GÉPI LÁTÁS

GKNB_INTM038

KÖZLEKEDÉSI TÁBLÁK KLASSZIFIKÁLÁSA

TÓTH SÁNDOR BALÁZS

IBW7AN

GYŐR, 2020/2021 I. FÉLÉV

Tartalom

1 Bevezetés

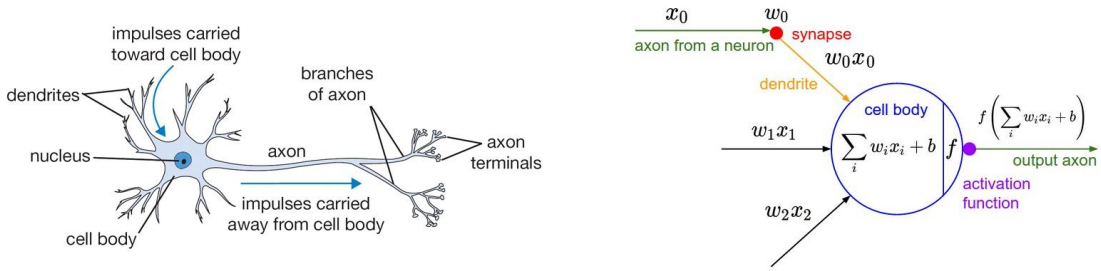
Az utóbbi időben a mesterséges intelligencia és a gépi tanulás tárházába tartozó eszközök rohamos fejlődése a gépi látás területére óriási hatással volt. Közülük is kiemelkedően fontos szerep jutott a neurális hálózatoknak, melyek rendkívül széles körben kerülnek felhasználásra. Segítségükkel lehetőségünk van objektumok felismerésére, detektálására. Valószínűsíthetően a neuronhálók szerepe az önvezető és vezetéstámogató rendszerrel ellátott járművek esetén is jelentős lesz, sőt már napjainkban is felhasználásra kerülnek ilyen megoldások.
Az önvezető autókat csoportosíthatjuk aszerint, hogy milyen mértékben képesek önálló közlekedésre, illetve döntéshozásra. Az viszont könnyen belátható, hogy csaknem minden fokozat esetén szükség van a környezet bizonyos mértékű feltérképezésére, elemzésére. Ehhez számos eszköz áll a járművek rendelkezésére, például radar, lidar, ultrahang, illetve természetesen a kamera. Ez utóbbi által szolgáltatott kép számít jelenleg a neuronhálók elsődleges inputjának. A kamera segítségével a neuronhálók képesek lehetnek számos, a közlekedés szempontjából fontos objektum detektálására, klasszifikálására. Ezek lehetnek a közlekedés résztvevői (gyalogosok, biciklisek, autósok), illetve a KRESZ betartása és a biztonságos közlekedés elősegítése szempontjából fontos környezeti elemek, például útburkolati jelek, jelzőlámpák, és ami jelen feladat szempontjából is érdekes, a közlekedési táblák. Utóbbiak detektálása és pontos klasszifikálása elengedhetetlen az önvezető autó helyes és biztonságos működéséhez.
Munkám során egy közlekedési táblák klasszifikálását végző neurális háló került betanításra és tesztelésre. Emellett két olyan módszer is kipróbálásra került, melyek a tesztkészlet képein különböző változtatásokat eszközölnek annak érdekében, hogy a hálót megtévesszék és téves klasszifikációt eredményezzenek.

2 A neurális hálók, a klasszifikáció és az ellenséges példák bemutatása

A mesterséges neurális háló megalkotásakor az emberi agyműködés szolgált mintául. Nem állt rendelkezésünkre ugyanis a megjelenésük előtt olyan algoritmus, mely hatékonyan lenne képes alakzatokat felismerni, így az agyműködés lemodellezése tűnt a legkézenfekvőbb megoldásnak. A mesterséges neurális háló az agyhoz hasonlóan neuronokból áll, melyek bizonyos módon kapcsolatban állnak egymással. A kapcsolatok kialakítása különböző topológiák szerint valósulhat meg. A neuronhálók igen nagy teljesítmény elérésére képesek. Ennek oka, hogy olyan számítások elvégzésére képesek, melyek viszonylag kis számú párhuzamos nemlineáris lépésből állnak. A továbbiakban a Stanford CS231n: Convolutional Neural Networks for Visual Recognition [2] című kurzus felhasználásával mutatom be a neurális hálók felépítését és működésének alapjait.

2.1 A neurális hálók felépítése és működése

A neuronhálók legkisebb egységei a neuronok. Az 1. ábra a biológiai neuront és annak matematikai modelljét mutatja.



1. ábra: a biológiai neuron és annak matematikai modellje [2]

A biológiában egy neuron az agy alapvető számítási egysége. A neuronhoz a dendriteken keresztül jutnak el a különböző jelek, melyekből az axon segítségével pedig előállítja a kimeneti jeleket. Az axon végül szétágazik és szinapszisok révén kapcsolódik más neuronok dendritjeihez. A számítási modell esetében az axonon végigfutó $x0$ jel a $w0$ szinapszisnak megfelelő intenzitással fog a másik neuronnal kapcsolatba lépni. A különböző w jelek tanulhatók és a jel hatásának erősségét befolyásolják. A különböző erősségű jelek a neuron belsejében összegzésre kerülnek. Ha ez az összeg meghalad egy küszöbértéket, a neuron tüzelni fog.
A matematikai modell esetében a neuron tüzelési rátáját az aktivációs függvény határozza meg. Néhány aktivációs függvény:

- Sigmoid: a bemenetül kapott valós számot a [0,-1] intervallumba szűkíti
- ReLU (Rectified Linear Unit): a függvény küszöbértéke 0
- Tahn: a bemenetre érkező valós számot a [-1,1] intervallumba szűkíti

A neurális hálók szerkezete megadható neuronok gráfokba szervezésével. A gráfban az egyes neuronok kimenetei más neuronoknak bemenetül szolgálnak, mely alapján a neuronok rétegekbe szerveződnek. Az általános felépítés szerint három fő rétegtípust különböztethetünk meg:

- bemeneti réteg: a háló input adatai, például képek
- rejtett rétegek
- kimeneti réteg: a réteg célja az eredmény közlése. Ez klasszifikációs feladat során jelentheti azt, hogy a kép egy kutyát vagy macskát ábrázol-e. Az eredmény lehet egy vektor, vagy valós szám is.

A veszteségfüggvény segítségével meghatározható, hogy a modell eredménye és a valós osztályok között mekkora a differencia. Pár általánosan használt veszteségfüggvény:

- Multiclass Support Vector Machine Loss
- Cross Entropy Loss

A veszteségfüggvény segítségével számszerűen megadható a hálózat minősége minden adott W súlyokból álló halmaz esetén. A cél egy olyan W halmaz megtalálása, mely a veszteség minimalizálását segíti elő. Kezdetben a súlyok véletlenszerű értékekkel rendelkeznek, de megadható nekik más kezdőérték is. Ahhoz, hogy minél hatékonyabban minimalizáljuk a súlyértékeket tartalmazó vektort, a legmeredekebb csökkenés irányába kell lépést tennünk. Ezt a veszteségfüggvény **gradiense** biztosítja számunkra, melyre tekinthetünk lejtőként, azonban a függvény nem egy konkrét értéket, hanem egy vektort kap bemenetül.
A háló teljesítményének kiértékeléséhez az egyik legáltalánosabb megközelítés az, hogy az adathalmazt felosztjuk egy tanító és egy teszthalmazra. A modell tanulása során a tanító halmazt fogja felhasználni, majd miután a modell tanulása befejeződött, bementül adjuk neki a számára eddig ismeretlen teszthalmaz képeit és megállapíthatjuk a modell pontosságát. Fontos, hogy a tanító halmaz minden kategória vagy osztály képeiből megfelelő mennyiségűvel rendelkezzen. Az, hogy a képeket milyen arányban választjuk szét a modelltől függhet, de érdemes jelentősen nagyobb tanító halmazt megalkotni.

2.2 A klasszifikáció

A képfeldolgozás egyik legjelentősebb területét jelenti a képklasszifikáció, mely során célunk az, hogy az adott képhez előre megadott címkék/osztályok közül egyet hozzárendeljünk. Például, ha az adathalmaz olyan képekből áll, melyek 1-től 10-ig ábrázolhatnak egy számot, akkor az a feladatunk, hogy mindegyik képhez hozzárendeljük, hogy melyik szám szerepel rajta. Bizonyos esetekben nem feltétlenül egy osztályt, hanem az osztályoknak való megfelelés valószínűsége kerül megadásra. Bár a feladat emberi nézőpontból nem jelent nagy kihívást, a számítógépes képfeldolgozás során számos tényezőt kell figyelembe vennünk az ilyen feladatok során: nézőpont variancia, skála variancia, megvilágítás, deformáció, elfedés stb.

2.3 Ellenséges példák

A neurális hálók megtévesztésének legjelentősebb területe a háló „ellenséges példák” (adversarial exapmples) segítségével történő manipulálása. A terület 2014-től örvend igazán nagy figyelemnek, amikor Christian Szegedy [1] tanulmányában kifejtette, hogy a neurális hálót be lehet csapni a képek oly módon történő manipulálásával, hogy az szabad szemmel szinte észrevehetetlen legyen. A modell ennek révén akár nagy mértékben is képes tévesen klasszifikálni a képeken levő különböző objektumokat.

Munkám során két, ellenséges példák létrehozását végző módszert használtam fel:

- Fast Gradient Sign Method [3]
- Projected Gradient Descent [4]

3 Közlekedési táblák klasszifikálása

3.1 A felhasznált adatkészlet

Munkámhoz a German Traffic Sign Recognition Benchmark (GTSRB) [5] által kínált, szabadon hozzáférhető adatkészlet képeit használtam fel. Az adatkészlet összesen több, mint 50000 képet tartalmaz 43 különböző osztályból, tehát akár 43 különböző közlekedési tábla klasszifikálása is megvalósítható segítségével.

2. ábra: példák a GTSRB adatkészlet képeiből [5]

Munkámhoz a megoldandó feladatot leegyszerűsítettem és 4 darab osztály képei kerültek felhasználásra a háló tanításához, illetve az eredmény kiértékeléséhez, ezzel meggyorsítva a háló tanítását és növelve pontosságát.
Bár a képeken németországi közlekedési táblák szerepelnek, ez nem jelentett különösebb problémát, ugyanis az általam vizsgált 4 osztály esetén ezek megjelenése megegyezik a hazánkban található táblákéval.
Az adott táblákat tartalmazó osztályok számok segítségével kerültek reprezentálásra, így például az általam vizsgált képek az alábbi osztályokhoz tartoznak:

- 12 - Főútvonalat jelző tábla
- 13 – Elsőbbségadás kötelező tábla
- 14 – STOP tábla
- 15 – Mindkét irányból behajtani tilos tábla

A vizsgálandó osztályok kiválasztásakor fontos szempont volt a táblák által betöltött szerep, ugyanis mind a négy tábla általános, viszonylag sűrűn előforduló és be nem tartása súlyos következménnyel járhat.
Az adatkészlet úgy épül fel, hogy a 43 különböző mappa az osztályt megjelölő számmal került elnevezésre, ezzel egyértelműen azonosítható, hogy az abban a mappában található képek mely osztályhoz tartoznak.

3.2 Az adatok feldolgozása

Az 1. kódrészlet a későbbi tanítókészletet alkotó adatok beolvasását és előfeldolgozását mutatja.

1. kódrészlet: A GTSRB adatkészlet képeinek beolvasása és feldolgozása

```
data=[]
labels=[]

height = 30
width = 30
channels = 3
n_inputs = height * width*channels

for i in range(12, 16) :
    path="c:/Users/tsbalazs/Gepi_latas/GTSRB/Final_Training/{}/".format(i)
    print(path)
    Class=os.listdir(path)
    for a in Class:
        try:
            image=cv2.imread(path+a)
            image_from_array = Image.fromarray(image, 'RGB')
            size_image = image_from_array.resize((height, width))
            data.append(np.array(size_image))
            labels.append(i-12)
        except AttributeError:
            print(" ")

Cells=np.array(data)
labels=np.array(labels)
```

Ahogy látható, a data listába kerültek a feldolgozott képek, a labels osztályba pedig a hozzájuk tartozó osztály száma. A képeket tartalmazó mappák elrési útjának megadását követően a tartalmazott képeket az opencv imread függvénye segítségével olvastam be. A képeket egységesen 30 pixel szélességűvé és magasságúvá méreteztem át, majd numpy tömbkét a data listához fűztem.

Ezt követően az osztályszámok a labels listába kerültek, melyek értékéből 12-t kivonva elértem, hogy indexelésük 0-tól kezdődjön.
Miután ez megtörtént, az adatok sorrendje randomizálásra került. Az adatok ily módon történő véletlenszerű összekeverésére azért van szükség, mivel így hatékonyabb tanulás és pontosabb klasszifikáció érhető el. Emellett az adatok későbbi tanító és validációs készletre történő felosztásakor is biztosítja, hogy mindkét készlet kellően heterogén képkészletet tartalmazzon.
Az eddig beolvasott adatokat ezt követően osztottam fel tanítási és validációs készletre. A felosztás során a képek 80 %-a került a tanító, 20 %-a a validációs készlet képei közé. Utóbbira azért van szükség, hogy a tanítási folyamat nyomon követése során láthassuk a modell pontosságának alakulását, illetve képet kapjunk a modell pontosságáról. Az osztálycímkéket tartalmazó tömbök adatait one-hot-encode vektortokká alakítottam, ugyanis a modell a későbbiekben ilyen formában várja az erre vonatkozó adatokat. A one-hot-encode vektorok olyan vektorok, melyek pontosan egy darab 1-est tartalmaznak, míg a többi értéken 0 szerepel. Az 1-es azon az indexen található, amely index a képhez tartozó osztály száma. Miután a korábbiakban a képek beolvasásánál az eredeti számok módostultak, a különböző osztályok az alábbi címkékkel rendelkeznek:

- 0 - Főútvonalat jelző tábla
- 1 – Elsőbbségadás kötelező tábla
- 2 – STOP tábla
- 3 – Mindkét irányból behajtani tilos tábla

Ez alapján tehát egy STOP táblát ábrázoló kép osztályát egy one-hot encode vektor az alábbi módon ír le: [0, 0, 1, 0].
Az adatok beolvasását követően a tanítási készlet képei a 3. ábrán látható módon néznek ki.

3. ábra: példák a tanulási készlet képeiből

3.3 A felhasznált neurális háló

A feladathoz felhasznált neurális háló [6] felépítését a 2. kódrészlet mutatja.

2. kódrészlet: A neuronháló felépítése

```
1. input_shape = X_train.shape[1:] # (30,30,3)
2.
3. model = Sequential()
4. model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), activation='relu',
   input_shape=input_shape))
5. model.add(MaxPool2D(pool_size=(2, 2), strides=(2, 2)))
6. model.add(Conv2D(64, (5, 5), activation='relu'))
7. model.add(MaxPool2D(pool_size=(2, 2)))
8. model.add(Flatten())
9. model.add(Dense(1000, activation='relu'))
10.model.add(Dense(4, activation='softmax'))
11.model.compile(loss='categorical_crossentropy',optimizer='adam',
12.metrics=['accuracy'])
13.
14.epochs = 2
15.history = model.fit(X_train, y_train, batch_size=32, epochs=epochs,
   validation_data=(X_val, y_val))
```

A felhasznált háló egy konvolúciós neuronháló, mely két konvolúciós réteggel is rendelkezik. Ahogy az input_shape változó is utal rá, a bemenetet egy 30*30*3-as képekből álló képhalmaz fogja jelenteni. A szorzat első két eleme a kép méretét, a harmadik pedig a három színcsatornát jelenti. Ahogy látható, a két konvolúciós réteg különböző méretű konvolúciós kernellel, szűrővel rendelkezik, azonban mindkettő esetén a ReLu aktivációs függvény került felhasználásra. A konvolúciós rétegeket MaxPool rétegek követik, melyek célja a képi jellemzők kiemelése pl.: élek, sarokpontok. A konvolúciós rétegeket követően az input a Flatten() metódus segítségével átalakításra kerül, melynek révén az a későbbi lazán csatolt rétegeknek átadható lesz. Az első lazán csatolt réteg még a ReLu aktivációs függvényt használja, az utolsó azonban, mely az eredmény közlésére szolgál már a softmax függvényt. A felhasznált veszteségfüggvény a Cross Entropy Loss.
A tanítás során a modellnek összesen két alkalommal kerülnek átadásra az input adatok, mivel 2 epoch került megadásra. Átadásra kerülnek még a tanító készlet képei és címkéi, valamint a validációs készlet képei. A képek 32 képből álló batchek formájában kerülnek átadásra a hálónak. A 4536 képből álló tanító halmaz esetén tehát nagyjából 142 batch jön létre.

4 Az eredmények kiértékelése

4.1 A neurális háló teljesítménye

A tesztelés során a neuronháló teljesítménye a tesztkészleten mutatott pontosság és a tanítási idő alapján került értékelésre. Ahhoz, hogy kellően reprezentatív eredmény születhessen, összesen 10 alkalommal került a háló tanításra. A tanítás végbemenetele és annak időtartamának lemérése után kerültek a tesztkészlet képei beolvasásra.
A GTSRB adatkészlet tesztkészlete minden osztály képeiből tárol képeket. A képekhez tartozó osztályok megállapítása a GT-final_test.csv nevű fájl segítségével lehetséges, mely tartalmazza a képfájl nevét és az ahhoz tartozó valós osztály számát. Jelen esetben természetesen csak a vizsgált 4 osztály képei kerültek felhasználásra a tesztkészletből. A futási eredmények az 1. táblázatban láthatók, melyben a pontosság %-ban, a futási idő pedig másodpercben került megadásra.

1. Táblázat: a neurális háló futási eredményei

Futás száma	Pontosság [%]	Tanítási idő [s]
1	98,41	13,28
2	97,99	12,58
3	98,89	12,21
4	98,84	13,1
5	98,47	13,53
6	98,41	12,39
7	98,52	12,44
8	98,31	12,27
9	98,52	12,3
10	98,52	12,41
Átlagos érték	98,49	12,65
Legkisebb érték	97,99	12,21
Legnagyobb érték	98,89	13,53
Terjedelem	0,90	1,32

Ahogy a táblázat mutatja, a háló átlagosan 98,49 %-os pontosság elérésére volt képes a tesztkészlet képein. A legpontatlanabb futása is 97,99 %-os, mely nem sokkal marad el az átlagos teljesítménytől. A 10 futás alatt elért legjobb pontossági érték 98,89 %-os volt. Látható tehát, hogy a háló pontossága viszonylag állandó, kis mértékben ingadozik.
A háló tanítása átlagosan 12,65 s alatt ment végbe. A futási idő tekintetében sem figyelhető meg túlzottan nagy ingadozás, mivel a leglassabb futás 12,21 s, a leggyorsabb pedig 13,53 s volt.
Összességében elmondható, hogy a háló viszonylag rövid tanítási idő mellett is jó pontosság elérésére képes 4 osztály közlekedési tábláinak klasszifikálása esetén. Mi történik azonban, ha a tesztkészlet képein különböző változtatásokat eszközölünk annak érdekében, hogy a hálózat teljesítményét befolyásoljuk?

4.2 A háló teljesítménye a tesztkészlet képeinek manipulálása után

A tesztkészlet képeinek módosításához a CleverHans [7] python könyvtárat használtam fel, mely különböző, ellenséges példák generálását végző módszerek gyűjteménye. Segítségével lehetőségünk van az általunk kiválasztott módszerek felhasználásával képek módosítására. Munkám során 2 ilyen módszert próbáltam ki, melyek a Fast Gradient Sign Method (FGSM) és a Projected Gradient Descent (PGD). A kiértékeléshez a háló módosított képeken való pontossága és a képek generálásának ideje került rögzítésre.

4.2.1 Az FGSM futási eredményei

Az FGSM módszer futási eredményei a 2. táblázatban láthatók.

2. Táblázat: az FGSM futási eredményei

Futás száma	Pontosság [%]	Generálási idő [s]
1	2,06	1,69
2	6,88	1,48
3	9,1	1,5
4	2,8	1,53
5	9,05	1,5
6	1,53	1,48
7	6,14	1,5
8	4,44	1,5
9	2,43	1,47
10	2,22	1,47
Átlagos érték	4,67	1,51
Legkisebb érték	1,53	1,47
Legnagyobb érték	9,10	1,69
Terjedelem	7,57	0,22

Ahogy a táblázat mutatja, a pontosság drasztikus mértékben csökkent manipulált képeken. Míg eredetileg a háló 98,49 %-os pontosság elérésére volt képes, ez az FGSM által manipulált képeken csupán 4,67 %-os.
A pontossági értékek viszonylag ingadozóak az FGSM esetén. Egy esetben képes volt 1,53 %-ra lerontania a háló pontosságát, míg a támadó szempontjából a legkevésbé hatékony futáskor a pontosság 9,1 %-os volt. Ez a 7,57 %-os eltérés viszonylag nagynak tekinthető.
A manipulált képek előállítási idejeit figyelembe véve elmondható, hogy egy gyors lefutású módszerről van szó. Átlagosan 1,51 s alatt ment végbe a képek előállítása. A pontossággal ellentétben a futási idők nem túl ingadozóak, csupán 0,22 s volt a különbség a leggyorsabb és leglassabb futások közt.

4.2.2 A PGD futási eredményei

A PGD módszer futási eredményeit a 3. táblázat tartalmazza.

3. Táblázat: a PGD futási eredményei

Futás száma	Pontosság [%]	Generálási idő [s]
1	1,48	39,11
2	1,9	40,09
3	1,06	39,12
4	1,01	43,09
5	1,43	42,39
6	1,22	39,29
7	1,38	39,53
8	1,64	39,08
9	1,27	39,11
10	1,22	39,63
Átlagos érték	1,36	40,04
Legkisebb érték	1,01	39,08
Legnagyobb érték	1,90	43,09
Terjedelem	0,89	4,01

Ahogy a 3. táblázat mutatja, a Projeced Gradient Desent még az FGSM-nél is nagyobb pontosságbeli csökkenés elérésére volt képes. A módszer által generált képeken a neuronháló csupán 1,36 %-os átlagos pontosság elérésére volt képes. A pontosság ingadozásának mértéke itt jóval kisebb, mint az FGSM esetén. Csupán 0,89 % a különbség a legpontosabb és legkevésbé pontos futások közt.
A képek generálási ideje jelentősen nagyobb a másik módszerhez képest. Átlagosan 40,04 másodpercre volt szükség a példák generálásához. A differencia a leggyorsabb és leglassabb futások közt 4,01 s volt.

5 Felhasználói dokumentáció

A feladat kidolgozását és a tesztelést az alábbi specifikációjú számítógépen végeztem el:

- operációs rendszer: Windows 10 Home 64bit
- processzor: Intel Core i5-7200U 4 magos CPU
- memória mérete: 8GB

A háló tanítása a CPU-n történt, GPU nem került ehhez felhasználásra.
A programot conda rendszer felhasználásával készítettem el és próbáltam ki Windows 10-es környezetben. A szükséges főbb rendszerek és verzióik az alábbiak:

- Conda 4.9.0 (opcionális, más környezet is használható)
- Python 3.7.1
- Tensorflow 2.2.0
- Keras 2.4.3
- Cleverhans 3.0.1 (az ellenséges példák előállításához)

A programkód a github oldalon megtalálható .py kiterjesztésű fájlként és .ipynb kiterjesztésű notebook fájlként is. Előbbit Visual Studio Code, utóbbit pedig Jupyter Notebook környezetben készítettem el és próbáltam ki.
A program számos pyhon könyvtárat felhasznál, ezek hiányában telepítésük szükséges lehet. Az alábbiakban látható a szükséges libraryk listája:

- numpy
- pandas
- matplotlib
- opencv
- PIL
- os
- tensorflow
- keras
- sklearn
- cleverhans
- absl

Amennyiben az adatkészlet képeinek letöltése megtörtént fontos, hogy a programban az adatok beolvasása során pontos elérési utat adjunk meg attól függően, hol található az adatkészlet. Erre a tanító készlet és a tesztkészlet képei esetén is érdemes figyelni.

6 Irodalomjegyzék

[1] C. Szegedy, W. Zaremba, I. Sutskever és J. Bruna, „Intriguing properties of neural networks,” 2014.

[2] A. Karpathy, „Cs231n convolutional neural networks for visual recognition,” *Neural networks,* 2016. (2020.11.26)

[3] I. J. Goodfellow, J. Shlens és C. Szegedy, „Explaining and Harnessing Adversarial Examples,” 2015.

[4] A. Madry, A. Makelov, L. Schmidt és D. Tsipras, „Towards Deep Learning Models Resistant to Adversarial Attacks,” 2019.

[5] „German Traffic Sign Recognition Benchmark,”. Available: https://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset. (2020.10.15)

[6] „Keras tutorial – build a convolutional neural network in 11 lines,”. Available: https://adventuresinmachinelearning.com/keras-tutorial-cnn-11-lines/. (2020.11.2)

[7] „CleverHans,” . Available: https://github.com/cleverhans-lab/cleverhans. (2020.11.28)