EÖTVÖS LORÁND TUDOMÁNYEGYETEM
INFORMATIKAI KAR

Közlekedési táblák automatikus térképezése

DIPLOMAMUNKA
TÉRKÉPÉSZ MESTERSZAK

Készítette:
Dusek Bence

Témavezető:
Dr. Gede Mátyás
egyetemi docens
ELTE Térképtudományi és Geoinformatikai Intézet

Budapest, 2020

EÖTVÖS LORÁND TUDOMÁNYEGYETEM

INFORMATIKAI KAR

TÉRKÉPTUDOMÁNYI ÉS GEOINFORMATIKAI TANSZÉK

DIPLOMAMUNKA TÉMABEJELENTŐ

Hallgató adatai:

Név: Dusek Bence

Neptun kód: FLWLBV

Képzési adatok:

Szak: térképész, mesterképzés (MA/MSc)

Tagozat: Nappali

Belső témavezetővel rendelkezem

Témavezető neve: *Dr. Gede Mátyás*

munkahelyének neve: *ELTE IK Térképtudományi és Geoinformatikai Tanszék*

munkahelyének címe: *1117, Budapest, Pázmány Péter sétány 1/A.*

beosztás és iskolai végzettsége: *egyetemi docens*

A diplomamunka címe: Közlekedési táblák automatikus térképezése

A diplomamunka témája:

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben diplomamunka témájának leírását)

A dolgozat célja egy olyan rendszer felépítése, ami egy sztereo kamerapár képeit feldolgozva felismeri a kamerák látómezejébe kerülő közlekedési táblákat és megállapítja azok relatív helyzetét. Mindezt nem hagyományos programozás, hanem deep learning és neurális hálózatok alapján elkészítve. A rendszert helyzetmeghatározó eszközökkel (GPS, IMU) kiegészítve a táblák abszolút helyzete is meghatározható, azokból geoadatbázis építhető.

Budapest, 2020.11.22.

TARTALOMJEGYZÉK

1. BEVEZETÉS

A XXI. század egyértelműen az *információs technológia* korának tekinthető. Olyan forradalmi változások tanúi lehetünk, amely változás globális méreteket ölt. A technológiai fejlődés soha nem látott mértékű, és ez a gyorsuló modernizáció ma is alakítja és át is alakítja világunkat. Néhány évtizede még elképzelhetetlenek voltak a ma márt ismert és alkalmazott eszközök, technológiák, elég csak a mikrochipekre, az internetre vagy a dolgozat vázát is jelentő mesterséges intelligenciára gondolni.

Természetesen felmerülhet, hogy a mesterséges intelligencia témaköre és a szerző tanulmányai nincsenek paritásban egymással. Hogyan és miért foglalkoznak a térképészetben ezzel a témával, érdemes-e vele egyáltalán? Érthet-e hozzá olyan, aki nem programtervező végzettségű, lehet-e kellő tudása ahhoz, hogy egy ilyen programot megírjon? Ezek mind olyan kérdések, amelyek bennem is felmerültek, de reményeim szerint dolgozatommal sikerül eloszlatnom a kételyeket.

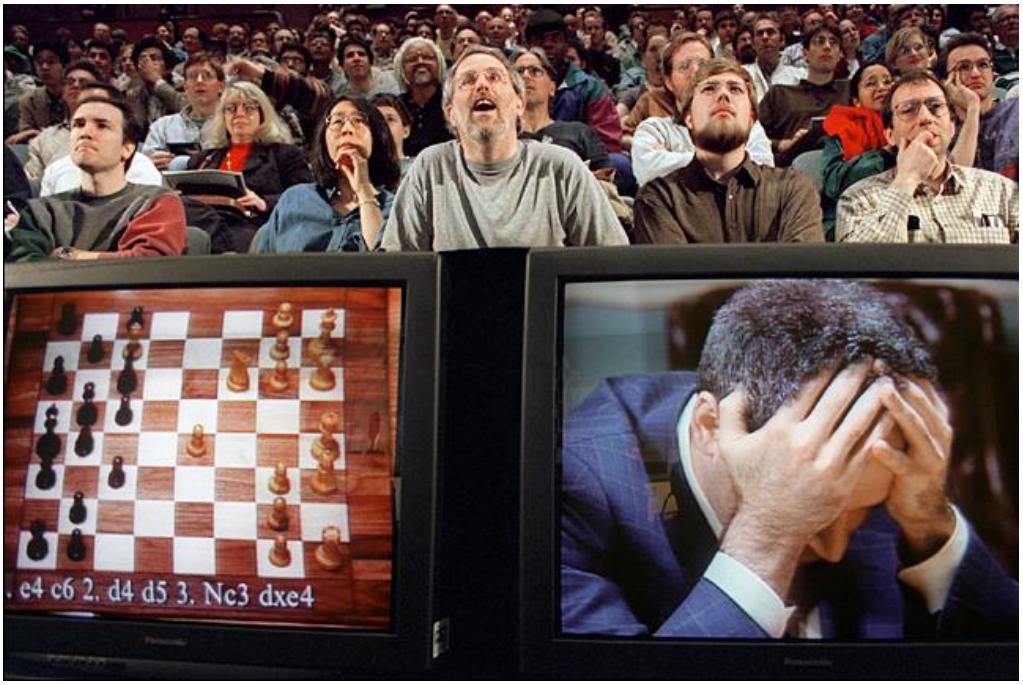
Meggyőződésem volt, hogy – mint oly sok tudományterületen – ebben az esetben is érdemes legalább megpróbálni egy új eszköznek/technológiának az adott tudományba, a térképészetbe történő implementálását, az információs technológia korában szocializálódott fiatalként pedig mindig is fogékony voltam az új és innovatív megoldásokra, kihívásokra. Bár a dolgozat megírása előtt kevés ismeretanyaggal rendelkeztem a machine- és deep learning területéről, mégis inspiráló volt, hogy egy olyan tudományág, olyan kutatási területében mélyülhetek el, amivel még relatíve kevesen foglalkoztak előttem.

2. AZ INTELLIGENS SZÁMÍTÓGÉPEK

2.1. Mesterséges intelligencia

A téma feldolgozása előtt feltétlenül definiálnunk kell az artificial intelligence (azaz A.I. vagy magyarul mesterséges intelligencia) fogalmát. Különösen azért van szükség erre, mert a köznyelvben használt definíciók (sci-fi A.I., videójáték A.I.) csak részben fedik le a tudományos magyarázatot.

A mesterséges intelligenciakutatás egyik legfőbb kérdése, hogy a számítógép hogyan lesz képes olyan problémákat megoldani, amelyek csak a gép számára jelentenek kihívást, viszont az ember számára puszta intuíció alapján megoldhatóak. Működése teljesen az általunk betáplált szabályokon múlik, ezek alapján reagál a gép, olyan helyzetekre, amiket lefednek ezek a szabályok (ROSEBROCK, 2017). Tehát a végcél egy olyan program, ágens létrehozása, amely hasonló hatékonysággal oldja meg a komplex problémákat, mint az ember, de olyan módon, mint a számítógép. Ha ezt sikerülne tökéletesíteni, akkor az ember (intuíció) és a gép (gyorsaság, logika) agyának legjobb tulajdonságait lenne lehetséges ötvözni, és temérdek feladatot lehetne automatizálni.

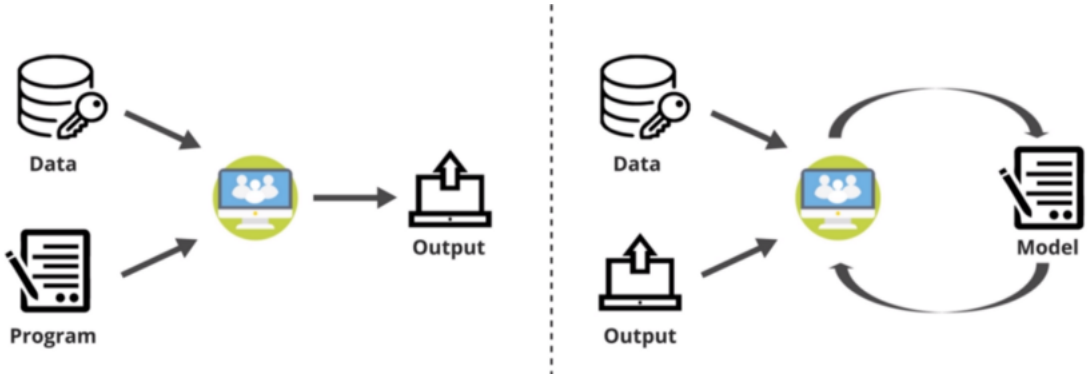


1. ábra
A Deep Blue nevű IMB számítógép legyőzi Garri Kaszparov sakk bajnokot 1997-ben.

A tudósok már több mint 70 éve foglalkoznak és kutatnak a témában, de erőforrás, megfelelő mennyiségű adat és kiforrott algoritmusok hiányában csak az utóbbi egy-két évtizedben sikerült olyan látványos fejlődést elérni (ROSEBROCK, 2017), mint amilyen például a Deep Blue is (1. ábra). Azon belül pedig az utolsó pár évben találhatunk olyan megvalósított, több tudományterületen is óriási népszerűségnek örvendő projekteket, amik korábban csak a sci-fi művekben léteztek (például chatbot, arcfelismerés, önvezető autó).

2.2. Gépi tanulás

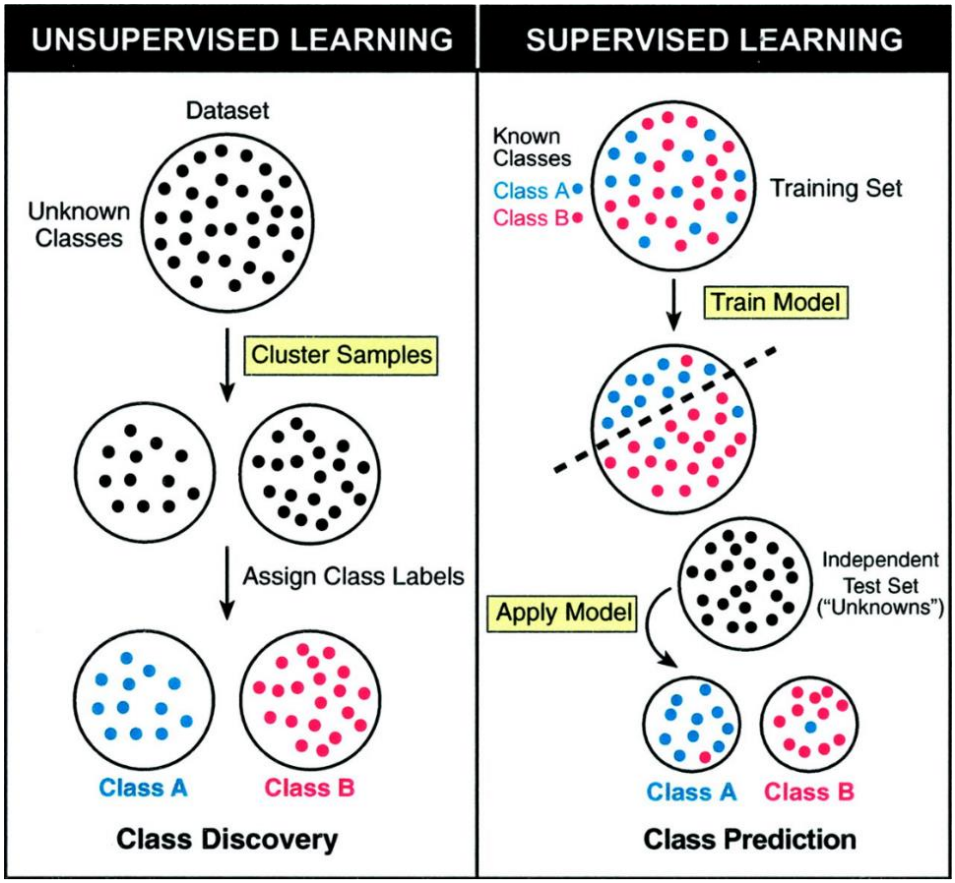
Az A.I. tehát összetett feladatok gyors, automatizált elvégzésére képes, de csak előre meghatározott szabályok alapján. Ennek egyik részterülete a machine learning (azaz M.L. vagy magyarul gépi tanulás), amely megközelítés olyan algoritmust alkalmaz a már előkészített adatainkra, hogy a bennük előforduló mintákat képes legyen felismerni és ezek alapján tanulni, felülbírálni magát. Tehát nem előre meghatározott szabályok, hanem adathalmazok és minták szerint működik (HADAWALE, 2020).



2. ábra
Tradicionális kontra „gépi tanulásos” megközelítése egy problémának.
A tradicionális módszer esetében a kódnak egzakt módon mondjuk meg hogyan oldja meg a feladatot, míg a másik módszer esetében létrejön egy modell, ami folyamatosan „tanítja” a programot (HADAWALE, 2020).

A szakirodalom két nagy alkategóriára osztja a gépi tanulási módszereket: supervised (felügyelt) és unsupervised (felügyelet nélküli). Több is létezik (semi-supervised, reinforcement stb.), de ezek számunkra most nem relevánsak. Az előbbi algoritmusnak a lényege, hogy tanító adatbázis minden egyes adatához tartozik egy – vagy több – közvetlen információ is. Más megfogalmazással, „felcímkézett” adatokat használunk, és ezekkel mondjuk meg a gépnek, hogy milyen kimenetet várunk el tőle adott esetben. Ha ez a

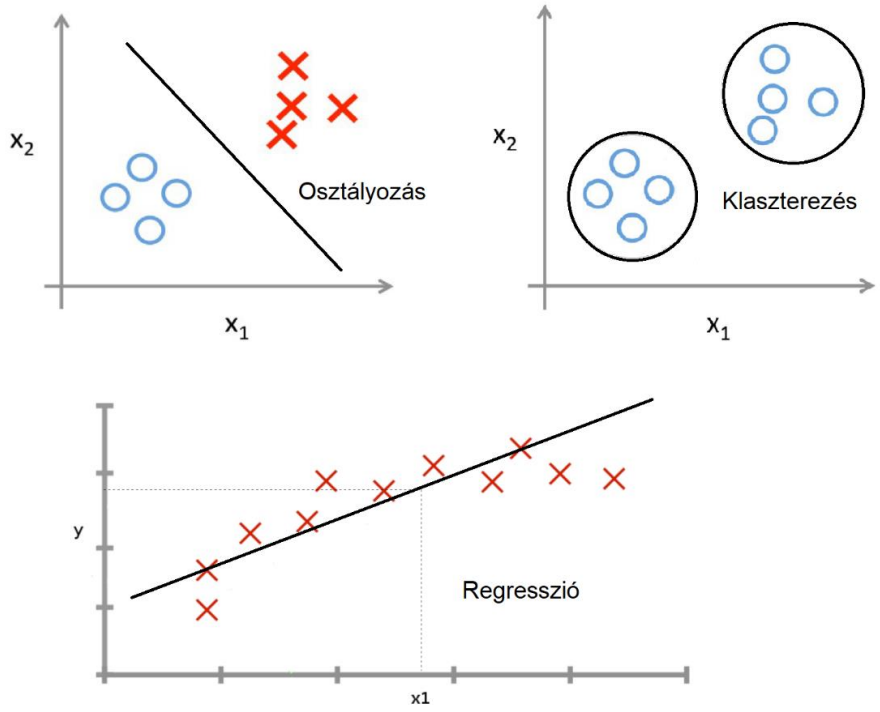
kimenet helytelen, akkor a gép újra számol és javít. Ezek a címkék a felügyelet nélküli tanulás adatai számára nem érhetőek el, itt magának a számítógépnek kell feltérképeznie az adatbázis belső struktúráit (ORMÁNDI, 2013), (HULLÁM ET AL., 2016).



3. ábra
Nem felügyelt és felügyelt tanulás.

A tanító adatbázis adataitól és a felhasználás céljától függően alkalmazhatóak a specifikusabb tanító eljárások, ezeket a 4. ábra három példája szemlélteti A felügyelt algoritmusok körébe tartoznak az osztályozó és a regressziós algoritmusok. Annak függvényében, hogy diszkrét vagy folytonos kimenetet várunk el a modellünktől, döntünk a fajtájukról. Az osztályozó esetében a bemeneti adatok két vagy több csoportra oszlanak, és a modellnek ugyanennyi diszkrét számú kategóriát kell létrehoznia a kimenetben (pl. modell, ami megmondja egy e-mailről, hogy spam-e vagy sem). A regressziós algoritmusok esetében ugyanakkor nem tudunk a címkék alapján éles határvonalat húzni ezen a kategóriák között, ugyanis általában a címkékben szereplő adatok számszerűek (pl. modell, amivel ingatlanok árait tudjuk előre jelezni) (ORMÁNDI, 2013).

A nem felügyelt algoritmusokból legismertebb a klaszterezési algoritmus. Lényege, hogy a gép saját maga csoportosítja az adatokat mindenféle címke vagy előzetes információ nélkül, nem lehet előre tudni, hogy hány kimeneti kategória fog keletkezni (ORMÁNDI, 2013). Legjellemzőbb példa erre a marketinges A.I.-k, amelyeknek az a célja, hogy a felhasználót összegyűjtött vásárlási szokásai alapján különböző kategóriákba sorolja, annak érdekében, hogy nagyobb valószínűséggel tudjon neki olyan termékeket kínálni, amelyek érdekelhetik az adott ügyfelet.

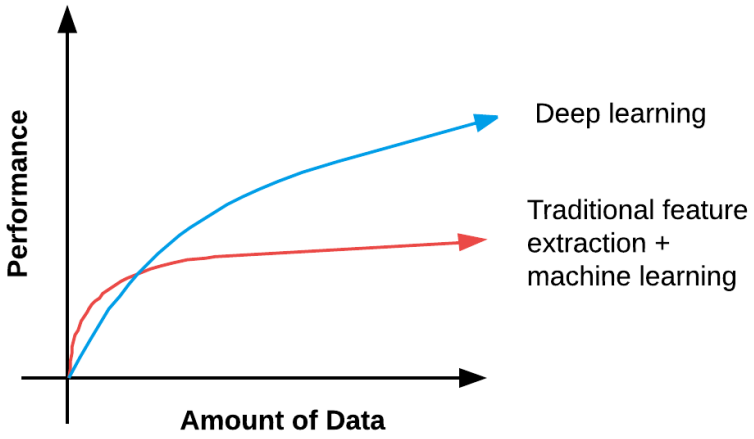


4. ábra
A legismertebb machine learning algoritmusok.
A fekete egyenesek és körök egy-egy lehetséges modellt ábrázolnak.

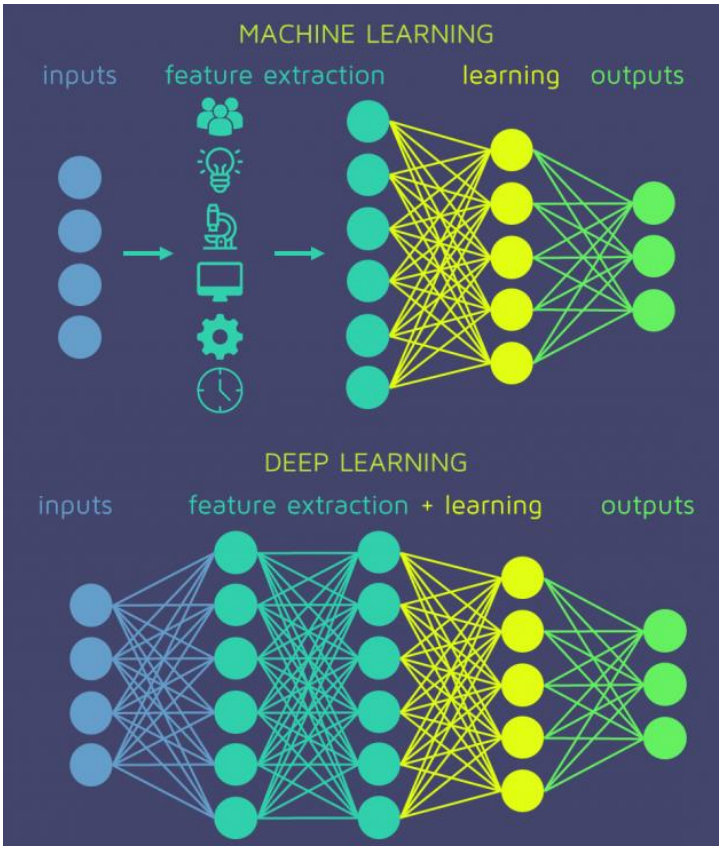
2.3. Mélytanulás

Annak ellenére, hogy az M.L. esetében nem adunk meg pontos, szabályszerű utasításokat a gépnek munkája elvégzésére vonatkozóan, mégis segítségre szorul abban, hogy milyen jellemzőket, karakterisztikákat nézzen, ami alapján meg tud különböztetni két egymástól eltérő jelenséget. A deep learning (azaz D.L. vagy magyarul mélytanulás) esetében nincs ilyen korlát (LECUN ET AL., 2015) (5. és 6. ábra), ugyanis *„úgy különbözteti meg nagy biztonsággal a kutyát a macskától, hogy közben nem magyarázza el a*

rendszernek senki, hogy mit jelent a kutya és a macska. A deep learning nem utánozza az embert, nem tudása van, hanem tudáselsajátítási képessége." – állítja Szabados Levente, A.I. szakértő (TRAPP, 2018).



5. ábra
A machine- és a deep learning teljesítményének különbsége az adatok mennyiségének függvényében.



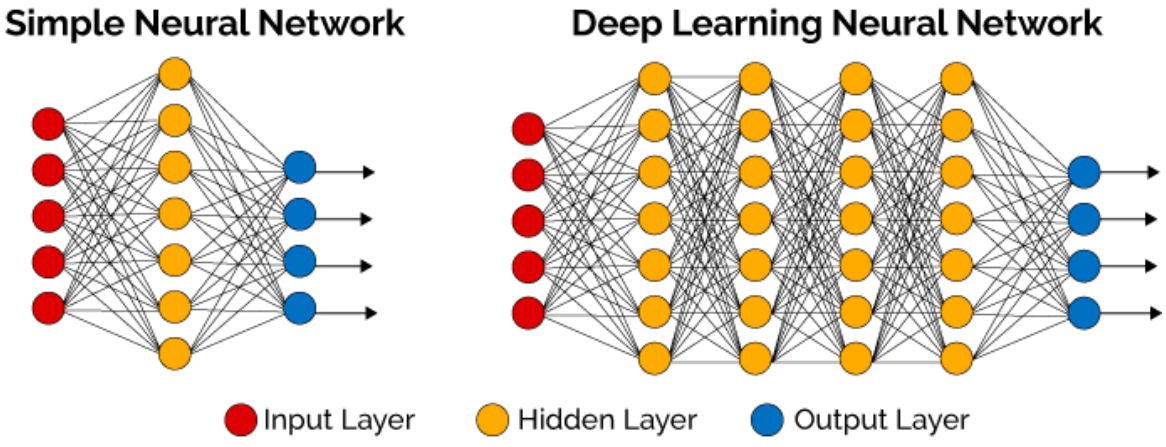
6. ábra
A machine- és a deep learning közti fő különbség az, hogy ki végzi el a „feature extraction" munkafolyamatot, az ember vagy a gép.

2.4. Neurális hálózatok

2.4.1. Felépítés

A mélytanulás a gépi tanulásnak egy olyan alkategóriája, melynek létrejöttét elsődlegesen az emberi agy működése inspirálta. Ezen módszerrel olyan modellek hozhatóak létre, amelyek az idegrendszeri, neurális hálózatunk biológiai tevékenységeit próbálják meg rekonstruálni (ROSEBROCK, 2017). Ennek neve az angol szakzsargonban Artificial Neural Network, azaz A.N.N., magyarul Mesterséges Neurális Hálózat.

Ezek az A.N.N. modellek (7. ábra) rétegekből (angolul layer) épülnek fel, amelyekből háromféle fordulhat elő: a bemeneti réteg, a kimeneti réteg és a rejtett réteg. Az első kettő layer funkcióját egyértelműsíti a nevük, az egyiknél a modell megkapja a nyers adatait, és elkezdi a feldolgozásukat; a másiknál kiadja őket az általunk meghatározott kategóriák szerint szortírozva; a rejtett rétegek pedig ennek a folyamatnak a számításait végzik. Minden utóbbi kategóriába eső réteg feladata az, hogy az őt megelőző layer kimenete alapján tanítsa a modellt újabb – az azonos kategóriájú adatokat összekötő – jellemzők felismerésére. Minél több rejtett réteg van, annál apróbb jellemvonások és tulajdonságok alapján képes vizsgálni adatokat a modell, legyenek bármilyen minőségűek a vizsgált adatok. (GOODFELLOW ET AL., 2016)



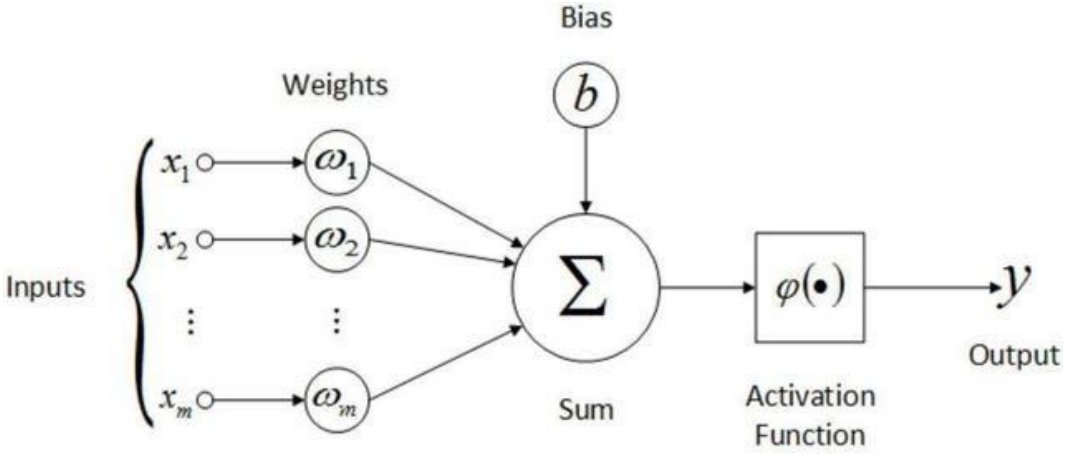
7. ábra
Egy egyszerű, kevés rétegű, sekély neurális hálózat (bal) és egy deep learninges, több rétegű, mély neurális hálózat (jobb).

A neurális hálózatok egyik egyszerű és meghatározó fajtája a Deep Feedforward (azaz D.F.F.), melynek rétegei a 7. ábrán látható módon állnak kapcsolatban egymással. Magát a feldolgozást és a munkát – a szó szoros értelmében véve – nem a layerek, hanem az őket alkotó csomópontok, node-ok vagy neuronok végzik. Minden réteg előre meghatározott számú csomópontokból áll.

Mindegyik node végez valamilyen szintű számításokat, amely számítások aztán jelként továbbhaladnak egyik neuronról a másikra, feltételezve, hogy azok kapcsolatban állnak egymással. Ezek a kapcsolatok mind súlyozottak, ami segít a modellnek eldönteni, hogy az egyes nexusok mennyire relevánsak. Ha két neuron relációja között pozitív előjelű a súly, akkor a rajta átérkező jelek – a súly értékkel megsokszorozva – felerősödnek, azt kommunikálva ezzel a modellnek, hogy ezek a szignálok hangsúlyosak a végeredmény szempontjából. Negatív előjelű súly esetében azonban a jelek gyengülnek, a modell pedig ezt úgy fogja értelmezi, hogy a node kimenete kevésbé jelentős a legvégső kategorizálásnál (ROSEBROCK, 2017).

2.4.2. Aktivációs függvény

Az viszont, hogy egy neuront bemeneti jelzések érnek, önmagában még nem jelent semmit. Az emberi agyban lévő neuronok is elektromos szignálokat küldenek egymásnak, amiket aztán továbbítanak a következőknek, és azok ugyanúgy a következőknek. Azonban ezeknek az inputoknak kellően nagynak és erősnek kell lenniük, hogy aktiválni tudják a

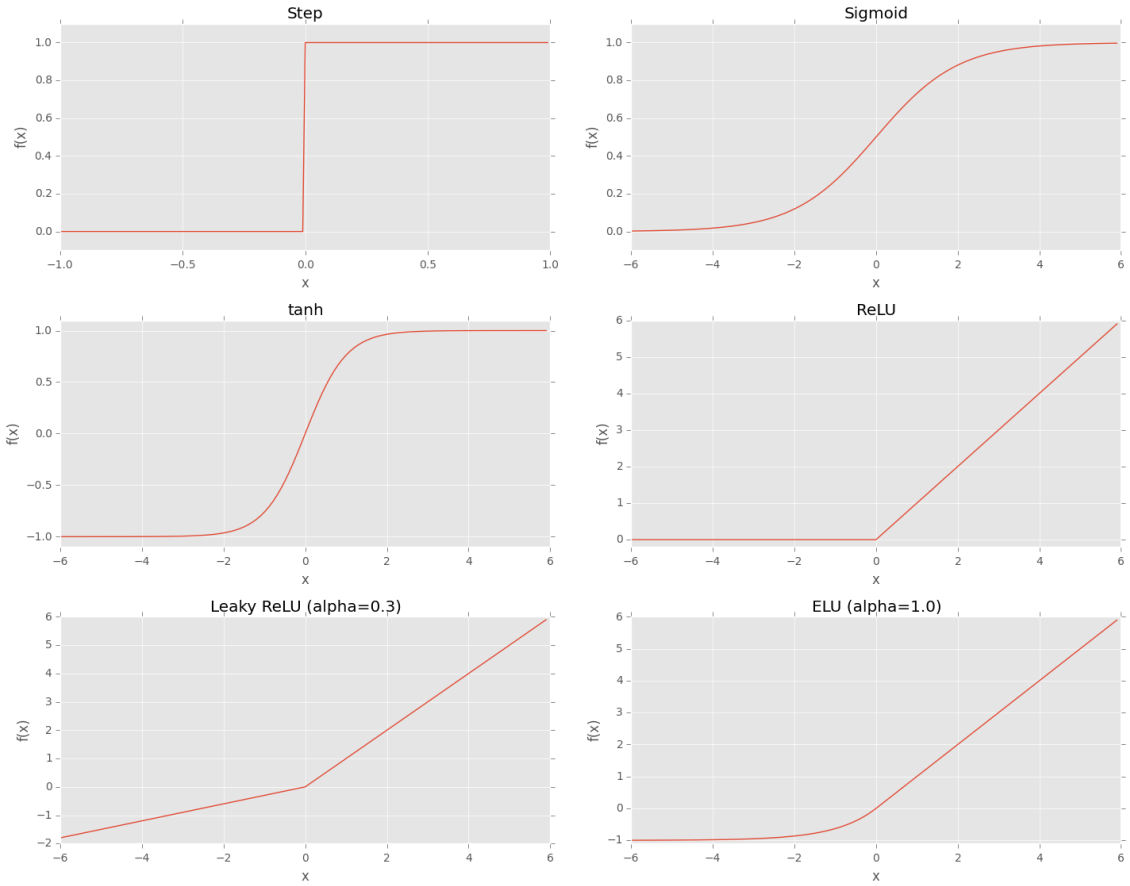


8. ábra
Egy egyszerű neurális hálózat, ami veszi az inputok (x) – és az eltolósúly (b) – súlyozott összegét (Σ) a súlyok (ω) alapján, majd ez az összeg átfut egy aktivációs függvényen (φ). A függvény tartalmazza a küszöbértéket és ennek értelmében eldönti, hogy aktiválódjon-e a neuron vagy sem (ROSEBROCK, 2017).

további neuronokat, amik aztán ugyanígy folytathatják a műveletet. Ha ez nem teljesül, akkor a jel elhal (ROSEBROCK, 2017).

Ez a minta szolgáltatta a fő inspirációt az A.N.N. működtetéséhez, mégpedig azt, hogy egy neuron aktiváltságának nincsenek mértékei, hiszen az egy bináris művelet. Viszont fontos tudni, hogy az érték, amit továbbítani fog, az nem bináris, hanem valamilyen előre megadott felső- és alsó határ közti eredmény lesz. Ez pedig akkor történik meg, ha a jelek erősségének összessége elért egy bizonyos küszöbértéket, ahogy azt a 8. ábra is reprezentálja. A képen látható modell matematikai egyenletként leírva: $f(\varphi), ahol\ \varphi = \sum_{i=1}^{m} \omega_i x_i + b_i$. (SANDERSON, 2017)

Ahhoz, hogy lássuk, mitől működik jól a neuron aktiváció, meg kell ismerkednünk az aktivációs függvényekkel. Ezek a neurális háló nemlinearitásáért felelős, differenciálható függvények (NIELSEN, 2015). Minden fajtájának megvannak az előnyei és hátrányai, azonfelül mindegyik hozzájárult valamilyen formában a technológia fejlődéséhez. A 9. ábrán a teljesség igénye nélkül látható néhány fontosabb függvény.



9. ábra
A legismertebb aktivációs függvények.

Áttekintve: az első három függvény a „klasszikusnak” számító függvények, a maradék három pedig „modern”, „21. századi” függvények. Tradicionálisan a sigmoid és a tanh függvények voltak a leggyakrabban használatosak a mesterséges hálózatok betanításához, azonban ezeket egyre inkább kiszorítják az újabb, hatékonyabb függvények. Jelenleg a ReLU – valamint ennek a számtalan variánsa, mint például a Leaky ReLU, ELU– számít a leginkább használatos aktivációs függvénynek a deep learninges rendszereken belül (ROSEBROCK, 2017).

Ezen függvények működésének az alapja a következő: a modell a bemeneti-, súly- és eltolósúly paraméterek összege alapján aktiválja a fontosabb neuronokat az aktivációs függvény segítségével, ezzel előállítva egy számszerű kimeneti értéket, amely valamilyen felső- és alsó határ között helyezkedik el (leggyakrabban ezek a 0 és 1 értékek, de a függvénytől függ) (DEEPLIZARD, 2017). És vajon mi történik akkor, ha nem jó vagy nem elég jó értékeket ad a modellünk? Ha pedig ezek nekünk nem elég jók, akkor hogyan fogja tudni, hogy mit, milyen irányba és mekkora mértékben kell megváltoztatnia az adott állapot megszüntetéséhez?

Ha a fenti kérdésekre tudjuk a választ, akkor azt is megértjük, hogyan zajlik egy modell betanítása, ugyanis ez a folyamat nem más, mint egy optimalizációs probléma megoldása (SANDERSON, 2017). Úgy gondolom, érdemes a kérdésfeltevés sorrendje szerint megvizsgálni a válaszokat, kezdve a hogyannal, amire jelen esetben a veszteségfüggvény a válasz.

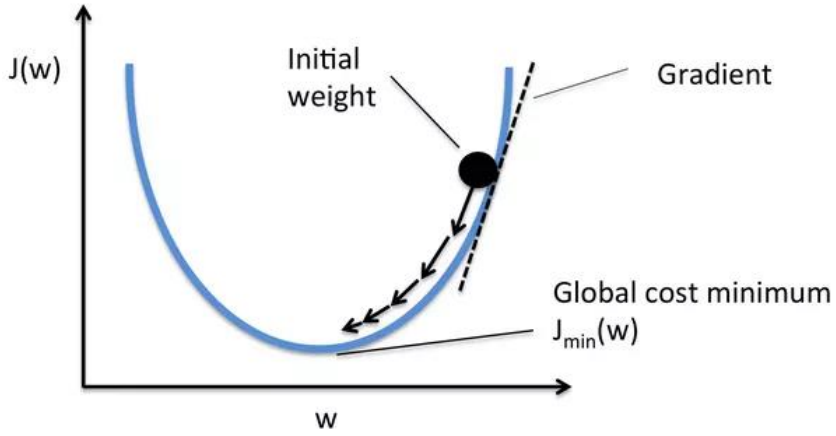
2.4.3. Optimalizálók és tanulási ráta

A veszteségfüggvény lényegi funkcióját tekintve rendkívül egyszerű: megmutatja, hogy a modellünk által megállapított kimeneti értékeink mennyire felelnek meg a valóságnak. Minél pontosabb a predikció, annál kisebb a veszteségfüggvény értéke.

Az átlagos négyzetes eltérés alapú veszteségfüggvényen keresztül jól megérthető a koncepció. A modell a tanítás során minden egyes tanító adatra kiszámít egy hibaértéket, vagyis veszi az összes kimeneti értéket, amit az adat után számolt ki. Ismert számára, hogy ezeknek a kimeneti neuronoknak a valóságban milyen értékeket kellene felvenniük, szóval ezen számok szerint képes kiszámítani a csomópontok kapott és valós értékének a különbségét, majd azt négyzetre emelni. Ezt és a többi tanító adat kapott hibaértékét

átlagolja a modell, és ennek a folyamatnak a legvégén jön ki eredményül a modell munkájának minőségét vizsgáló szám, amely segítségével már azt is meg tudjuk mondani, hogy mit és milyen irányba kell változtatnunk (SANDERSON, 2017).

A mélytanulás kifejezésben a tanulás nem a gép által végzett tényleges memorizálást jelenti, hanem a munkáját ellenőrző veszteségfüggvény értékének a minimalizálását. Viszont ahhoz, hogy ezt az értéket tudjuk csökkenteni, be kell vezetnünk az optimalizációs metódusok, röviden optimalizálók fogalmát. Ezeknek az a feladata, hogy segítsék a modellt a súlyok módosításában, így az képes legyen megbízhatóbb predikciók előállítására. Ehhez azonban szükség van a veszteségfüggvény gradiensére is. Legegyszerűbb esetben elég, ha ezt a gradienst csak kivonjuk a súlyokból, ez a 10. ábrán is látható gradiensereszkedés. Ez az eljárás iteratív módon kiértékeli a paramétereinket – azaz a súlyokat –, kiszámolja a veszteségértékünket, majd kis lépésenként a megfelelő irány felé haladva minimalizálja ezt az értéket (RAICEA, 2017).



10. ábra
Gradiensereszkedés.
Ideális esetben mindig a globális minimumát keressük a veszteségfüggvénynek, de valamikor az is megfelelő, amikor olyan lokális minimumot találunk, amire azt mondhatjuk: „Az már elég jó!”

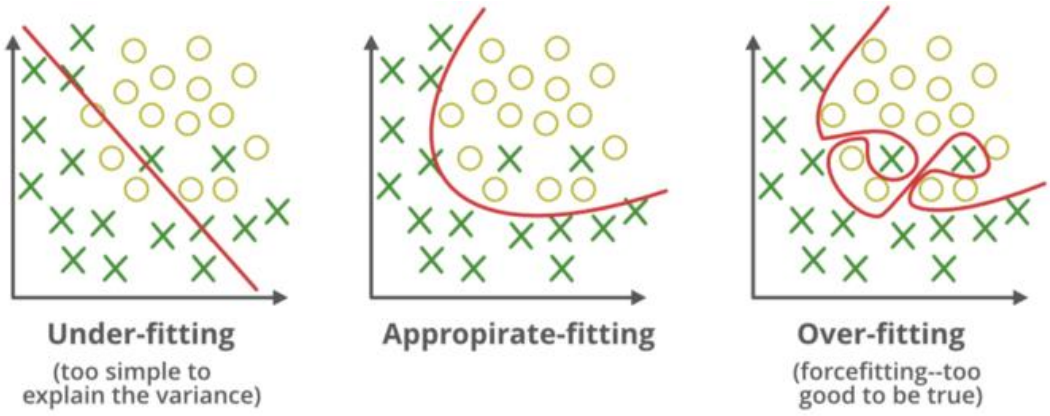
Fontos érték a modell szempontjából az ún. tanulási ráta, ami megadja, hogy a minimalizáló lépések pontosan mekkorák legyenek. Ha ezzel az értékkel beszorozzuk a gradienseinket, majd azokat kivonjuk az addigi súlyainkból, megkapjuk az új, aktualizált súlyainkat. Ez a viszonylag kicsi – általában 0,01 és 0,0001 közötti – érték egyike azoknak a hiperparamétereknek, amiket nehéz elsőre jól beállítani. Ugyanis, ha túl nagyra sikerül,

könnyen és észrevétlenül átléphetünk a keresett minimumértéken, ha viszont túl kicsi, akkor jóval több időbe fog telni, míg megtaláljuk ezt az értéket (DEEPLIZARD, 2017).

2.4.4. Regularizáció

A másik ilyen fontos, kézzel állítható változó a regularizáció. Ez egy olyan metódus, amely a minimális változtatásokat eszközöl a tanuló algoritmuson, de ettől a modellünk képes lesz a hatékonyabb generalizálásra. Ez azt jelenti, hogy javul a modell teljesítménye azon új, bemeneti adatok esetében, amik nem voltak részei a tanuló adatbázisnak (GOODFELLOW ET AL., 2016).

Hasonlóan a térképészetben ismert generalizációhoz, itt is az a lényeg, hogy az adott paraméterekkel rendelkező adathalmazunkat úgy általánosítsuk, hogy az adataink egyes jellemzői még jól megkülönböztethetőek legyenek, ugyanakkor mégse túl részletesek és elaprózottak. Utóbbi eset akkor áll fenn, ha egyáltalán nem vagy keveset regularizálunk, illetve ezzel ellentétes, amikor túl nagy a regularizáció mértéke, és túlgeneralizál a modellünk (11. ábra).

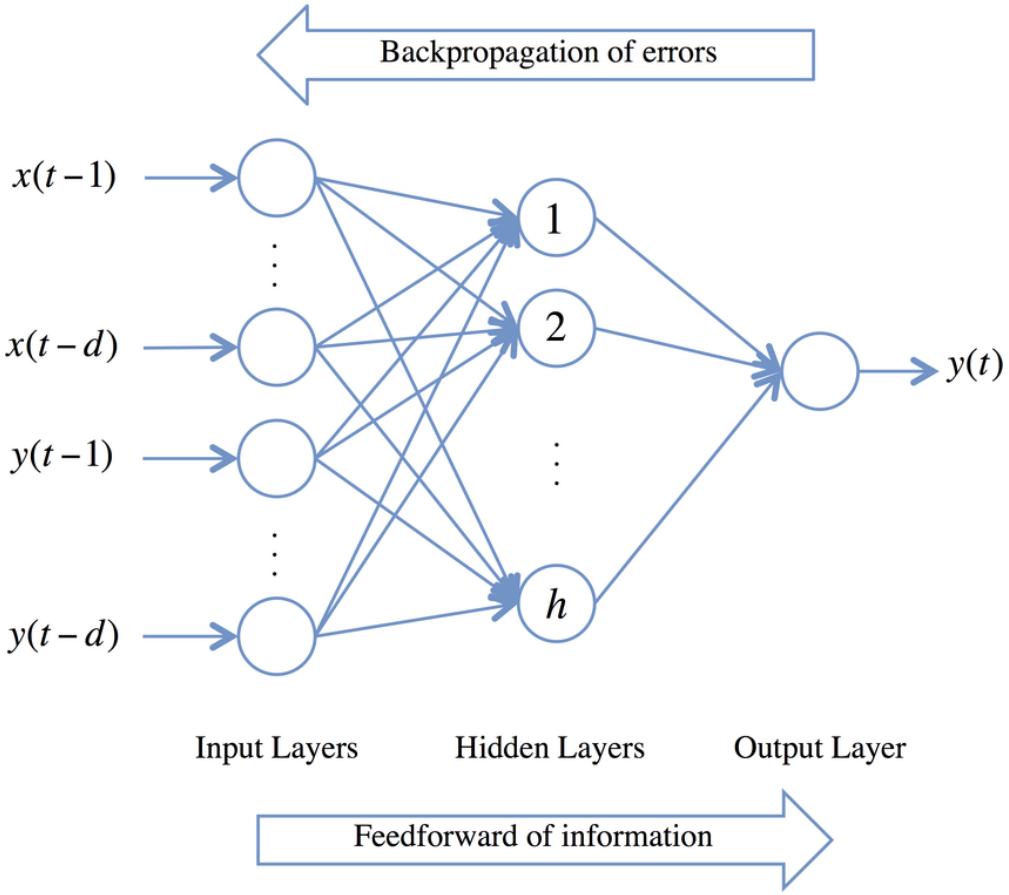


11. ábra
Alulillesztett-, jól generalizált- és túlillesztett modellek.

2.4.5. Visszacsatolás

Említésre került már, hogy a modell a végleges predikciók helyessége alapján elkezd javítani a rendszer súlyain, és ezzel ideális esetben képes elérni a közel 0 veszteségértéket, de nem került kifejtésre, hogy ez részleteiben hogyan történik. Ezt a folyamatot – ami

gradiens számítás révén frissíti a súlyokat – hívjuk visszacsatolásnak vagy angolul *backpropagationnek*. Ez az eljárás valójában a fő eszköze a gradiensereszkedésnek, ennek segítségével képes kiszámítani a veszteségfüggvény deriváltját/gradiensét (ROSEBROCK, 2017).



12. ábra
A visszacsatolás az előterjesztéssel (feedforward) ellentétes irányban zajlik.

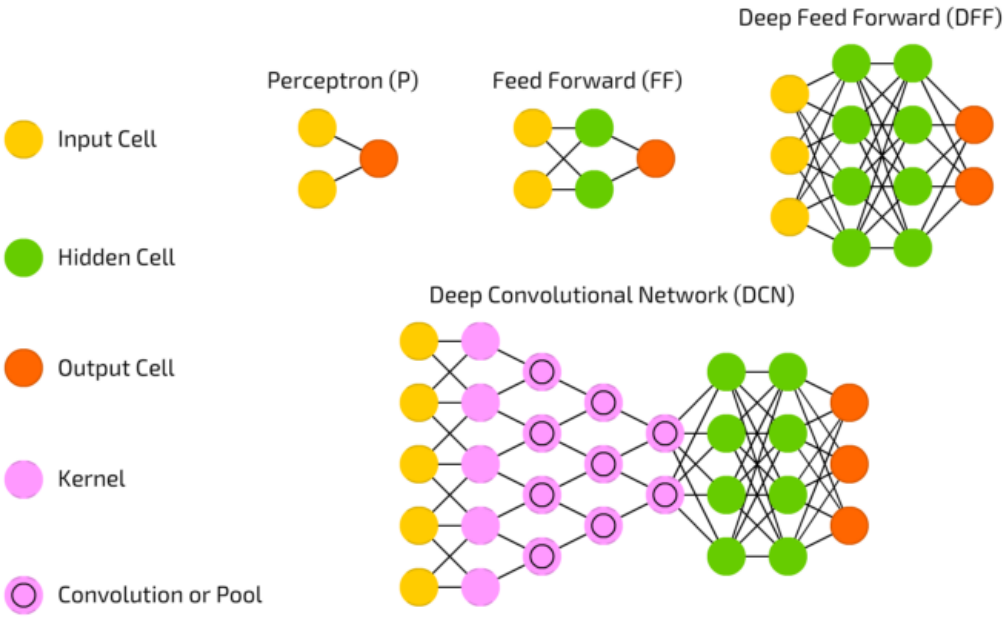
A gradiensereszkedés a veszteségértéket kétféle módon képes csökkenteni. Mivel az ismert, hogy a kimeneti réteg minden egyes neuronjának értéke az előtte lévő layer aktivációs értékeinek a súlyozott összege, így a képlet szerint, a súlyokat és/vagy az előző réteg aktivációs értékeit változtatja meg a függvény. Ezeket az aktivációs értékeket nem tudja közvetlenül modifikálni az algoritmus, de a súlyokon keresztül, indirekt módon van ráhatása ezekre az értékekre. Ez az eredmény úgy érhető el, hogy még egy réteget visszalép az eljárás, ahol az előző layer súlyait módosítja. Iteratív módon, addig zajlik ez a folyamat, amíg az algoritmus el nem éri a bemeneti réteget, de az ott lévő értékek már nem változnak, hiszen ott tárolódnak az általunk megadott bemeneti adatok (SANDERSON, 2017).

Azt is érdemes megemlíteni, hogy a metódus a lehető leghatásosabb módon fogja a súlyok értékeit frissíteni, hiszen ezzel érhető el legeredményesebben a cél, a veszteségfüggvény értékének a csökkentése. Ez viszont a megváltoztatandó súlyok gradienseinek a nagyságán fog múlni, hiszen minél nagyobb ez a változó, annál jelentősebb mértékű lesz a veszteségértékben bekövetkezett változás, függetlenül attól, hogy milyen irányúak ezek a deriváltak (DEEPLIZARD, 2017).

Visszatekintve a 7. ábrára, felmerülhet a kérdés, hogy hány rejtett rétegtől számíthat mélynek a mély, és mikortól lehet azt mondani, hogy ez a rendszer egy mély neurális háló. Ebben nincsen teljes konszenzus a szakértők között sem, hiszen mindez nem egyértelműen lehatárolható, ezért a válasz is csak szubjektív lehet. Ez számos tényezőtől függhet, de talán a legfontosabb jellemző az adott neurális hálózat típusa és rejtett rétegeinek száma lehet.

2.5. Konvolúciós neurális hálózat (C.N.N.)

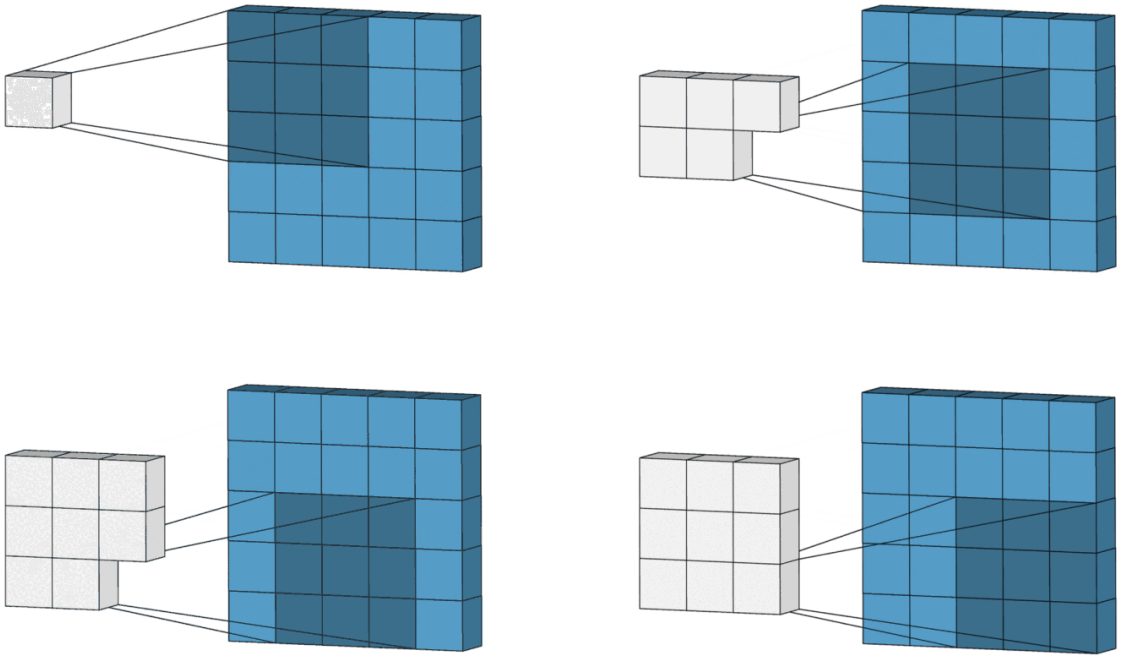
A „mélység” kérdésére a válasz egyértelműen személyfüggő és sokténye zős, ennek ellenére a kettőnél több rejtett réteggel rendelkező modell esetében már indokolt a mély hálózat megnevezés, attól függően, hogy milyen jellegű és komplexitású feladat végrehajtására lett megalkotva (ROSEBROCK, 2017).



13. ábra
Alapvető neurális hálózati fajták (perceptron, feed forward, deep feed forward) és a konvolúciós neurális hálózat egy speciális alfajtája a deep convolutional network. Abban az esetben indokolt a C.N.N. megnevezés használata az adott modellre, ha rendelkezik minimum egy konvolúciós rejtett réteggel.

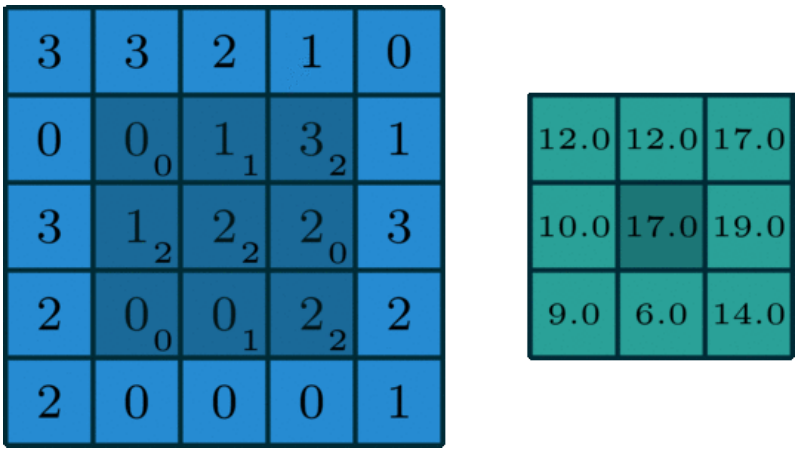
Az évtizedek során számottevően sok probléma megoldására használták fel az A.N.N.-t, és az idő, valamint a technológia előrehaladtával egyre nagyobb hatékonysággal voltak képesek ezek az architektúrák az adott feladatok elvégzésére. Így lehetséges az, hogy nagyszámú neurális hálózati fajta – pl. perceptron, feed forward – létezik, és közülük az egyik legjelentősebb a convolutional neural network (azaz C.N.N. vagy magyarul konvolúciós neurális hálózat).

A C.N.N.-t leginkább képfelismerési feladatokhoz alkalmazzák a mintázat detektáló tulajdonsága miatt (LeCun et al., 2015). Mintázatok alatt a képeken előforduló élek, alakzatok, textúrák és egyéb ehhez hasonló attribútumok értendők. Ez a művelet a modell speciális rejtett rétegeiben ún. konvolúciós rétegeiben zajlik le a filterek hatására. Ezek a filterek – vagy másnéven kernelek, szűrők – relatíve kisméretű mátrixok súlyértékekkel ellátva, amelyek mérete a fejlesztő előzetes beállításaitól függ. Általában célszerű ehhez páratlan számokat választani (3x3, 5x5, 7x7). Feladatuk a kétdimenziós bemeneti adatok végig „pásztázása” a 14. ábrán látható módon.



14. ábra
A 3x3-as kernel (árnyékolt kockák) az előre megadott egy pixeles lépésközzel halad végig az 5x5-ös bemeneti adat elemein (kék kockák). Ez a művelet a 3x3-as kimeneti adat elemeit (szürke kockák) eredményezi.

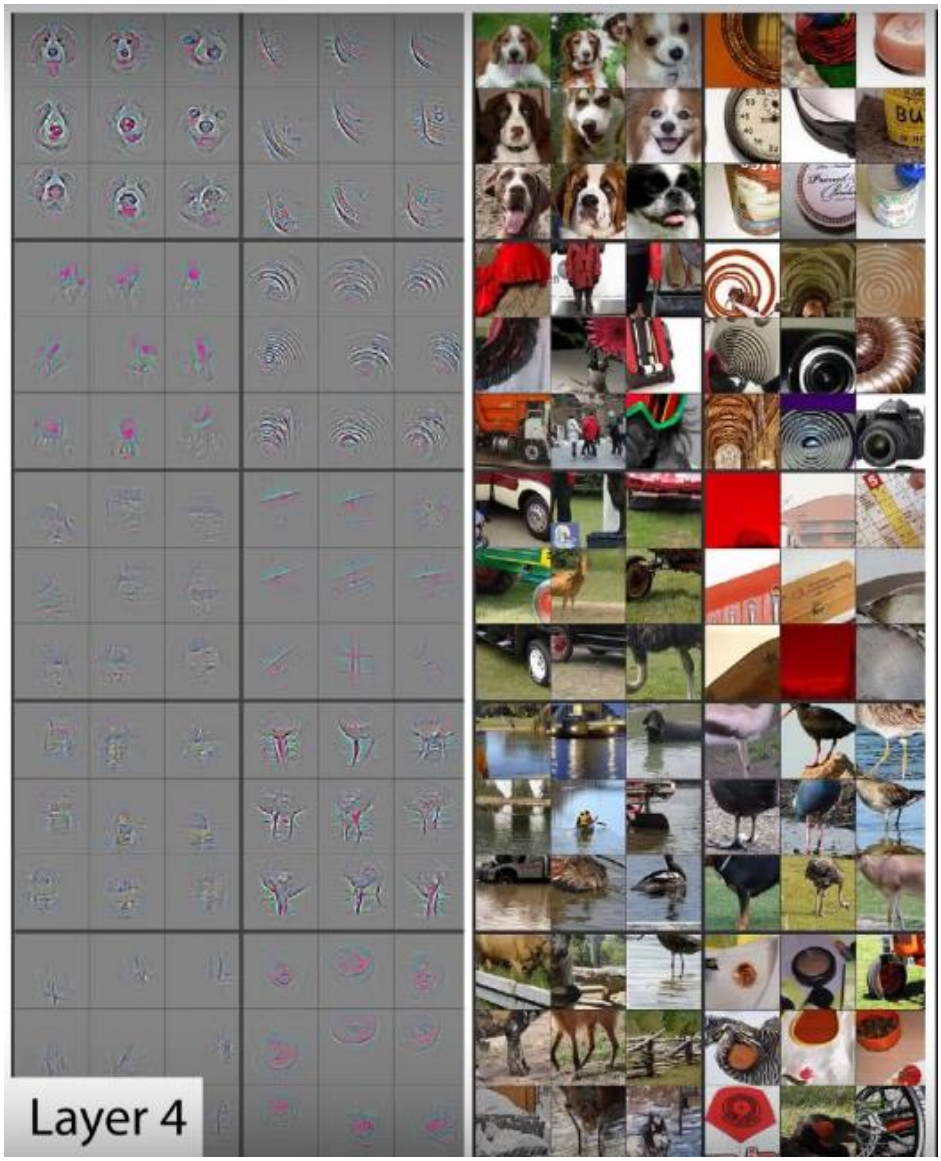
A 14. és 15. ábrán lévő elemek alapján megállapítható, hogy a konvolúciós rétegeknek szükségük van egy bemeneti csatornára (kék négyzetek) és egy kimeneti csatornára, vagy más néven aktivációs térképre (zöld négyzetek) a rendeltetésszerű működéshez. Ezen mátrixokban található értékek lényegében ugyanazok, amelyekről már az előző alfejezetben is szó esett. Lényegében, az itteni kimeneti értékek a súlyozott összegek, amik a filterben tárolt súlyok és a bemeneti csatornában lévő aktivációs értékek alapján kerülnek kiszámításra (SHAFKAT, 2018) a 15. ábrán is megfigyelhető módon.



15. ábra
Egy konvolúciós rétegben történő aritmetikai művelet.
Ez jelen esetben a bemeneti csatorna és a kernel értékei szorzatának az összegét jelenti.
Látható, hogy 9 elemből végeredményben csak egyetlen elem lesz a konvolválás után.

Az előző két ábrán a legegyszerűbb esetek kerültek bemutatásra, a valóságban ez annyiban eltérő, hogy egy layeren belül nemcsak egy, hanem akár több filter is feltérképezheti a bemeneti csatornát. Ezek száma meg fog egyezni az aktivációs térképek számával.

Az első layerben ezek a szűrők egyenként még mind más és más egyszerű jellemzők detektálására alkalmasak (horizontális-, vertikális-, diagonális élek), viszont ha a modell több konvolúciós réteggel is rendelkezik, akkor ezek a detektorok egyre komplexebb formákat lesznek képesek felismerni (körök, sarkok, görbék stb.). Megfelelő számú réteg esetén pedig már olyan összetettségű jelenségek észlelésére is alkalmassá válnak, mint a macskák farka, a kombájnok ajtaja vagy a stop táblák – természetesen a bemeneti adatok tartalmának függvényében (DEEPLIZARD, 2017).



16. ábra
Összetett formák, amiket a filterek már a modell 4. rétegében képesek detektálni (kutyafej vagy madárláb).

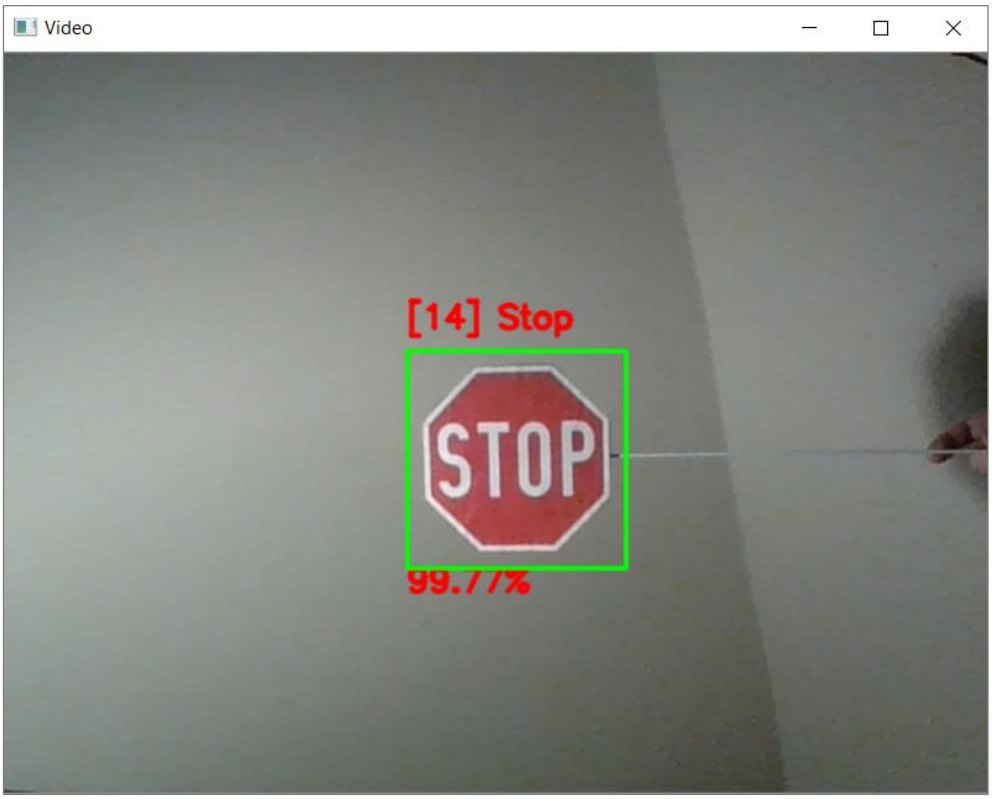
A program teljes megértéséhez szükséges alapfogalmak áttekintése után a dolgozat gyakorlati részének kifejtése következik, a tanító- és tesztprogramok elemzése, nevezetesen a jelzőtábla-felismerő rendszer építési folyamatának leírása.

3. JELZŐTÁBLA-FELISMERŐ RENDSZER ÉPÍTÉSE

3.1. Projekt előkészítése

3.1.1. Célok definiálása

A projekt elsődleges célja az volt, hogy térképészként betekintést nyerhessek egy olyan technológiának a működésébe, ami potenciálisan hasznossá válhat a tudományterület számára. Ehhez azonban az alapoktól kellett építkezni. A fentebb is részletezett alapvető fogalmak után a megszerzett tudást lépésenként át kellett ültetni a gyakorlatba. Ez a feladatrész egy – a modern autókban is található – jelzőtábla-felismerő rendszer kiépítésében manifesztálódott (17. ábra).



17. ábra
A betanított modell működés közben.

A témaválasztás fő indoka az volt, hogy a projekt létrehozásának beugró-haladó szintű nehézsége és jól dokumentáltsága ideálissá tette a technológia alapszintű és alapvető tanulmányozását. Számos jól működő, hasonló rendszert hoztak létre és helyeztek már

üzembe magáncégek, és számottevően sok ilyen architektúrát készítettek magánemberek is hobbi vagy oktatási célzattal, melyeknek nagy része – magyarázattal és dokumentációval mellékelve – ingyenesen elérhető az interneten. Ezenfelül, viszonylagosan jól illeszkedik a feldolgozott téma az adott tudományterületbe, az intézet már korábban is részt vett önvezető autó és okos város tematikájú projektekben. Ráadásképpen ez a munka megadhatja másoknak is azt az inspirációt, ami a témában további dolgozatokat, tudományos írásokat eredményezhet.

3.1.2. Hardveres és szoftveres eszközök

Az információs technológia jelen fejlettségi szintjének egyik nagy hozadéka, hogy könnyen és egyszerűen érhetőek el mindenki számára olyan eszközök, amelyek hihetetlenül nagy mennyiségű erőforrást hordoznak. Alig két évtized alatt szinte az össze számítógépes hardveres alkatrésznek egy – de bizonyos esetekben akár két – egész nagyságrenddel megugrott a számítási teljesítménye. Már van olyan erős CPU, GPU, RAM, tárhely, hogy egy ilyen projektet komfortosan képes az ember akár egy laptopról is megcsinálni. A feladat elvégzésére saját Acer Aspire XV laptopomat használtam, amit alapvetően játékra terveztek, de éppen ezért igen nagy teljesítményű, és jól alkalmazható ilyen esetekben is. A gép specifikációi a mellékelt táblázatban láthatóak.

Laptop hardveres specifikációi	
Processzor	Intel Core i7-7700HQ @ 2.80 GHz
Memória	16 (DDR4) GB
SSD	256 GB
HDD	1024 GB
Kijelző	15,6" IPS FHD LED matt
Videokártya	4 GB NVIDIA GeForce GTX 1050 Ti

Igaz, hogy ezek a nagy teljesítményű alkatrészek könnyen hozzáférhetőkké váltak, de sajnálatos módon így is költségigényesek. A projekt esetében használt szoftveres résszel azonban nem ez a helyzet. A programozási nyelv kiválasztása az egyik legmeghatározóbb mozzanat ilyen volumenű feladatnál, azért a *Pythonra* esett ez a választás, mert az utóbbi időben az egyik legnépszerűbb nyelvvé vált a neurális hálózatok és deep learning

fejlesztésekhez, köszönhetően annak, hogy az ember számára az egyik legegyszerűbben olvasható és tanulható programozási nyelvek közé tartozik (ROSEBROCK, 2017).

Számos mennyiségű, mélytanulással kapcsolatos implementált eszköz és függvénykönyvtár könnyíti meg a feladat elvégzését. Egyik ilyen a *Keras*, egy magasszintű deep learning API, ami eredetileg három különböző, alacsony szintű, neurális hálózati függvénykönyvtárat tudott alkalmazni a háttér számításokhoz: a *Theanot*, a *CNTK*-t és a *Tensorflow*-t. Mostanra azonban a Google teljes mértékben integrálta a *Kerast* a *Tensorflow* könyvtárba, így amikor installáljuk a *Tensorflow*-t, automatikusan települ a *Keras* is.

Ezenkívül még egyéb számításokhoz, képfeldolgozáshoz, vizualizációhoz és gépi tanuláshoz olyan fontos könyvtárak kerültek felhasználásra, mint például az *OpenCV*, *NumPy*, *Matplotlib*, *scikit-learn*, *imutils*. Továbbá fejlesztő környezet gyanánt a *PyCharmra* esett a választás, mivel intuitív kezelőfelülettel rendelkezik, és ezen keresztül könnyen feltelepíthetőek a fentebb említett *Python* csomagok.

18. ábra
A projekt során használt szoftveres eszközök logói.

3.2. A munka menete

3.2.1. Telepítés

Nulladik lépés a telepítés volt. Az előző alfejezetben felsorolt programok és könyvtárak installációja verzió inkompatibilitás miatt nem ment problémamentesen. Neurális hálózati architektúrák építésével kapcsolatban az első mozzanat, amit a fejlesztőnek el kell döntenie, hogy a gép melyik részével szeretné elvégeztetni a számításokat, a processzorral vagy a videókártyával. Előbbinek az előnye, hogy egyszerű és a *Keras* alapbeállításból azt választja ki, viszont az ilyen típusú feladatoknál lassan számol. Utóbbi már ezt nagyságrendekkel gyorsabban teszi, cserébe nehezebben üzemelhető be.

Éppen ezért a GPU meghajtásra esett a választásom, hiszen úgy véltem, hogy ilyen jellegű feladat esetében többet nyerek vele, mint amennyit veszítek, másfelől érdekelt, hogy mekkora mértékkel gyorsul ezen számítások sebessége. Beüzemelésének fő nehézségét a *Tensorflow*-val kompatibilis NVIDIA *CUDA* verziószámának pontos összeegyeztetése és ennek a mély neurális hálózatokra specializált függvénykönyvtárának, a *cuDNN*-nek a telepítése jelentette.

3.2.2. Tanító program

Ezek után a következő lépésként elkezdődhetett a modell üzembe helyezése. Egy ilyen feladatra általában két programkódot írnak: egyet, amelyik betanítja és egyet, amelyik teszteli „éles helyzetben” ezt a modellt, ergo a tanító adatoktól független és eltérő képeket kap a rendszer bemeneti értékként. Ehhez a projekthez végül nem saját építésű (HASSAN, 2020), ugyanis ilyen célra számos jól működő, legálisan és díjmentesen felhasználható tanító program létezik már, amik megfelelően vannak paraméterezve, A program German Traffic Sign Recognition Benchmark (GTSRB) adatbázis alapján állítja fel a modellt, mely a Németországban előforduló közlekedési táblák egy részét tartalmazza (STALLKAMP ET AL., 2012). Szerencsés módon ezek rendkívül hasonlóak a Magyarországon található jelzőtáblákhoz.

19. ábra
A GTSRB adatbázisban található 43 közúti jelzőtábla kategória.

A következő pár oldalon a tanító program működésének a fontosabb, lényegi mozzanatait szeretném bemutatni:

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3   import cv2
4   from keras.models import Sequential
5   from keras.layers import Dense
6   from keras.optimizers import Adam
7   from keras.utils.np_utils import to_categorical
8   from keras.layers import Dropout, Flatten
9   from keras.layers.convolutional import Conv2D, MaxPooling2D
10  from keras.preprocessing.image import ImageDataGenerator
11  from sklearn.model_selection import train_test_split
12  import os
13  import pandas as pd
14  import random
```

Ebben a kódrészletben a már említett könyvtárak kerülnek importálásra, a *NumPy* a numerikus számítások (1. sor), a *Matplotlib* az eredmények és az adatok vizuális reprezentációja (2. sor), az *OpenCV* a képi adatokkal kapcsolatos műveletek (3. sor), a *Keras* pedig a neurális hálózat felépítése céljából (4.-10. sor). A 11. sorban lévő `train_test_split` fogja szétválasztani a tanuló- és tesztadatokat az adatbázisból. A maradék

három könyvtár (12-14. sor) kizárólag egy-egy kisebb feladat elvégzéséhez kell (relatív elérési út beolvasása stb.).

```
18  path = "myData"  # folder with all the class folders
19  labelFile = 'labels.csv'  # file with all names of classes
20  batch_size_val = 50  # how many to process together
21  steps_per_epoch_val = 400
22  epochs_val = 50
23  imageDimesions = (32, 32, 3)
24  testRatio = 0.2  # if 1000 images split will 200 for testing
25  validationRatio = 0.2  # if 1000 images 20% of remaining 800 will be 160 for
                           validation
```

A fentiekben láthatóak a modellhez szükséges paraméterek beállításai. A `path` és a `labelFile` változókba kerültek rögzítésre az adatbázisban szereplő képek és a hozzájuk tartozó címkék elérési útjai. A `batch_size_val` azt tárolja, hogy mennyi adatot dolgoz fel a modell egyszerre, az `epochs_val` azt, hogy hányszor iterál végig a megadott mennyiségű adaton, amely mennyiséget pedig a `steps_per_epoch_val`-lal lehet megadni. Ez lényegében egy segítség a rendszernek, annak érdekében, hogy meg lehessen adni, hány batch/porciónyi adat után érjen véget egy epoch. Az `imageDimesions` a képek transzformáció utáni dimenzióját tárolja. Az 24-25. sorban lévő kód az ellenőrzéshez és a megerősítéshez használatos adatok arányszámát tartalmazza.

```
30  count = 0
31  images = []
32  classNo = []
33  myList = os.listdir(path)
34  print("Total Classes Detected:", len(myList))
35  noOfClasses = len(myList)
36  print("Importing Classes.....")
37  for x in range(0, len(myList)):
38      myPicList = os.listdir(path + "/" + str(count))
39      for y in myPicList:
40          curImg = cv2.imread(path + "/" + str(count) + "/" + y)
41          images.append(curImg)
42          classNo.append(count)
43      print(count, end=" ")
44      count += 1
45  print(" ")
46  images = np.array(images)
47  classNo = np.array(classNo)
```

Ezen kódrészletben – a paraméterek inicializálása, illetve változókba történő mentése után (30-35. sor) – történik a közlekedésitábla-kategóriák importálása minden adott osztályhoz tartozó képpel (37-44. sor). Ezeket az iterációk után a program eltárolja két *NumPy* tömbbe (46. és 47. sorok).

```
50  X_train, X_test, y_train, y_test = train_test_split(images, classNo,
                                                        test_size=testRatio)
51  X_train, X_validation, y_train, y_validation=train_test_split (X_train, y_train,
                                                          test_size=validationRatio)
```

A meglévő adatok és hozzájuk tartozó címkék felosztására a fentiek szerint kerül sor. Először a tanító adatokból és címkékből kerülnek kiválogatásra a tesztadatok és címkék a korábban definiált `testRatio` nevű változóban megadott értékhányados alapján, majd ugyanezen metódus mintájára történik a tanító- és az ellenőrző adatok és címkék szétválasztása a `validationRatio` szerint.

Ennek a feldarabolásnak a lényege a túlillesztés elkerülése, illetve a modell próbára tétele. Mindhárom adathalmaz elnevezése jól leírja azok gyakorlati funkcióját is. A tanító adatok egyszerűen arra szolgálnak, hogy betanítsák, valamint minden epoch végén fejlesszék a modellt, illetve az általa megadott predikciós értékeket. Ezzel szemben az érvényesítő adatoknak az a feladatuk, hogy a neurális modell képes legyen jól generalizálni és ne csak a tanító adathalmazban előforduló képeket kategorizálni. Ezen adatokra adott „jóslások” minősítése folyamatosan zajlik a tanító adatokéval párhuzamosan, s ha az utóbbi értékei érzékelhetően magasabbak, mint az előbbié, akkor van szó túlillesztésről. A tesztadatok csak a tanulás folyamata után kerülnek tényleges felhasználásra, a program így értékeli a modell betanításának sikerességét. Fontos megemlíteni, hogy a teszteléskor a rendszer nem ismeri a bemeneti adatokhoz tartozó címkéket, hiszen ezek a „terepen” sem lesznek elérhetőek.

A mélytanulásban az ’x’ jelölés általában az adatokat, az ’y’ pedig a hozzájuk tartozó címkét jelöli. Jelenesetben az ’x’ a közlekedési táblák képeit, az ’y’ pedig az egyes képen szereplő táblák fajtáját tartalmazza.

```
99  def grayscale(img):
100     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
101     return img
102
103 def equalize(img):
104     img = cv2.equalizeHist(img)
105     return img
106
107 def preprocessing(img):
108     img = grayscale(img)  # CONVERT TO GRAYSCALE
109     img = equalize(img)  # STANDARDIZE THE LIGHTING IN AN IMAGE
110     img = img / 255  # TO NORMALIZE VALUES BETWEEN 0 AND 1 INSTEAD OF 0 TO 255
111     return img
112
113
114 X_train = np.array(list(map(preprocessing, X_train))) # TO IRETATE AND
                                                          PREPROCESS ALL IMAGES
115 X_validation = np.array(list(map(preprocessing, X_validation)))
116 X_test = np.array(list(map(preprocessing, X_test)))
```

A 99.-től a 111. sorig a képek előfeldolgozása zajlik. A `grayscale` függvény végzi a képek színcsatornáinak megváltoztatását RGB-ről Grayscale-re, az `equalize` ezen szürkeárnyalatos képeken hisztogram kiegyenlítést hajt végre. A `preprocessing` függvényben meghívjuk az előző függvényeket, ami ezek után a kapott – 0-tól 255-ig terjedő – értéket normalizálja, tehát 0 és 1 közötti értékké alakítja. A 114-116. sorokban a program meghívja az utolsó függvényt, és minden egyes tanító, ellenőrző és tesztadaton lefuttatja azt, annak érdekében, hogy a modell képes legyen értelmezni ezeket az értékeket.

```
120 X_train=X_train.reshape(X_train.shape[0], X_train.shape[1], X_train.shape[2],1)
121 X_validation = X_validation.reshape(X_validation.shape[0],
                                        X_validation.shape[1],
                                        X_validation.shape[2], 1)
122 X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], X_test.shape[2], 1)
```

A hálózat gördülékenyebb működése érdekében a program átformálja a `.reshape()` paranccsal az adatokat, és kiegészíti azokat még egy dimenzióval. A gyakorlatban ez azt jelenti, hogy a képekhez rendel még egy kiterjedést, aminek az értéke 1 lesz.

```
125 dataGen = ImageDataGenerator(width_shift_range=0.1, # 0.1 = 10%
126                              height_shift_range=0.1, # IF MORE THAN 1 E.G 10
                                 THEN IT REFFERS TO NO. OF  PIXELS EG 10 PIXELS
127                              zoom_range=0.2, #0.2 MEANS CAN GO FROM 0.8 TO 1.2
128                              shear_range=0.1,  # MAGNITUDE OF SHEAR ANGLE
129                              rotation_range=10)  # DEGREES
130 dataGen.fit(X_train)
```

A 125. sorban definiált generátor átalakítja az adatbázisban szereplő képek szerkezetét, megváltoztatva a kódban felsorolt értékekkel a paramétereiket, valamint ezekkel a módosított képekkel kiegészíti a meglévő adatállományt is. Ezt a folyamatot idegen szóval augmentációnak nevezzük, és célja az, hogy egy képből – kis energiabefektetéssel – akár több, alapjában véve hasonló, de eléggé különböző kép generálódhasson. Ezzel a módszerrel általában a túlillesztés eshetőségét csökkentjük, illetve egy kis adatbázis méretét növeljük.

```
143  y_train = to_categorical(y_train, noOfClasses)
144  y_validation = to_categorical(y_validation, noOfClasses)
145  y_test = to_categorical(y_test, noOfClasses)
```

A 143-145. sorban végzi a program a meglévő tanító-, érvényesítő- és tesztcímkék transzformálását az eredeti alakjukból bináris – ún. ’one-hot’ – kódolásúvá, ugyanis a gép számára ezek csak ebben a formában értelmezhetőek. Az eredeti címkék átalakulnak nullákból és egyesekből álló vektorokká, amelyeknek a hossza megegyezik az adataink fajtáinak, kategóriáinak a számával (jelen esetben az adatbázisban lévő jelzőtáblafajták számával). Azért ’one-hot’, mert minden esetben egy darab index veszi fel a vektoron belül az 1 értéket, és ez attól függően történik, hogy a vektor melyik kategóriának, címkének feleltethető meg (DEEPLIZARD, 2017).

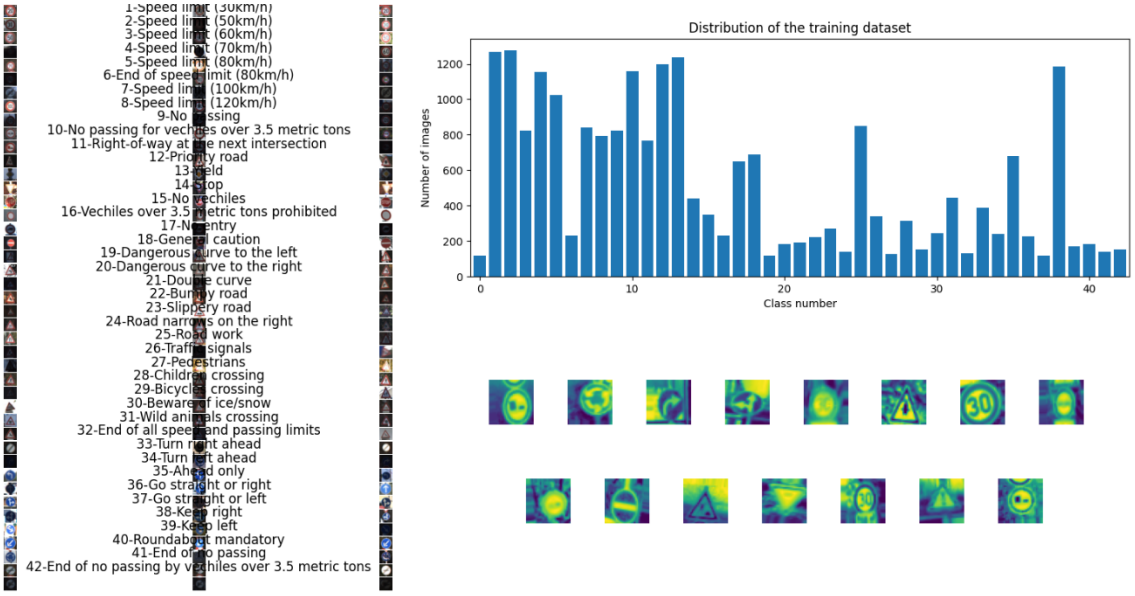
```
149 def myModel():
150     no_Of_Filters = 60
151     size_of_Filter = (5, 5)  # THIS IS THE KERNEL THAT MOVE AROUND THE IMAGE TO
                                   GET THE FEATURES.
152     # THIS WOULD REMOVE 2 PIXELS FROM EACH BORDER WHEN USING 32 32 IMAGE
153     size_of_Filter2 = (3, 3)
154     size_of_pool = (2, 2)  # SCALE DOWN ALL FEATURE MAP TO GERNALIZE MORE, TO
                               REDUCE OVERFITTING
155     no_Of_Nodes = 500  # NO. OF NODES IN HIDDEN LAYERS
156     model = Sequential()
157     model.add((Conv2D(no_Of_Filters, size_of_Filter,
        input_shape=(imageDimesions[0], imageDimesions[1], 1), activation='relu')))
        # ADDING MORE CONVOLUTION LAYERS = LESS FEATURES BUT CAN CAUSE ACCURACY TO
          INCREASE
158     model.add((Conv2D(no_Of_Filters, size_of_Filter, activation='relu')))
159     model.add(MaxPooling2D(pool_size=size_of_pool))  # DOES NOT EFFECT THE
                                                          DEPTH/NO OF FILTERS
160
161     model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
162     model.add((Conv2D(no_Of_Filters // 2, size_of_Filter2, activation='relu')))
163     model.add(MaxPooling2D(pool_size=size_of_pool))
164     model.add(Dropout(0.5))
165
166     model.add(Flatten())
167     model.add(Dense(no_Of_Nodes, activation='relu'))
168     model.add(Dropout(0.5))  # INPUTS NODES TO DROP WITH EACH UPDATE 1 ALL 0
                                 NONE
169     model.add(Dense(noOfClasses, activation='softmax'))  # OUTPUT LAYER
170     # COMPILE MODEL
171     model.compile(Adam(lr=0.001), loss='categorical_crossentropy',
                      metrics=['accuracy'])
172     return model
```

A fent lévő kódsorozat tartalmazza a program lényegi részét, a konvolúciós neurális hálózati modellt. A 150-155. sorokban olyan, a C.N.N.-re jellemző különleges paramétereknek a beállítására kerül sor, mint például a képeken átfutó szűrők száma, mérete és a rejtett rétegekben fellelhető összes neuron száma. Ezután a függvény többi sorában a konvolúciós és összevonó (angolul pooling) layerek – ezekből épül fel a modell – kerülnek definiálásra.

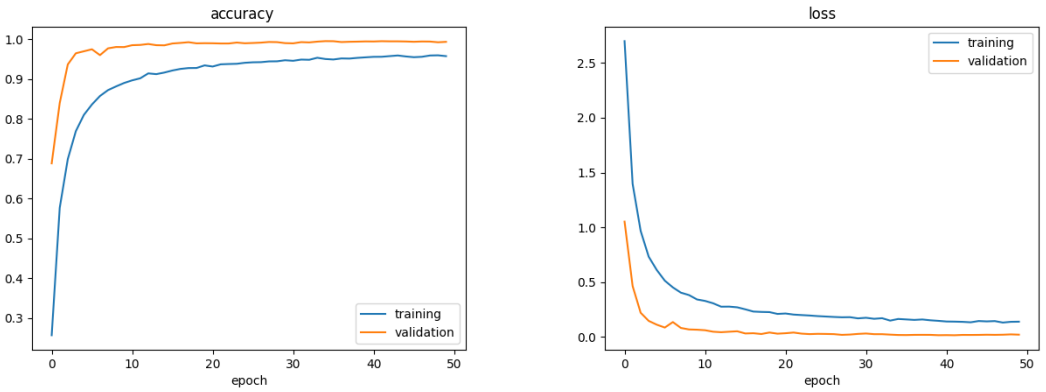
```
176 model = myModel()
177 print(model.summary())
178 history = model.fit_generator(dataGen.flow
                                  (X_train, y_train, batch_size=batch_size_val),
179                               steps_per_epoch=steps_per_epoch_val,
                                  epochs=epochs_val,
180                               validation_data=(X_validation, y_validation),
181                               shuffle=1)
```

Maga a modell betanítása .fit_generator segítségével zajlik. Paraméterként meg kell neki adni az adatokat és a 20-22. sorban tárolt változókat. A program lefuttatásának

legvégén még mutat a felhasználónak egy összegzést a tanítás sikerességéről és hatásfokáról, majd elmenti a kapott modellt egy előre megadott mappába.



20. ábra
A program futása közben megjelenő ábrák, melyek az adatbázisról prezentálnak vizuális információkat.
Bal: táblakategóriák példákkal;
Jobb felső: a képek kategóriák függvényében történő eloszlása;
Jobb alsó: példák az augmentált adatokra

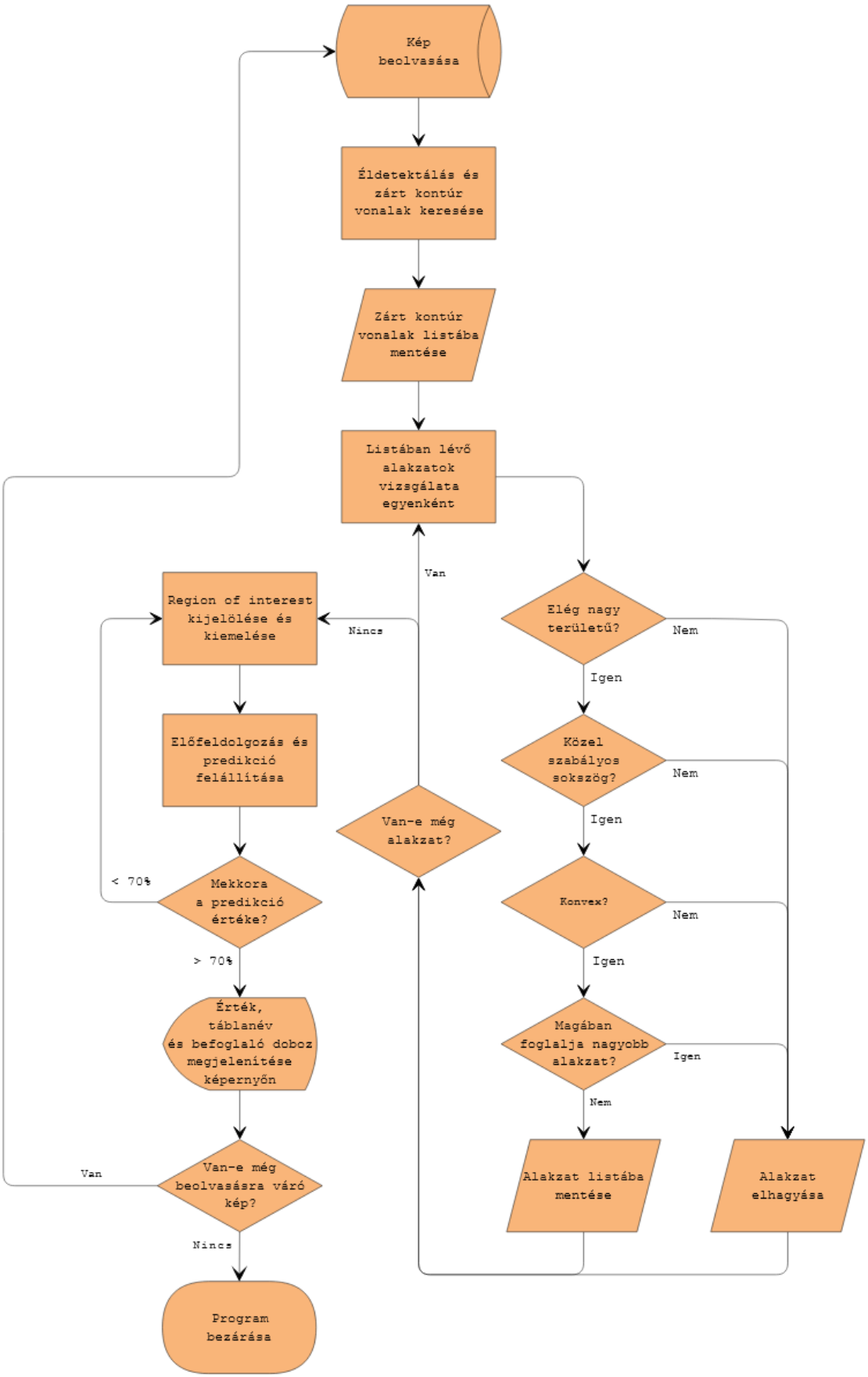



```
Epoch 48/50
400/400 [==============================] - 5s 14ms/step - loss: 0.1321 - accuracy: 0.9592 - val_loss: 0.0214 - val_accuracy: 0.9943
Epoch 49/50
400/400 [==============================] - 5s 14ms/step - loss: 0.1390 - accuracy: 0.9597 - val_loss: 0.0245 - val_accuracy: 0.9925
Epoch 50/50
400/400 [==============================] - 6s 14ms/step - loss: 0.1402 - accuracy: 0.9576 - val_loss: 0.0219 - val_accuracy: 0.9935
Test Score: 0.018655357882380486
Test Accuracy: 0.9946839213371277
```

21. ábra
A modell betanításának folyamata és végeredménye.
A felső két diagram megmutatja, ahogy az epochok elteltével nő a pontosság és csökken a veszteségérték, az alsó ábra pedig ezen végleges értékek számszerűsített alakjait.

3.2.3. Tesztprogram

A tesztprogram feladata a modell pontosságának próbára tétele, az, hogy meggyőződhessünk róla, ténylegesen azokat az eredményeket adja-e vissza, mint amelyeket fejlesztőként elvártunk. Ahhoz, hogy a modellt eredményesen lehessen tesztelni, szükség lesz újabb, független bemeneti adatokra, ami legegyszerűbben és leghatékonyabban egy videókamera segítségével biztosítható. Ezek lehetnek előre rögzített kép-, illetve videófájl formátumúak. Jelen esetben az egyszerű és kényelmes tesztelés kedvéért a laptopom beépített kameráját használtam, mint valós idejű bemeneti forrást. Majd ezen forráson bejövő képeken olyan éldetektáló szűrőket futtattam le, amelyek segítségével meghatározható volt, hogy van-e – és ha igen, hol – az adott képen olyan szabályos alakzat, ami potenciálisan valamiféle közlekedési táblaként azonosítható (22. ábra).



22. ábra
Folyamatábra a program működéséről.

A projekt ezen része saját fejlesztésű, aminek programkódját az alábbi alfejezetben fogom részletesen elemezni:

```
8   import numpy as np
9   import cv2
10  import imutils as ut
11  from keras.models import load_model
12
13  # Import the trained model and defining a font
14  model = load_model("my_model")
15  font = cv2.FONT_HERSHEY_SIMPLEX
```

Hasonlóan a tanító programhoz itt is importálásra kerültek a *NumPy*, az *OpenCV* és a *Keras* könyvtárak. Az *imutils* egy képfeldolgozáshoz használt programkönyvtár, ami hasznos funkciókkal egészíti ki az *OpenCV* parancsait. A 14. sorban a betanított modellt töltettem be a programmal.

```
100 # Input and camera settings
101 cap = cv2.VideoCapture(0)
102 cap.set(3, 640)
103 cap.set(4, 480)
104
105 while True:
106
107     # Reading the image
108     r, originalImage = cap.read()
109
110     # Applying edge detector and finding the contours of the objects
111     edgeDetect = autoCanny(originalImage)
112
113     contours, hierarchy = cv2.findContours(edgeDetect, cv2.RETR_CCOMP,
        cv2.CHAIN_APPROX_SIMPLE)
```

Miután a 101. sorban megadtam a laptopom webkameráját mint a bemeneti adatok forrását, a 105. sortól egy olyan végtelen ciklus definiálására volt szükség, ami biztosította, hogy az inputok küldése a program számára folyamatos módon történjen, ellenkező parancsig. Minden egyes pillanatképet az `originalImage` változó tárol el, majd az automatikusan paraméterezett Canny éldetektor segítségével a képeken található objektumok kontúrvonalait érzékeli és emeli ki a program.

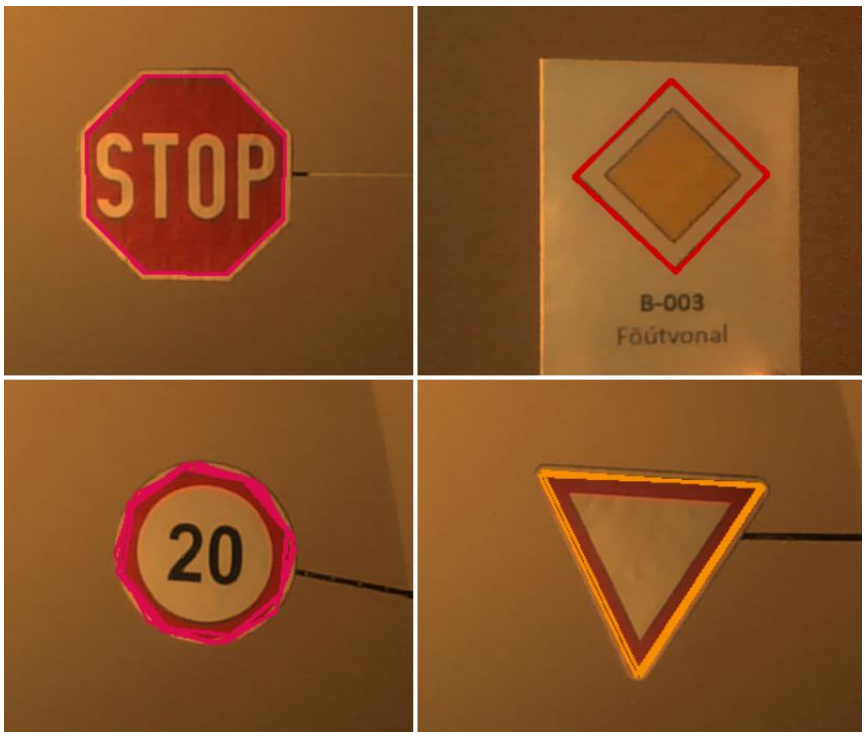
23. ábra
Az éldetektor működés közben.
Jobb oldalon a valós kép, bal oldalon ugyanaz az éldetektor lefuttatása után.

```
26  # Defining edge detector
27  def autoCanny(img):
28      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
29      img = cv2.GaussianBlur(img, (5, 5), 1)
30      canny = ut.auto_canny(img)
31      return canny
32
33
34  def manualCanny(img):
35      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
36      img = cv2.GaussianBlur(img, (5, 5), 1)
37      canny = cv2.Canny(img, 150, 200)
38      return canny
```

Ezen kódrészlet előkészíti az éldetektáláshoz a bemeneti képeket, majd végig is futtatja rajtuk a detektort. Ezen detektor paraméterezettsége rendkívül jelentős a megadott objektumok beazonosítása szempontjából. Az ezzel való kísérletezés céljából definiáltam kétféle Canny éldetektort is. A program végső verziójában az `autoCanny` függvény szerepel, ugyanis viszonylag jól képes – a fényviszonyok tekintetében – változtatni a detektor beállításain. A végleges működéshez elegendő lenne csupán az egyik függvény is, de úgy vélem, érdemes volt megtartani mindkettőt, ha esetleg később én vagy bárki más változtatna ennek a beállításain.

```
115     # Initializing lists for specifically shaped objects (triangle, rectangle,
          octagon)
116     tris = []
117     rects = []
118     octos = []
119
120     # Iterate through the contours list
121     for cnt in contours:
122         area = cv2.contourArea(cnt)
123
124         # Drop too small features
125         if area > 100:
126             perim = cv2.arcLength(cnt, True)
127             approx = cv2.approxPolyDP(cnt, 0.02*perim, True)
128             P = cv2.arcLength(approx, True)
129             convex = cv2.isContourConvex(approx)
130             corPnts = len(approx)
```

A `cv2.findContours` által beazonosított zárt kontúrvonalakból válogattam ki a számomra felhasználható alakzatokat. Ezen poligonok a háromszög, a négyszög és a nyolcszög formák, amikhez külön-külön listaváltozókat is rögzítettem. A 121. sortól néhány vizsgálatnak és műveletnek vetettem alá az alakzatokat, annak érdekében, hogy meggyőződhessek használhatóságukról. A 125. sor vizsgálja az adott poligon területét, amely ha meghalad egy bizonyos értéket, a program felhasználja azt, és megállapítja a sarokpontjainak számát.



24. ábra
A három felismert alakzat: nyolcszög (bal), négyszög (jobb felső), háromszög (jobb alsó).
A kör alakú táblákat a program a nyolcszög listában tárolja el.

```
132             # Drop too flat features then put the detected shapes in their
                  respective lists
133             if P * P / area < 30:
134                 if corPnts == 3:
135                     tris.append(approx)
136                 elif corPnts == 4 and convex:
137                     rects.append(approx)
138                 elif corPnts == 8 and convex:
139                     octos.append(approx)
140
141     # Applying dropSmaller function for polygon lists
142     dropSmaller(rects)
143     dropSmaller(tris)
144     dropSmaller(octos)
```

Ezután a program megvizsgálja, hogy az adott közlekedési tábla alakzatának körülírt ellipszise mennyire tér el a szabályos körtől. Ezzel azt vizsgálja, hogy az adott alakzat – mint egy valóságban is előforduló jelzőtábla – szabályos sokszög-e. Ha közel szabályos, akkor eltárolja a 116-118. sorokban inicializált listák közül a megfelelőbe, feltéve, hogy konvex poligonról van szó. Ha nem, akkor egyszerűen figyelmen kívül hagyja a poligont, és elkezdi vizsgálni a következőt. A 142-144. sorig bezárólag pedig lefuttattam ezeken a listákon az általam megírt `dropSmaller` eljárást, ami két részből tevődik össze.

```
52  # Test if contour c1 contains contour c2
53  def contains(c1, c2):
54      for p in c2:
55          if cv2.pointPolygonTest(c1, tuple(p[0]), False) < 0:
56              return False
57      return True
58
59
60  # Drop smaller polygons sharing the same center or being contained
61  def dropSmaller(polys):
62      i = 0
63      while i < len(polys):
64          j = i + 1
65          deli = False
66          while j < len(polys):
67              if contains(polys[i], polys[j]):
68                  del polys[j]
69              elif contains(polys[j], polys[i]):
70                  del polys[i]
71                  deli = True
72                  break
73              else:
74                  j += 1
75          if not deli:
76              i += 1
```

Ezeket a függvényeket a táblafelismerés optimalizálása végett implementáltam a programkódba. Amíg a `contains` csak vizsgálja, hogy van-e a bemeneti képen két olyan

poligon, ahol az egyik geometriailag magában foglalja a másikat, addig a `dropSmaller` már ki is törli azt a bizonyos másik, nagyobb poligonban foglalt, kisebb alakzatot.

```
146     # Initialize list for regions of interest
147     rois = []
148
149     # Iterate through the elements from the combined shape lists
150     for o in rects+tris+octos:
151
152         # Get the region of interests from the input pictures then predict
              their probabilities
153         (x, y, w, h) = expandRoi(cv2.boundingRect(o), 10)
154         if x < 0:
155             x = 0
156         if y < 0:
157             y = 0
158         rois.append((x, y, w, h))
159         crop = originalImage[y:y+h, x:x+w]
160         classIndex, prob = identify(crop)
```

Most már a program képes poligon detektálás alapján felismerni a kép azon részeit, amelyeken valószínűsíthetően jelzőtáblák szerepelnek, bár annak is megvan a lehetősége, hogy csak egy másik szabályos alakzatot érzékel. Következő lépésként ezeket a képrészeket kellett kiemelni és lefuttatni rajtuk a neurális hálózati modellt. A fennlátható `for` ciklusban a program megvizsgálja a poligonokat tároló listákat, majd minden egyes alakzatot tartalmazó képrészletet befoglaló dobozzal határol le, dimenzióit pedig elmenti a `rois` listába. Ezt követően a képből kivágja ugyanezen képrészletet, és előfeldolgozza azt a modell „jóslásához”.

```
79  # Expand the region of interest by the given pixels
80  def expandRoi(r, p):
81      (x, y, w, h) = r
82      return (x-p, y-p, w+2*p, h+2*p)
83
84
85  # Processing the images and getting the predictions
86  def identify(imgOri):
87      # Process image
88      img = np.asarray(imgOri)
89      img = cv2.resize(img, (32, 32))
90      img = preprocessing(img)
91      img = img.reshape(1, 32, 32, 1)
92
93      # Predict image
94      predictions = model.predict(img)
95      classIndex = model.predict_classes(img)
96      probabilityValue = np.amax(predictions)
97      return classIndex, probabilityValue
```

Az `expandRoi` az észlelt poligont befoglaló doboz pár pixeles kiterjesztése. Lényegében a függvény a kiemelt képrészletek felnagyítását végzi a táblák széleinek – modell általi – jobb elemezhetősége céljából. Tapasztalat szerint, ezen a kiegészítés segítségével pontosabb predikciós értékeket ad vissza a program. A kódsorozat egyik jelentős része a 86. sorban, az `identify` függvényben kerül definiálásra. Itt történik az adott képrészlet tényleges feldolgozása és transzformálása a gép számára feldolgozható formába, valamint a modell által felállított predikció értékének meghatározása. A függvény az adott képen szereplő tábla – modell által meghatározott – kategóriáját és a hozzá tartozó „jóslás" értékét adja végeredményül.

```
26  # Preprocess the input image for the model
27  def preprocessing(img):
28      img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
29      img = cv2.equalizeHist(img)
30      img = img / 255
31      return img
```

Itt található az `identify`-ban történő képi előfeldolgozást lebonyolító függvényhívás. Hasonlóan a tanító programhoz jelen esetben is először szürkére konvertálás folyamatán megy át a képi adat, ezután hisztogram kiegyenlítésen, majd végül normalizáláson esik át.

```
162         # If the probability value is bigger than x then print the image class
              and the prob value on the screen
163         if prob > .7:
164             cv2.putText(originalImage,
165                         str(classIndex) + " " + str(getClassName(classIndex)),
166                         (x, y), font, 0.7, (0, 0, 255), 2, cv2.LINE_AA)
167             cv2.putText(originalImage,
168                         str(round(prob*100, 2)) + "%", (x, y+100),
169                         font, 0.7, (0, 0, 255), 2, cv2.LINE_AA)
170
171             # Draw on screen a bounding box around the roi
172             for i in range(len(rois)):
173                 (x, y, w, h) = rois[i]
174                 cv2.rectangle(originalImage, (x, y), (x+w, y+h), (0, 255, 0),2)
```

Mivel a közúton a táblák beazonosítása egy igencsak lényeges feltétele a balesetmentes közlekedésnek, és mivel a program egyik célja, hogy ezt a folyamatot ugyanolyan jól – de inkább jobban – hajtsa végre, mint az ember, ezért a megkapott predikciós értékeket további vizsgálatnak vetjük alá. Csak és kizárólag akkor ír ki a

képernyőre bármit is, ha ez az érték nagyobb, mint 70%, azaz a rendszer minimum ennyi százalékban biztos benne, hogy az általa érzékelt tábla kategóriája megfelel a modell által „jósoltnak”. Ha eléri ezt a küszöbértéket, akkor a program megjeleníti a képernyőn az általa prediktált jelzőtábla fajtáját, a predikciós értéket és egy befoglaló dobozt a detektált alakzat köré.

```
41  # Loading and giving the correct class name to the corresponding value
42  def getClassName(classNo):
43      sourceFile = open("labels.csv")
44      labelNames = sourceFile.read().strip().split("\n")[1:]
45      sourceFile.close()
46      labelNames = [l.split(",")[1] for l in labelNames]
47      for i, l in enumerate(labelNames):
48          if classNo == i:
49              return l
```

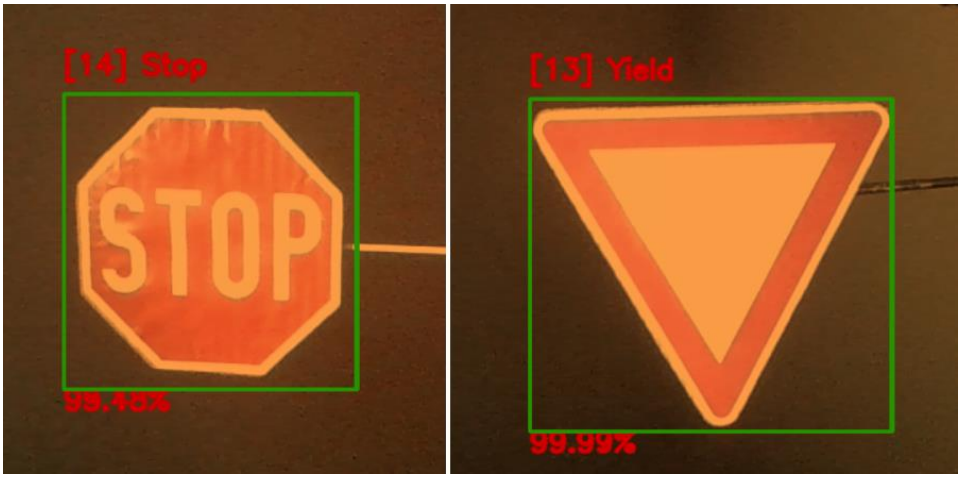
A `getClassName` nevű függvény funkciója az, hogy kicserélje mind a 43 meglévő táblafajta – gép általi azonosításra használt – megnevezéseit, az ember számára is értelmezhetővé téve ezzel.

```
181     cv2.imshow('Video', originalImage)
182
183     if cv2.waitKey(1) & 0xFF == ord('q'):
184         break
185
186 cap.release()
187 cv2.destroyAllWindows()
```

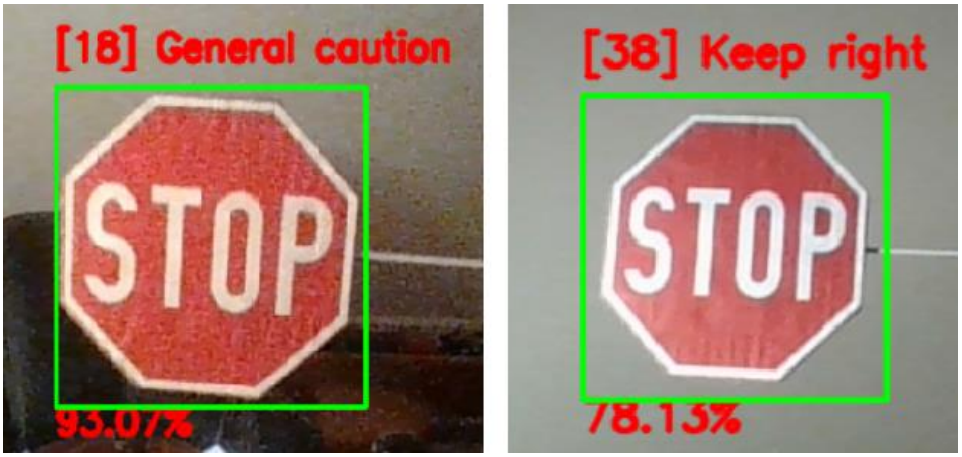
A kódsor végén megadtam, hogy megjelenjen egy kép, amin keresztül a felhasználó is láthatja az eredményeket, valamint definiáltam, hogy a program a ’q’ gomb megnyomására kilépjen a ciklusból, majd a futása végén fejezze be a kamera használatát, és zárja be az ablakot.

3.3. Eredmények

Többféle vizsgálatnak is alávetettem a rendszert, és alapvetően az mondható el, hogy – egy-két esettől eltérően – a célnak megfelelően működik. A gyors és egyszerű tesztelhetőség érdekében elsődlegesen a webkamerás ellenőrzést választottam, aminek lényege, hogy a kamerán keresztül prezentálok papírra nyomtatott jelzőtáblás képeket a programnak.



25. ábra
A „Stop” és az „Elsőbbségadás kötelező” táblákat általában jól azonosítja.

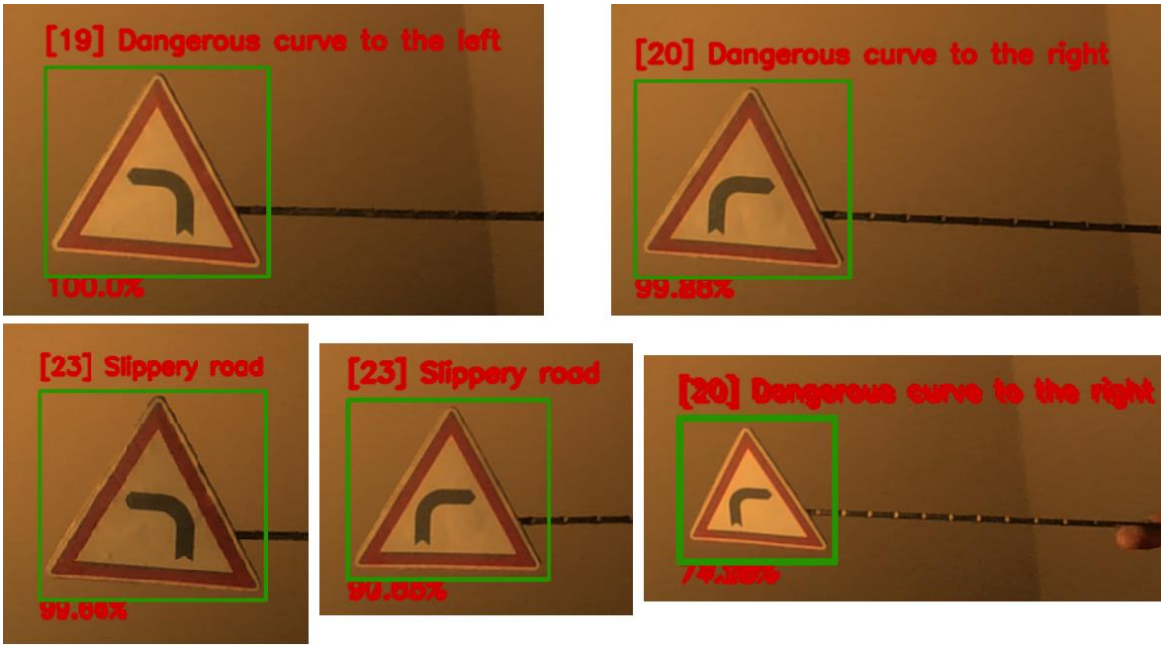


26. ábra
A „Stop” tábla esetében egy-egy képkocka erejéig előforduló anomáliák.

27. ábra
A „Sebességkorlátozás 20 km/h” táblákból nem volt sok az adatbázisban, bár a modell az esetek többségében jól felismeri, mégis fluktuál a „jóslás” értéke. (Első érték: 73,16%; Második érték: 80,37%; Harmadik érték: 97,72%)

Általában minél több adattal rendelkezik adott kategóriáról a neurális hálózat, annál nagyobb eséllyel fog a modell helyes predikciót visszaadni. Ez a szám a „Stop” tábla esetében 690 db, az „Elsőbbségadás kötelező” táblánál 1920 db, a „Sebességkorlátozás 20 km/h” táblánál pedig 180 db fájlt jelent. Ezért lehetséges, az, hogy a 25. ábrán a jobb oldali táblát magabiztosan azonosítja a program, a bal oldalit kicsivel rosszabbul, míg a 27. ábrán lévő tábla esetében látszik, hogy a predikciós érték megbízhatósága ingadozó.

28. ábra
A webkamerás ellenőrzés esetében ez az ingadozás, a „Körforgalom” tábla esetében nem fordult elő. A videós ellenőrzésnél azonban gyakran fordult elő, hogy nem, vagy „Főútvonal” táblaként azonosította a modell.

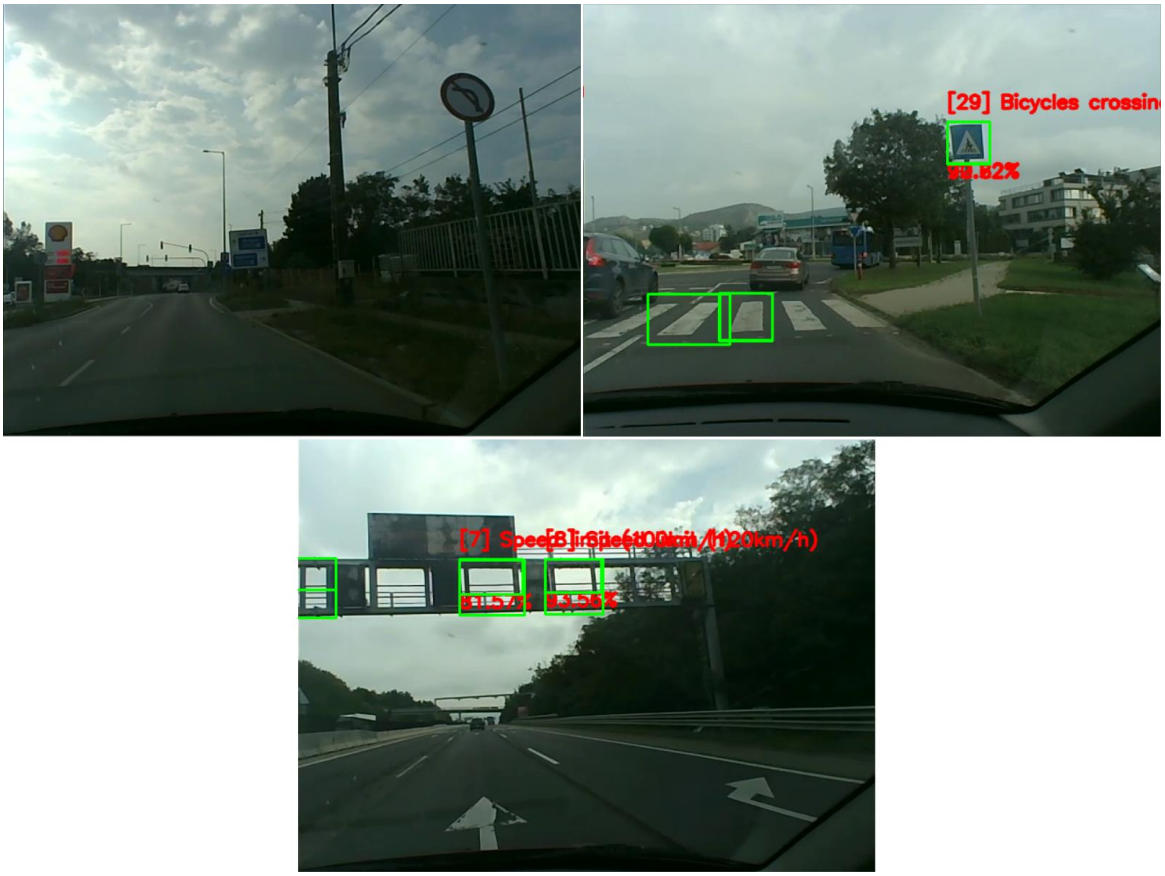


29. ábra
Az egymáshoz nagyon hasonló táblákat is képes jól megkülönböztetni, de mivel kevés kép van a két tábláról az adatbázisban (180 és 300 db), ezért előfordulhatnak olyan jelenségek, mint amelyek az alsó képeken is szerepelnek.

Másodlagosan egy autóútról előre rögzített videófájllal teszteltem a programot, mivel ez a megoldás sokkal életszerűbb az előzőnél, és nagyobb mértékben reprodukálható vele az éles helyzetben történő tesztelhetőség.

30. ábra
„Elsőbbségadás kötelező” és „Stop” tábla az utakon.

Azonban ebben az esetben nagyobb a hibázás lehetősége is. Több olyan objektum is előfordult az utakon, amelyeket a program érzékelt és a modell megpróbált azonosítani. De történt olyan is, hogy nem ismerte fel megfelelően a jelzőtáblát vagy egyáltalán nem is észlelte.



31. ábra
A rendszer tipikus hibái

Az ábrákon megjelenő predikciós értékek alapján kijelenthető, hogy a modell képes a betanított közlekedési táblák jelentős többségének helyes felismerésére és azonosítására, viszont mindezek ellenére közel sem tökéletes a rendszer. A program kiegészítésre szorul például a zajszűrés, illetve a hibás predikciók kezelése tekintetében. Előbbire megoldást jelenthet egy új, nem közlekedési táblákat gyűjtő kategória implementálása, utóbbira pedig az adatbázis bővítése további képekkel és közlekedési táblafajtákkal.

4. ÖSSZEGZÉS

Bár a dolgozat elején definiált célkitűzés tejesült, ennek ellenére megállapítható, hogy tartalma a mélytanulás és a neurális hálózatok felhasználhatóságának csak a töredékét érinti. A témának már most számos felhasználási területe létezik mind a térképészetben, mind a térinformatikában. Ez persze nem meglepő, hiszen a geoinformatika elsődleges feladatai közé a térbeli adatok kinyerése, tárolása, kezelése, elemzése és megjelenítése tartozik, a neurális hálózatoknak pedig éppen az a célja, hogy ilyen adatok alapján „tanuljon” és készítsen modelleket. A tudományterületen belüli konkrét feladatokra jó példa lehet a távérzékelésben használt légifotók kiértékelésének automatizálása, illetve – ennek a dolgozatnak tovább gondolt témájaként – a jelzőtáblák detektálása és pozíciójuk meghatározása.

A jövőre nézve kiváló irány lehet egy olyan architektúra kifejlesztése, amely rendelkezik egy GPS készülékkel a pozíció és egy orientációs szenzorral az irányszög meghatározása végett. Ezenkívül az egyszerű kamerát érdemes lecserélni egy sztereókamerára, amely segítségével a rendszer képes meghatározni a táblák készülékhez viszonyított relatív, a koordinátapár és irány figyelembevételével pedig az abszolút helyzetét is.

A témának ezen bővítése már biztos alapot képezhet egy nagyságrendekkel nagyobb projekt számára, amelynek része lehet egy – a detektált közlekedési táblák adatai alapján felépített – adatbázis létrehozása, valamint egy olyan webes vagy asztali (esetleg mobilos) applikáció fejlesztése, mely rendelkezik egy – ezen adatbázis rekordjait térbeli környezetbe helyező – térképes megjelenítő felülettel.

5. HIVATKOZÁSOK

5.1. Irodalomjegyzék

A webes hivatkozás utolsó elérésének dátuma: 2020.12.10.

ABADI, M. ET AL. (2015): *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* https://www.tensorflow.org/

CHOLLET, F. ET AL. (2015): *Keras.* https://github.com/keras-team/keras

DEEPLIZARD (2017): *Machine Learning & Deep Learning Fundamentals course*. https://deeplizard.com/learn/playlist/PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

GOODFELLOW, I. – BENGIO, Y. – COURVILLE, A. (2016): *Deep learning*. https://www.deeplearningbook.org/

HADAWALE, P. (2020): *Introduction to AI and Machine Learning (ML001-E).* https://medium.com/machine-learning-with-pratik/introduction-to-ai-and-machine-learning-ml001-e-c4a994c6c612

HASSAN, M. (2020): *Opencv Projects – Traffic Sign Classification.* https://www.murtazahassan.com/courses/opencv-projects/

HULLÁM G. – SALÁNKI Á. – KOCSIS I. (2016): *Gépi tanulási (Machine Learning) módszerek alkalmazása.* https://inf.mit.bme.hu/sites/default/files/materials/taxonomy/term/446/16/BD_ML_modszerek_HG_jav.pdf

ITSEEZ (2020): *Open Source Computer Vision Library (OpenCV).* https://github.com/opencv/opencv

LECUN, Y. – BENGIO, Y. – HINTON, G. (2015): *Deep learning.* https://doi.org/10.1038/nature14539

NIELSEN, M. A. (2015): *Neural Networks and Deep Learning.* http://neuralnetworksanddeeplearning.com

ORMÁNDI R. (2013): *Gépi tanulás a gyakorlatban – Bevezetés.* http://www.inf.u-szeged.hu/~ormandi/mlp/01-introduction.pdf

RAICEA, R. (2017): *Want to know how Deep Learning works? Here's a quick guide for everyone.* https://www.freecodecamp.org/news/want-to-know-how-deep-learning-works-heres-a-quick-guide-for-everyone-1aedeca88076/

ROSEBROCK, A. (2017): *Deep learning for Computer Vision with Python.* https://www.pyimagesearch.com/deep-learning-computer-vision-python-book/

ROSEBROCK, A. (2019): *A series of OpenCV convenience functions (imutils).* https://github.com/jrosebr1/imutils

ROSENBLATT, F. (1958): *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* https://doi.org/10.1037/h0042519

SANDERSON, G. (2017): *3Blue1Brown – Neural Networks course.* https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

SHAFKAT, I. (2018): *Intuitively Understanding Convolutions for Deep Learning.* https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

STALLKAMP, J.– SCHLIPSINGA, M. – SALMENA, J. – IGELB, C. (2012): *Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition.* https://doi.org/10.1016/j.neunet.2012.02.016

TRAPP H. (2018): *MI, deep learning, machine learning… avagy hogyan vezessük be a MI-t a cégünkbe.* https://szoval.hu/mi-deep-learning-machine-learning-avagy-hogyan-vezessuk-be-a-mi-t-a-cegunkbe/

WIKIPÉDIA (2020): *Mesterséges neurális hálózat*.
https://hu.wikipedia.org/wiki/Mesters%C3%A9ges_neur%C3%A1lis_h%C3%A1l%C3%B3zat

5.2. Ábrajegyzék

A webes hivatkozás utolsó elérésének dátuma: 2020.12.10.
A nem jelölt forrású ábrák saját szerkesztésűek.

7. ábra *Egy egyszerű, kevés rétegű, sekély neurális hálózat (bal)…*
https://www.xenonstack.com/images/blog/difference-between-neural-networks-deep-learning-neural-networks.png

8. ábra *Egy egyszerű neurális hálózat…*
https://miro.medium.com/max/640/1*8VSBCaqL2XeSCZQe_BAyVA.jpeg

9. ábra *A legismertebb aktivációs függvények.*
ROSEBROCK, A. (2017): *Deep learning for Computer Vision with Python.* pp. 125.

10. ábra *Gradiensereszkedés…*
https://sebastianraschka.com/images/faq/closed-form-vs-gd/ball.png

11. ábra *Alulillesztett-, jól generalizált- és túlillesztett modellek.*
https://miro.medium.com/max/700/0*CmDTGlQyibHUORQ0.png

12. ábra *A visszacsatolás az előterjesztéssel…*
CHAN PHOOI M'NG, J. – MEHRALIZADEH, M. (2016): *Forecasting East Asian Indices Futures via a Novel Hybrid of Wavelet-PCA Denoising and Artificial Neural Network Models.* https://doi.org/10.1371/journal.pone.0156338

13. ábra *Alapvető neurális hálózati fajták…*
https://miro.medium.com/max/1000/1*cuTSPlTq0a_327iTPJyD-Q.png
nyomán saját szerkesztés

14. ábra *A 3x3-as kernel (árnyékolt kockák) az előre megadott…*
https://miro.medium.com/max/700/1*Fw-ehcNBR9byHtho-Rxbtw.gif
nyomán saját szerkesztés

15. ábra *Egy konvolúciós rétegben történő aritmetikai művelet…*
https://miro.medium.com/max/535/1*Zx-ZMLKab7VOCQTxdZ1OAw.gif

16. ábra *Összetett formák, amiket a filterek már a modell 4. rétegében…*
https://deeplizard.com/images/CNN%20layer%204%20filters.jpg

19. ábra *A GTSRB adatbázisban található 43 közúti jelzőtábla kategória.*
WEN, L. – JO, K. (2017): *Traffic sign recognition and classification with modified residual networks.* https://doi.org/10.1109/SII.2017.8279326

6. KÖSZÖNETNYILVÁNÍTÁS

Tisztelettel köszönöm témavezetőmnek, dr. Gede Mátyásnak, hogy elfogadta felkérésemet, s szakmai tanácsaival, iránymutatásával folyamatosan segítette a munkámat.

Szeretnék továbbá köszönetet mondani a Térképtudományi és Geoinformatikai Intézet tanárainak, hogy segítették szakmai előmenetelemet, illetve családomnak és barátaimnak, akik mindvégig támogattak egyetemi tanulmányaim során.

7. MELLÉKLETEK

A diplomamunkához mellékelt .zip fájl szerkezete a következő:

diplomamunka/	– a .pdf formátumú dolgozatot és egyéb nyomtatványokat tartalmazó mappa
examples/	– a jelzőtáblafajtákról mintaképek
my_model/	– a betanított modell
myData/	– a jelzőtáblák képeivel teli adatbázis
labels.csv	– a jelzőtáblafajták azonosító számai és nevei
shape_based_select.py	– a tesztprogram
traning.py	– a tanító program

DIPLOMAMUNKA LEADÁSI
és
EREDETISÉG NYILATKOZAT

Alulírott **Dusek Bence** Neptun-kód: **FLWLBV**

az Eötvös Loránd Tudományegyetem Informatikai Karának, Térképtudományi és Geoinformatikai Intézetén

Közlekedési táblák automatikus térképezése

című diplomamunkámat a mai napon leadtam.

Témavezetőm neve: **dr. Gede Mátyás**

CD-t / DVD-t mellékelek *(aláhúzandó)*: igen nem

Büntetőjogi és fegyelmi felelősségem tudatában nyilatkozom, hogy jelen szakdolgozatom/diplomamunkám saját, önálló szellemi termékem; az abban hivatkozott szakirodalom felhasználása a szerzői jogok általános szabályainak megfelelően történt.
Tudomásul veszem, hogy szakdolgozat/diplomamunka esetén plágiumnak számít:

- szószerinti idézet közlése idézőjel és hivatkozás megjelölése nélkül;
- tartalmi idézet hivatkozás megjelölése nélkül;
- más publikált gondolatainak saját gondolatként való feltüntetése.

A témavezető által benyújtásra elfogadott szakdolgozat PDF formátumban való elektronikus publikálásához a tanszéki honlapon

HOZZÁJÁRULOK NEM JÁRULOK HOZZÁ

Budapest, 2020. december 15.

Dusek Bence
.....
hallgató aláírása