. A konvolúciós neurális hálózatok előnye a teljesen összekötött hálókhoz képest, hogy jóval kevesebb a szabad paraméter. A konvolúciós rétegben a bementi kép egy megadott nagyságú részét kivesszük és számítást végzünk rajt. Ebben a rétegben egy neuron érzékenységi mezőjének a bemeneti kép azon részét nevezzük, melytől függ a kimeneti értéke.

A pooling réteg lényegében mintavételezi a képet. Az avarage pooling a gyakori minták kiemelését segíti elő. A max pooling olyan jellemzők kiemelését segíti, melyek csak kis számban fordulnak elő. Az ilyen hálózatok rendelkeznek egy sorosító (angolul: flatten) réteggel is. Ennek a rétegnek a feladata a többdimenziós jelek egy dimenzióssá alakítása. Ez a réteg nem tartalmaz tanítható paramétereket. Ilyen neurális hálózat például a DenseNet(2017). A módszer lényege, hogy legyen rövid hiba-visszaterjesztési utak. További jellemzője, hogy egy blokkon belül minden réteg kimenete közvetlen bemenete minden azt követő rétegnek.

A konvolúciós neurális hálózati technológia egyik fő alkalmazási területe az objektumdetektálás. Ez a detektálás történhet csúszó ablakosan, régión alapúan. A csúszó ablakos megoldás különböző méretű téglalapokat tol végig a vizsgálandó képen, és az így kivágott részeket osztályozza. Az osztályokat a felismerendő objektum típusok és a háttér alkotja. Előnye, hogy kis munkaigényű, azonban hátránya, hogy vagy gyors és pontatlan, vagy lassú a pontosság növelése érdekében. A régión alapuló megoldás bemenete a kivágott és átméretezett képrészlet, míg a kimenete pedig a képrészlet osztályozása és a pozíciójának korrekciója. Gyakorlati alkalmazását tekintve egy rendkívül lassú megoldás, így létezik gyorsított változata is. A gyorsítás alapgondolata, hogy a teljes képet vizsgáljuk egy mély, teljesen összekapcsolt konvolúciós neurális hálóval, majd az abból kinyert jellemzőket vizsgáljuk egy sekélyebb neurális hálóval.

3 A feladat megvalósítási terve kivitelezése

3.1 Ötletek, alaptervek

A témaválasztást követően, miután már biztossá vált, hogy megkaptam az általam választott témát, elkezdtem különböző megoldások után kutatni. Az első keresések utána világossá vált, hogy biztosan szükség lesz egy adatbázisra és egy neurális hálózatra, amely az adatbázis segítségével tanítható és a tanítás befejeztével, a betanított neurális hálózat képes lesz felismerni különböző képeken a képeken található közlekedési táblákat. Ezenkívül az is biztossá vált, hogy szükség lesz tesztekre és tesztkészletekre, amelyekkel megállapítható a program felismerési hatékonysága.

3.2 Adatbázis

Eleinte komoly fejtörést okozott, hogy saját vagy már létező adatbázist használjak a program elkészítése során. Hosszas mérlegelés után úgy döntöttem, hogy egy már létező adatbázist fogok használni a neurális hálózat tanításához. Elsődleges szempont volt, hogy az adatbázis ingyenes legyen és olyan közlekedési táblákat tartalmazzon, amelyekkel Európában találkozhatunk, így akár saját magunk által készített képekkel is tesztelhető a program.

A keresgélés alatt sikerült egy olyan adatbázist találnom, ami megfelelt az általam előre meghatározott követelményeknek. Az adatbázis neve **German Traffic Sign Recognition Benchmark**(GTSRB) [5] amely több mint 50 ezer képet tartalmaz 43 db különböző közlekedési tábláról. Ez az adatbázis az alapmodell építésére kiválóan alkalmas volt.

3.3 Fejlesztői környezet, programnyelv, programcsomagok

A projekt elkészítése során fejlesztő környezetként a ***Visual Studio Code*** [6] alkalmazást használtam. Ennek az alkalmazásnak a használata rendkívül előnyös, mivel többféle programozási nyelvet támogat, emellett az általa elfoglalt hely is nagyon alacsony. További előnyként még megemlíthető, hogy platformfüggetlen. Ez a használat során nagyon nagy előnyt jelenthet a felhasználók számára.

A projekt ***Phyton*** nyelven íródott, elsősorban azért esett a választás erre a programozási nyelvre, mert a tárgy óráin is ezt a nyelvet használtuk és ezzel a programnyelvvel ismerkedhettem így meg a legjobban, emellett a Phyton a legkedveltebb nyelv a gépi tanulásos projektek készítőinek körében. Ezenkívül számomra az is előnyös volt, hogy a nyelv könnyen tanulható, szintaktikája és adatkezelése meglehetősen egyszerű.

A projekt elkészítése során többféle programcsomag használatára volt szükség. Először is szükség volt a ***tensorflow*** [7] programcsomag használatára, mivel ez elengedhetetlen a gépi tanulás és a neurális hálózatok végett. Ahhoz, hogy adatokat tudjunk kezelni beolvasni és feldolgozni szükség volt a ***numpy*** **[8]**, illetve ***pandas*** **[9]** programcsomagok használtára. Ezeken kívül ***matplotlib*** [12]programcsomag van még, ami azért van használva, hogy kiírassam a tanítás eredményét egy grafikon formában. A gépi tanulásos programrészeknél ezen kívül csupán két kisebb csomagösszetevőt használtam, melyek az ***sklearn [10]*** és a ***keras [11]*** programcsomagból származnak.

3.4 Konvolúciós neurális hálózat programozása jelzőtáblák felismeréséhez

Ezen bekezdés programkódját a **traffic_sign.py** fájl tartalmazza.

Azért választottam a konvolúciós megoldást a tanító algoritmus elkészítése során, mert ezzel jobb eredmények érhetők el a tanítás során az egyszerű neurális hálózathoz képest. Első lépésként betöltöm a tanító képeket majd numpy tömbökké alakítom azokat. Következő lépésként a bemeneti adatokat szétválogattam tanító és tesztelő bemenetekre. Ezt követően 43 kategóriába rendeztem őket.

Az általam választott hálózati modell egy szekvenciális, másnéven időfüggő neurális hálózat. Ezen modelltípus segítségével könnyen programozhatóak dinamikus rendszerek, ahol fontos az adatok időrendje is. Emellett az ilyen szekvenciális modellek alkalmasak jóslási problémák megoldására.

Modell első rétege egy kétdimenziós konvolúciós réteg. Ez a réteg 32 szűrőt tartalmaz melyek mérete 5x5-ös. A réteg aktivitása *relu,* vagyis javított lineáris akti-válási függvény. Ez az eljárás közvetlenül adja ki a bementet, ha pozitív, ellenkező esetben nullát ad vissza. Számos neurális hálózatban alapértelmezett aktiválási funkcióvá vált, mivel az ezt használó modellek könnyebben taníthatóak és pontosabbak. [13] Ezen réteg bemenet mérete megegyezik a tanító képek méretével. Következő rétegként egy az elsővel megegyező réteg került hozzáadásra. A harmadik réteg egy *MaxPooling2D* réteg, amelynek tulajdonsága, hogy a *pool_size* attribútum által meghatározott ablakon felüli maximális értéket veszi fel az egyes dimenziókhoz. [14] Az én modellemben ez az ablakméret 2x2-es. A következő három rétege a modellnek szinte teljes egészében megegyezik a modell első három rétegével. Csupán annyi a különbség, hogy a konvolúciós rétegek 64 szűrővel rendelkeznek és a szűrők mérete 3x3-as.

Az ezt követő réteg, egy *Flatten* réteg. Azért alkalmaztam ezt a réteget mivel a bemeneti képek egységes méretűek. Ez a réteg a bemeneteket egy dimenzióssá alakítja. A következő réteg egy 256 elemű *Dense* réteg, melynek aktivitása *relu.* Az ezt követő réteg egy *Dropout* réteg, mely véletlenszerűen nullára állítja a bemeneti egységeket. Ez segít a tanítás ideje alatt megelőzni a hálózat túlterhelését. [15] Az én hálózatomban a rate attribútum 0.5. A kimeneti réteg egy 43 elemű *Dense* réteg. Ezen réteg elemszáma megegyezik a tanítandó osztályok számával.

Miután a modell ezzel elkészült, következik a lefordítása. A fordítás során az *adam* optimalizálót használtam. Ez az optimalizáló algoritmus szolgál a hálózati súlyok iteratív frissítésére, a tréning adatok figyelembevételével. A veszteség számításához a kategorikus keresztcentrikusságot használtam. Ez az algoritmus arra szolgál, hogy kiszámítja a címkék és a jóslatok közötti keresztcentrikus veszteséget. Ezt a funkciót akkor célszerű használni, mikor kettő vagy több kimenet van, így ez jelen esetben kiválóan alkalmas a feladatra. [16] Metrikáknak a pontosságot használtam. Ez a módszer ki-számolja, hogy a jóslat milyen gyakran egyezik meg azonos kimenetekkel. Emellett két helyi változót hoz létre, a totalt és a countot. Ezek segítségével az egyezések gyakorisága könnyen kiszámolható. Az algoritmus a végén egyszerűen csak elosztja egymással a két adatot. [16]

A fordítás után következett a modell tanítása. A tanítás során beállítottam, hogy a hálózaton egyszerre 32 minta továbbítódjon. továbbá beállításra került, hogy a tanítási folyamat háromszor fusson vég. Ezt követően mentésre került a modell. A mentés utána a képernyőn megjelenik egy grafikon mely a tanítás sikerességét mutatja meg.

3.5 Grafikus felhasználói felület

Ezen bekezdés programkódját a **gui.py** fájl tartalmazza.

A feladat elkészítése során utolsó lépésként egy grafikus felületet készítettem. Ennek segítségével a felhasználó önmaga is kitudja próbálni és tesztelni is tudja a modellt. A program és a felület is tulajdonképpen nagyon egyszerű. Célja összeségében annyi, hogy úgyis lehessen tesztelni a különböző képeket, hogy látunk is valamit nem csak azt a két szót, ami a felismert tábla neve.

2. ábra A Grafikus felület használatának lépései
Forrás: saját forrás

A program indításakor láthatjuk, hogy tényleg egy egyszerű grafikus felületről van szó. Középen fent olvashatjuk, hogy ez egy közlekedési tábla felismerő program ez alatt található egy kép megjelenítő, ahova majd betudjuk tallózni a tesztelni kívánt képet. Alatta pedig egy gomb helyezkedik el „Kép betöltése” névvel. Ezt láthatjuk a 2. ábra bal oldali képén.

A középső képen láthatjuk, hogy a tesztelni kívánt kép betöltésre került a betöltés gomb megnyomásával. A képet pedig a képmegjelenítő már meg is jelenítette. Ezután megjelent jobb oldalt egy „Felismerés” gomb, amivel majd elindítható a kép tesztelése.

A jobb oldali képen látható, hogy a felismerés gomb megnyomása után a program elvégezte a felismerést és a kép felett megjelent a program által felismert közlekedési tábla neve.

4 A szoftver tesztelése

Ahogy az a címből is érezhető itt a kész program/szoftver működésének a teszteléséről lesz szó. A tesztelések segítségével sok információt állapíthatunk meg a program működéséről. Tesztelés során különböző visszajelzéseket kapunk, amik alapján megállapítható mi az, ami jól működik és mi az, ami nem. Ha minden jól működik akkor örülünk viszont, ha vannak hibák a programban azokat érdemes kijavítani. A program teszteléséhez először egy teszttervet készítettem.

4.1 Teszttervezés

A tervezés első lépéseként átgondoltam, hogyan kellene vagy hogyan lehetne alaposan letesztelni az elkészített szoftvert. A tesztelni kívánt képeket több csoportra osztottam különböző tulajdonságok alapján. Először könnyebben kezelhető bemenetekre akartam tesztelni és így haladva az olyan képekig, ahol már nehezebben ismerhetőek fel a táblák. Ebből a gondolatból adódóan először olyan képeket akartam tesztelni, ahol a kép középpontjában helyezkednek el és jól láthatóak a különböző táblák. Ezután jöhettek a fekete/fehér képek, amik már okozhatnak nehézséget a programnak. Majd ezután következhettek a kopott, illetve szennyeződésekkel rendelkező táblák.

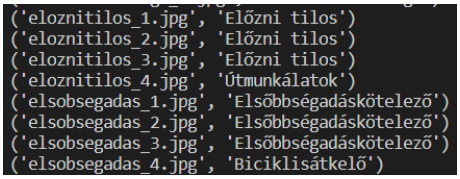
4.2 Tesztkészletek

A közlekedési táblák teszteléséhez megpróbáltam 4 csoportra osztani az általam tesztelni kívánt képeket. Ezek a tesztkészletek különböző nehezítéseket tartalmaznak a program számára. A négy tesztkészlet tulajdonságai a következőek:

- *„csak_a_tabla”* tesztkészlet. Itt csak olyan képeket találunk, amelyeken csak egy tábla található. A tábla középpontban helyezkedik el és csak egy nagyon kicsi háttér (természeti vagy sima fehér háttér) látható.
- *„fekete_feher”* tesztkészlet fekete-fehér alapú képeket tartalmaz. Ez a tesztkészlet a „csak a tábla” tesztkészlettől annyiban különbözik, hogy a kép az fekete-fehér.
- *„kopott”* tesztkészlet olyan táblákat tartalmaz, amelyeket már az időjárás vagy a nap egy kicsit megkoptatott. A képeken a táblák egy picivel nehezebben láthatók, illetve a szennyeződések ki is takarják egy részét a táblának.
- *„tobb_tabla_egy_kepen”* tesztkészletben, mint ahogy a név is sugallja, egy képen több tábla is szerepel. Itt arra voltam kíváncsi, hogy melyik táblát ismeri fel vagy nincs értékelhető eredmény.

4.3 Tesztek előkészítése, automatizálása, tesztelő algoritmus

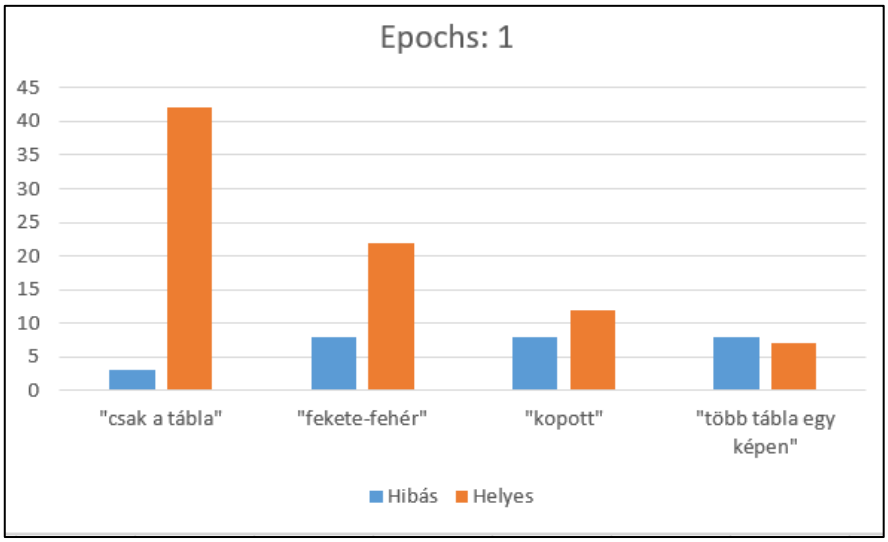
A teszttervezés során szembesültem azzal, hogy valószínűleg ahhoz, hogy minél pontosabb információkat kapjak vissza a modell működéséről, szükség lesz egy tesztprogramra. Egy ilyen tesztprogrammal nagyon sok időt takaríthatunk meg, közben sokkal eredményesebben tesztelhetjük a modellt. Másik előnyként megemlíthető, hogy a teszteredmények kiértékelése is gyorsabb. A mappastruktúrát is megpróbáltam úgy kialakítani, hogy minél könnyebb legyen a tesztelés. Van egy *Test_Images* mappa, amelyen belül 4 mappa található a tesztkészletek neveivel. További segítség a tesztek könnyű kiértékeléséhez, hogy a képek a képen látható tábla nevével rendelkeznek. A tesztprogram először kiírja az éppen tesztelt kép fájlnevét majd melléje az általa felismert tábla nevet. Ez egy könnyű tesztelést eredményez, aminek segítségével gyorsan lehet sok képet tesztelni.



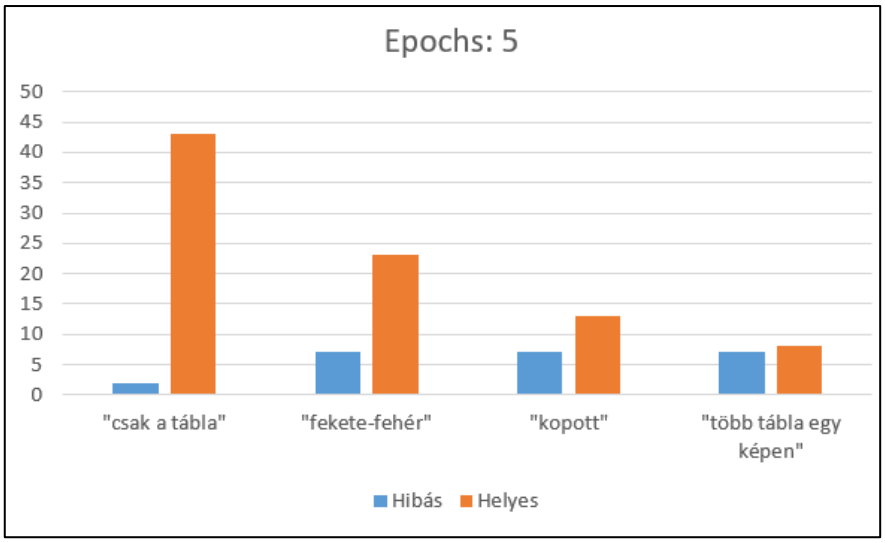
3. ábra Példa a program működéséről
Forrás: saját forrás

4.4 *Teszteredmények*

A tesztelés során két modellel teszteltem a képeket, a modellek közti különbség csupán annyi, hogy az egyik modell egyszer a másik modell ötször ment végig a tanító adatokon. A tesztkészletek különböző számú képeket tartalmaznak ezért a diagrammokon látható a hibás és helyes felismerések száma is:



4. ábra Egyszer tanított modell eredménye
Forrás: saját forrás



5. ábra Ötször tanított modell eredménye
Forrás: saját forrás

Az eredmények nagyjából ahogy várható volt úgy alakultak. Az első tesztkészlet a pontossági százalék nagyon magas, viszont ahogy egyre több nehezítés található a képeken úgy egyre kisebb ez a százalék. Ez valamilyen szinten köszönhető az tanítókészleteknek és a szoftver működésének is. A tesztelés során az világossá vált, hogy minél inkább a középpontban, fókuszban van a tábla és minél jobb a minőség (itt arra gondolok, hogy nincsenek például szennyeződések a táblán) annál nagyobb eséllyel ad a program vissza helyes táblanevet az adott képhez. Az is észrevehető, hogy a két tesztelés között

nincsen jelentős javulás, ez köszönhető annak is, hogy a egyszeres betanítás sikeressége 94% körüli a ötszörösé pedig 97% körüli volt.

Ahhoz, hogy a diagrammokat könnyebben lehessen értelmezni, azt érdemes letisztázni, hogy az első tesztkészlet 45, a második 30, a harmadik 20 és az utolsó 15 képet tartalmazott.

4.5 Összegzés

A tesztelés végén megállapítható, hogy biztos nem ez a leghatékonyabban működő közlekedési tábla felismerő szoftver, ellenben ha megfelelő bemeneti adatokat kap, akkor megkapjuk az elvárt kimenetet. Az észrevehető, hogy amint vannak nehezítések a képeken, abban a pillanatban a sikeres felismerések száma csökken. „Csak a tábla” tesztkészletnél a validált pontosság 90% feletti. A „fekete-fehér” tesztkészletnél a szoftver pontossága körülbelül 70-80% közé tehető. A „kopott” tesztkészletnél a pontosság 60% körül található. A negyedik egyben utolsó tesztkészletnél észrevehető, hogy a program inkább csak a középpontban található táblákat tudja érzékelni. Itt a pontosság 50% alatti.

5 Felhasználói leírás

5.1 A program telepítése

A program használatához először is szükség van a GitHub repository leklónozására. Ezt követően ahhoz, hogy a programot használni is lehessen szükség van a dokumentációban szereplő Phyton programcsomagok telepítésére, majd a felhasználó eldöntheti, hogy használja a repositoryban található modellt vagy betanítja a modellt és azt használja. Ha megvan a modell a program már használható is.

A program bármilyen Phyton fejlesztőkörnyezetben elindítható, de ajánlott Visual Studio Code-t használni. Azért javasolt mert ahogy már a dokumentáció korábbi részében is kifejtettem nem foglal nagy helyet és könnyen használható.

A Phyton programcsomagok telepítésénél érdemes odafigyelni, hogy az adott programcsomagok hányas verzióját telepítjük, mert előfordulhat, hogy a programcsomagok különböző verziói nem képesek egymással együttműködni és ez futási hibákat eredményezhet. Fokozottan érdemes odafigyelni, hogy a *tensorflow* programcsomag melyik *numpy* programcsomagot támogatja.

5.2 Felhasználói felület használata

A program felhasználói felületének használata nagyon egyszerű. A program indítása után a felhasználó a „Kép betöltése” gomb segítségével a számítógépről tud képet betallózni, amin szeretné, hogy a program felismerje a rajta található táblát, szándékosan nem írok táblákat mivel ez a program csak egy képen csak egy táblát képes felismerni, amennyiben több táblát szerepel a képen, az hibás eredményeket okozhat. Miután a kép betöltése megtörténik a felhasználói felület jobb oldalán megjelenik egy „Felismerés” gomb, amelynek segítségével el lehet indítani a felismerést. A program bezárása a felület jobb felső sarkában található X-szel könnyen megoldható.

5.3 Tesztprogram használata

Amennyiben a felhasználó saját tesztképekkel szeretné tesztelni a programot és nem csak 1-1 képet szeretne tesztelni. abban az esetben szükség van a tesztprogram változtatására. A tesztképeket tartalmazó mappát el kell helyezni a program mappájában. Ez az egyik teendő, a másik teendő pedig az, hogy a tesztprogram kódjában át kell írni a mappa nevét. Ez a legegyszerűbb megoldás mert így az elérési útvonalat nem kell mindig megváltoztatni. Amennyiben a képeket tartalmazó mappa máshol található a számítógépen abban az esetben a teljes elérési útvonalat is meg kell változtatni.

5.4 Összegzés

Összeségében a program nem igazán nevezhető felhasználó barát programnak. Tehát megállapítható, hogy van mit fejlődni ezen a téren. Jelenleg a felhasználói leírás tartalmazza azt, amit a felhasználónak tudnia, ismernie kell a programról. Remélem a program használata ezáltal nem okoz gondot egyetlen felhasználó számára sem. Problémamentes program használatot kívánok!

6 Felhasznált irodalom

[1] https://en.wikipedia.org/wiki/Traffic-sign_recognition
[2] Gyarmati Péter (2019): Gondolatok a mesterséges intelligencia, gépi tanulás kapcsán. Mesterséges intelligencia – interdiszciplináris folyóirat, I. évf. 2019/1. szám. 31–39.
[3] Fazekas István 2013. Neurális Hálózatok
[4] Hadházi Dániel (2018): Mély konvolúciós neurális hálózatok
[5] https://benchmark.ini.rub.de/
[6] Visual Studio Code weboldala Retrieved 3. Jan. 2021, from https://code.visualstudio.com/
[7] Tensorflow programcsomag Pythonhoz Retrieved 3. Jan. 2021, from https://pypi.org/project/tensorflow/
[8] Numpy programcsomag Retrieved 3. Jan. 2021, from https://numpy.org/
[9] Pandas programcsomag Retrieved 3. Jan. 2021, from https://pandas.pydata.org/
[10] Scikit-learn programcsomag Retrieved 3. Jan. 2021, from https://scikit-learn.org/stable/
[11] Keras programcsomag Retrieved 3. Jan. 2021, from https://keras.io/
[12] Matplotlib programcsomag Retrieved 3. Jan 2021, from https://matplotlib.org/tutorials/introductory/pyplot.html
[13] ReLU Retrieved 3. Jan. 2021, from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
[14] MaxPooling2D Retrieved 3. Jan 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D?fbclid=IwAR1WULBGGRaBmZ7RoorDhmkY4h0iNmDnf6oiEwCT7Xw-wp8CHrD33_yi_T0
[15] Dropout Retrieved 3. Jan 2021, from https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout?fbclid=IwAR1yuUA_8olDadUJIzsW_-eO-lkJVOWk6XDdfNQKKgbNMA6EgOMuNwjgL-E
[16] Keras programcsomag Retrieved 3. Jan 2021, from https://keras.io/
[17] További hivatkozások
[18] További hivatkozások
[19] További hivatkozások
[20] További hivatkozások
[21] További hivatkozások
[22] További hivatkozások
[23] További hivatkozások
[24] További hivatkozások
[25] További hivatkozások
[26] További hivatkozások
[27] További hivatkozások
[28] További hivatkozások
[29] További hivatkozások
[30] További hivatkozások
[31] További hivatkozások
[32] További hivatkozások
[33] További hivatkozások
[34] További hivatkozások
[35] További hivatkozások
[36] További hivatkozások
[37] További hivatkozások
[38] További hivatkozások
[39] További hivatkozások
[40] További hivatkozások
[41] További hivatkozások
[42] További hivatkozások
[43] További hivatkozások
[44] További hivatkozások
[45] További hivatkozások
[46] További hivatkozások
[47] További hivatkozások
[48] További hivatkozások
[49] További hivatkozások

[50] További hivatkozások
[51] További hivatkozások
[52] További hivatkozások
[53] További hivatkozások
[54] További hivatkozások
[55] További hivatkozások
[56] További hivatkozások
[57] További hivatkozások
[58] További hivatkozások
[59] További hivatkozások
[60] További hivatkozások
[61] További hivatkozások
[62] További hivatkozások
[63] További hivatkozások
[64] További hivatkozások
[65] További hivatkozások
[66] További hivatkozások
[67] További hivatkozások