Széchenyi István Egyetem
Gépészmérnöki, Informatikai és
Villamosmérnöki Kar

Mesterséges intelligencia beadandó (GKNB_INTM002)

Szebik Levente
(A0TDT5)

2023/2024/2

SZÉCHENYI EGYETEM
UNIVERSITY OF GYŐR
GÉPÉSZMÉRNÖKI, INFORMATIKAI ÉS VILLAMOSMÉRNÖKI KAR

Mesterséges intelligencia beadandó (GKNB_INTM002)

Sudoku Solver

Szebik Levente
(A0TDT5)

2023/2024/2

Tartalomjegyzék

Tartalom

Bevezetés

A probléma

A „probléma” megnevezés ennek a programnak az esetében talán nem a legtalálóbb, hiszen nem egy hétköznapi problémára nyújt megoldást, hanem egy sudoku rejtvény kitöltésével járó problémát old meg. A sudoku játék célja, hogy a kitöltő egy 9x9 mezőre osztott négyzetbe úgy írja be egytől kilencig a számokat, hogy minden sorban és minden oszlopban egy szám csak egyszer szerepeljen, viszont egyszer kötelezően szerepelnie kell. Valamint a 9 kis négyzetre osztott táblákat pedig szintén úgy kell kitölteni, hogy minden szám csak egyszer szerepeljen benne. A kitöltés nehézségét az határozza meg, hogy hány darab szám van előre megadva és ezek melyik mezőkön helyezkednek el.

A fejlesztés menete

A fejlesztés megkezdése előtt kisebb részfeladatokra bontottam a projektet úgynevezett „milestone” -okra. Ilyenek voltak például a rejtvény négyzetének felismerése, a számfelismerés, a sudoku megoldása. Mivel ez volt az első mesterséges intelligenciával foglalkozó projektem így rengeteg cikk elolvasásával, videó megnézésével, és stackoverflow olvasással telt a fejlesztés, mivel csak a funkciót tudtam, hogy mit akarok megvalósítani. Azt, hogy azt hogyan kell annak mindig utána kellett néznem.

A program

Felhasznált könyvtárak

A fejlesztés során több Python könyvtár is felhasználásra került, amelyek a program különböző funkcióinak megvalósításában játszanak szerepet. Ezeknek a könyvtáraknak a listáját a *requirements.txt* fájl tartalmazza, amelyből egyszerűen telepíthetőek egy parancs lefuttatásával.

NumPy (Numerical Python) – 1.21.5

A NumPy egy nyílt forráskódú projekt, amely lehetővé teszi a numerikus számításokat a Python segítségével. 2005-ben fejlesztették ki a Numeric és Numarray korai munkáira alapozva. [1]

A programban ennek a könyvtárnak a képfeldolgozásban van szerepe, hiszen ez a könyvtár lehetővé teszi a képek tömb formátumban való kezelését. Ezért a képekkel könnyen végezhetőek el műveletek, mint például: azok átméretezése vagy szürkeárnyalatúvá alakításuk.

Matplotlib - 3.5.1

A matplitlib egy, a Python programozási nyelvhez, és annak numerikus matematikai kiterjesztéséhez, a Numpy-hoz írt ábrázoló könyvtár. [2]

Ez a könyvtár főképpen a fejlesztés során volt hasznos. A különbözően transzformált képek megjelenítését, a felismert számok megjelenítését, ezáltal a teljes debugolás folyamatát elősegítette.

OpenCV (Open Source Computer Vision Libary) – 4.5.4.48

Az OpenCV egy nyílt forráskódú gépi látás és gépi tanulás megvalósítását elősegítő könyvtár. Azért fejlesztették ki, hogy közös infrastruktúrát biztosítson a gép látás alkalmazásaihoz, és felgyorsítsa a gépi érzékelés felhasználását a kereskedelmi termékekben. [3]

A program ennek a segítségével dolgozza fel a bemeneti képeket. Ennek a segítségével találja meg a sudoku szegélyeit, a számokat, a számok négyzeteit.

Tensorflow – 2.7.0

A Tensorflow egy ingyenes és nyílt forráskódú szoftverkönyvtár a gépi tanuláshoz és a mesterséges intelligenciához. Különösen jól alkalmazható a mély neurális hálózatok esetében. [4]

A programon belül a számjegyeket felismerő modell létrehozásához és annak a fájlba való betöltéséhez használatos.

Tqdm – 4.62.3

A tqdm egy könyvtár, amely az iterációkhoz készít előrehaladást jelző sávot. A konzolos kimeneteken a tanítás folyamata során használja a program az előrehaladás jelzéséhez.

Konfiguráció betöltése

A program konfigurációját a *config.txt* fájl tartalmazza, innen a configparser modul segítségével tudjuk kiolvasni az adatokat. Első körben példányosításra kerül majd *read()* metódusának használatával megadjuk a konfigurációs fájl elérési útvonalát. Ezt követően a *get()* funkcióval kiolvassuk a megadott konfigurációs paramétert. Ez azért hasznos mert így nem a programkódban kell átírni a paramétereket azoknak a változása esetén, hanem egy külön fájlból is megtehető.

Sudoku négyzet felismerése

Kamera kép

A program elindítását követően megjelenik egy ablak, amely a kamera képét jeleníti meg. A fejlesztés során a könnyebben való kezelhetőség miatt, egy androidos telefonra telepített IP Camera alkalmazás volt használva, hogy egy kézben fogott telefon kameraképét lehessen használni.

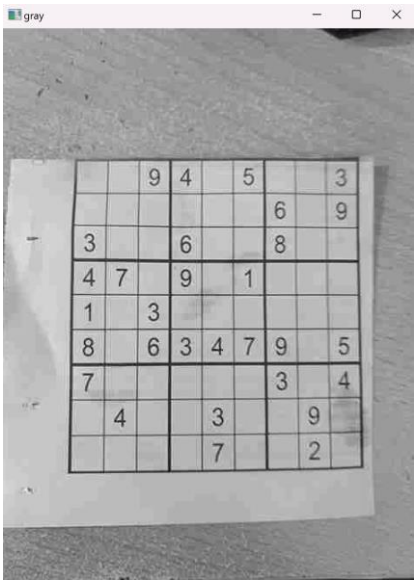
A *cv2.VideoCapture()* függvénnyel inicializáljuk a kamerát majd nyitjuk meg azt. Ezt követően a *read()* függvénnyel beolvassuk az aktuális képkockát. A képkockák beolvasása egészen addig tart amíg a program nem találja meg a sudoku rejtvényt vagy a felhasználó nem nyomja meg a *q* billentyűt. Amennyiben a második eset következik be, azaz a felhasználó lenyomja a *q* billentyűt akkor abban az esetben a program összes megnyitott és futó ablaka bezáródik és a futása befejeződik. Azonban, ha sikeresen felismerte a sudoku külső négyzetrács vonalait akkor megkezdődik a képfeldolgozás folyamata.

Framek transzformációja

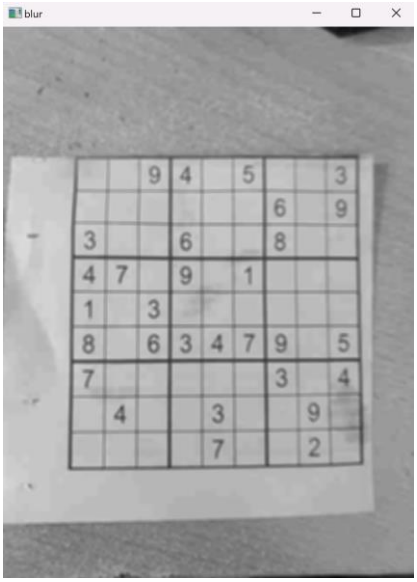
A program a kamera képéről beérkező frameket első körben a *cv2.cvtColor()* függvénnyel a színes képet átalakítja szürkeárnyalatossá. Erre azért van szükség mivel a szürkeárnyalatos képeket könnyebb feldolgozni, mivel ilyen módon a kép csak egy darab színcsatornát tartalmaz, amivel csökken a feldolgozáshoz szükséges számításigény és javul a pontosság. (2. ábra)

Ez után a szürkeárnyalatossá transzformált képet elhomályosítja a *cv2.GaussianBlur()* függvény használatával. Erre azért van szükség, mivel így a feldolgozandó képnek csökken a zajszintje és az elhanyagolható részek elsimítódnak így a későbbi számjegy felismerésbe nem zavarnak be. (3. ábra)

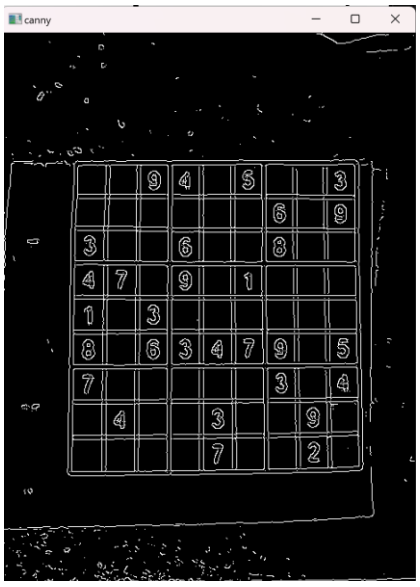
Ezt követően már csak egy darab módosítást eszközöl a képen azonban ez is legalább olyan fontos, mint az előzőek annak érdekében, hogy a modell a későbbiekben feltudja ismerni a számjegyeket. Ez a módosítás a képen elvégzett éldetektálás, amelyet a *cv2.Canny()* végez el. Ez azért fontos, hogy a képen látható objektumok (esetünkben a számok és azok határoló négyzetei) határait kiemelje. (1. ábra)



2. ábra Szürkeárnyalatos kép



3. ábra Elhomályosított kép



1. ábra Éldetektált kép

Külső körvonalak felismerése

Magának a sudoku rejtvények a képről történő felismerését a *getContours()* függvény végzi. Ez paraméterként kettő változót vár. Elsőként, az előző pontban utolsó módosításként elvégzett éldetektált képet várja. Másodikként pedig az eredeti kamerakép módosítatlan fram-ét. Elsőként a paraméterként kapott képen a *cv2.findContours()* függvénnyel megkeresi a rajta lévő külső kontúrokat. Ennek a függvények három paramétert adunk át. Először a vizsgált képet, másodikként a *cv2.RETR_EXTERNAL* paraméterrel meghatározzuk, hogy a külső kontúrokat keresse. Harmadikként pedig a *cv2.CHAIN_APPROX_NONE* paramétert ezzel jelezve, hogy nem csak a négyzet négy sarokpontjára vagyunk kíváncsiak, hanem a teljes négyzet külső vonalaira.

Ezek után egy *for* ciklussal végig iterálunk a megtalált kontúrokon. A *cv2.contourArea()* függvénnyel kiszámolja a kontúrvonalak területét majd egy feltétellel megvizsgálja, hogy az adott kontúr területe nagyobb-e 60000 pixelnél. Erre azért van szükség, hogy biztosan a teljes sudoku rajta legyen az így kapott képen és mondjuk ne történjen meg az, hogy tévesen az egyik kis négyzetrácsot vizsgálja csak a program.

Ezt követően a *cv2.arcLength()* függvénnyel meghatározzuk a kontúr körvonalának a hosszát. Fontos megjegyezni, hogy amíg ez a függvény a körvonal hosszának az értékét határozza meg, addig az előzőleg használt *cv2.contourArea()* a kontúr által bezárt területet adja vissza. A *cv2.approxPolyDP()* függvény segítségével meghatározza a négyzet sarokpontjait. Végül a függvény visszatér egy módosított szürkeárnyalatos képpel, amin már csak a sudoku rejtvény látszódik és semmi más.

Számok felismerése

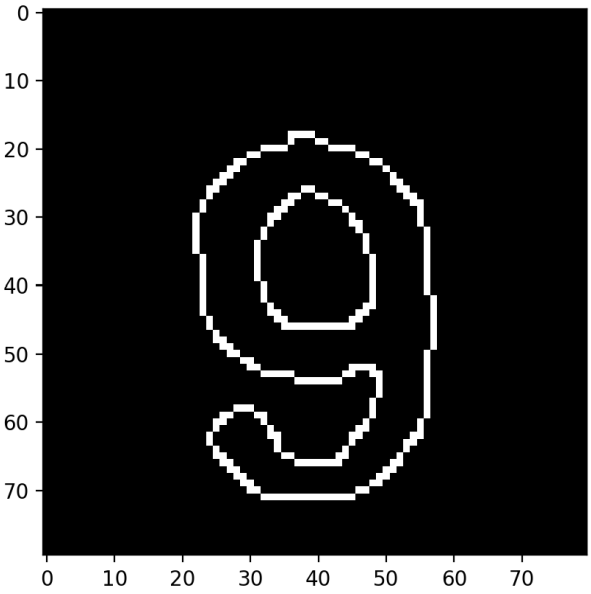
Az előzőleg megkapott képet, amely már csak a sudoku-t tartalmazza megkapja a *classify()* függvény paraméterként. Ez a függvény két egymásba ágyazott *for* ciklussal végigmegy a sudoku mind a 81 celláján. Maga a sudoku rejtvény felosztása tekinthető úgy, mint egy 9x9-es kétdimenziós mátrix. Mivel a *getContours()* által visszaadott kép 900x900-as és a függvény ezt a képet kapja paraméterül így könnyen kiszámítható az aktuális kis cella bal felső sarkának a koordinátai. Ez úgy tehető meg, hogy az aktuális ciklusváltozók megszorzásra kerülnek 100-zal és még hozzáadódik egy előre meghatározott érték, amellyel finomhangolható az, hogy a cella külső vonalai ne kerüljenek rá a képre, amelyről majd fel kell ismerni a számjegyet. Miután az eredeti képről kivágásra került a cella a nagy négyzet felismeréséhez hasonlóan elvégezzük a szükséges előkészítő transzformációkat. Mivel a kép már alapból szürkeárnyalatos és elhomályosított ezért már csak a

cv2.Canny() függvénnyel kell a képen található éleket detektálni.

A képen ezután a korábban bemutatott módszerrel megkeresésre kerülnek a kontúrok majd ezek egy *for* ciklussal végigiterál rajtuk a program. Ezután is meghatározásra kerül az éppen vizsgált kontúr által bezárt terület. Erre azért van szükség mert hiába kerül a kép széle levágásra, előfordul, hogy egy két pixelen belelóg a határoló négyzet kontúrja. Amennyiben nagyobb a vizsgált terület 7,5-nél akkor feltételezhető, hogy a szám került megtalálásra. (4. ábra)

A *cv2.boundingRect()* függvény alkalmazásával meghatározzuk a szám kontúrvonalait magába foglaló téglalap bal felső sarkának koordinátáit, szélességét, magasságát. A következőkben ezzel a kisebb téglalappal dolgozik tovább a program. Mivel a modell tanítása során 100x100-as képeket használtam ezért ezt a téglalapot is át kell méretezni egy ekkora négyzetté. Ezután a *img_to_array()* függvény segítségével a kép egy NumPy tömbbé kerül átalakításra. Továbbá a kép még több átalakításon keresztül megy, hogy megfeleljen a neurális hálózat bemeneti elvárásainak. Ilyen például az adatok lebegőpontos számmá való konvertálása majd 255.0-val történő leosztása. Ez azért szükséges mert a szürkeárnyalatos képek pixeleinek eredeti értéke a [0, 255] intervallumban helyezkedik el, míg ezzel az osztással a [0, 1] tartományba lehet besorolni az adatot. Ez a neurális hálózat tanulását és eredményességét segíti elő.

Végül a betöltött és betanított modell a *predict()* funkcióval „megtippeli”, hogy milyen számjegy szerepelhet a képen. Ezt úgy valósítja meg a program, hogy az összes számjegyre (1-9) megad egy valószínűséget azzal kapcsolatban, hogy mennyire valószínű az adott szám képen való megjelenése. Ezután ezek közül a valószínűségek közül kiválasztásra kerül a legnagyobb, azaz a legvalószínűbb. A konzolra kiíratásra kerül a felismert szám és a hozzá kapcsolódó valószínűség. A kapott számokat egy tömbbe tároljuk el és a cellák felismerését követően a számokat tartalmazó tömb kerül visszaadásra. Fontos megjegyezni, hogy a üres cellákat, tehát amikben nem szerepel számjegy a program 0-ként tárolja.



4. ábra A felismeréshez előkészített kép

A sudoku megoldása

A *classify()* függvény által visszakapott tömböt amely a sudoku számait tartalmazza egy 9x9-es mátrixszá alakítjukm, majd az így kapott mátrixot NumPy tömbbé alakítjuk a könnyebb kezelhetőség érdekében és készít róla egy másolatot amit majd a későbbiekben fog használni.

A rejtvény megoldását a *solve()* függvény végzi amely egy *backtracking* algoritmust használ és paraméterként megkapja az előzőleg átalakított NumPy tömböt. Ezt követően egyből meghívásra kerül a *find_empty()* függvény amely paraméterül szintén megkapja ezt a tömböt. Ez megkeresi az első üres cellát és visszaadja annak a sor- és oszlopszámát. Amennyiben *None* értékkel tér vissza azt jelenti, hogy a sudokuban nincs üres cella tehát teljesen kitöltött. Ebben az esetben a függvény visszatér *True* értékkel. Azonban, ha van üres cella, akkor ennek az üres cellának a helyére 1-9-ig megpróbál behelyettesíteni számokat. Ezt követően meghívásra kerül az *is_valid()* függvény amely le ellenőrzi, hogy a beírt szám helyes-e. Amennyiben igen rekurzívan meghívja önmagát a függvény és megpróbálja kitölteni a következő üres cellát. Abban az esetben, ha az újrahívást követően a függvény *False* értékkel tér vissza akkor a cella értékét visszaállítja nullára és megpróbálkozik egy másik számmal. A függvény két értékkel térhet vissza *True* vagy *False*, ez a két érték azt jelzi, hogy a sudoku megoldható-e vagy nem. A *False* értékkel akkor tér vissza, ha már kipróbálta az összes számot és az összes lehetőséget.

Az *is_valid()* függvény három paramétert vár és ezek segítségével dönti el , hogy a beírt számjegy helyes-e. Elsőként a NumPy tömböt amely a sudoku aktuális kitöltöttségét tartalmazza, másodikként a cellába aktuálisan beírni kívánt számot, harmadikként pedig a cella koordinátáit. A függvény három különböző dolgot ellenőriz, amelyek a sudoku szabályai. Elsőként azt, hogy az aktuális sorban a számjegy csak egyszer szerepeljen. Ezért ellenőrzi, hogy a sorban szerepel-e már a szám és ha szerepel akkor az aktuális cellában-e, mivel értelemszerűen az megengedett. Ezt követi az oszlop ellenőrzése, ez ugyanazon az elven történik, mint a sor ellenőrzése. Végül pedig a 3x3-as kis celláról is megbizonyosodik a függvény mivel abban is csak egyszer szerepelhet a szám. Amennyiben mindegyik ellenőrzésen megfelel a szám akkor *True* értékkel tér vissza, minden más esetben *False*-al.

Végül amennyiben a sudoku megoldható akkor a *save_sudoku()* függvény kirajzolja a sudoku rácsait és beleírja a megoldást majd ez egy képként elmenti *solved.png* (5. ábra) néven a program gyökérkönyvtárába.

6	8	9	4	2	5	1	7	3
2	1	4	7	8	3	6	5	9
3	5	7	6	1	9	8	4	2
4	7	5	9	6	1	2	3	8
1	9	3	2	5	8	4	6	7
8	2	6	3	4	7	9	1	5
7	6	1	5	9	2	3	8	4
5	4	2	8	3	6	7	9	1
9	3	8	1	7	4	5	2	6

5. ábra Megoldott sudoku

Modell

Az adatok előkészítése

A modell tanításához szükséges képeknek az elérési útvonalát a *config.txt* tartalmazza. A képeket elsőkörben a *Tensorflow* által biztosított *MNIST* adatbázissal akartam megadni a modellnek azonban ezek kézzel írott számokat ábrázolnak és amikor a felismerésre került a sor nem voltak túl pontosak a modell tippjei, ezért úgy döntöttem, hogy saját képeket készítek. Ezt úgy oldottam meg, hogy megadtam a programnak, a beolvasott sudoku a számokkal előre kitöltött mátrixát és a cellák képeit mappákba rendezve elmentettem a *detected_imgs* mappába.

A képek a *data_augmentation()* függvényen belül kerülnek beolvasásra. Mivel mindegyik számjegyhez csak egy darab képem volt így a hatékony tanítás eléréséhez adatnövelést alkalmaztam. Ez egy olyan technika, amellyel a meglévő képeket módosítva újabb és újabb képeket generál, amelyekkel megnő a tanítás hatékonysága. A függvény betölti a mappákba rendezett képeket majd párosítja azt az előre definiált és egy konstans tömbben tárolt számjegyekkel. Ez a tömb az összes lehetséges számjegyet tartalmazza, amely a cellákon belül megjelenhet 0-9 (a nulla az üres cellát jelöli) növekvő sorrendben. A képek módosításokon esnek át a korábban részletezett módokon, ilyen például a szürkeárnyalatossá alakítás vagy az éldetektáció. Ezeket a módosított képeket hozzáadja az adott számjeggyel párosítva a *training_data* tömbhöz majd a végén visszaadja ezt az adatokkal feltöltött tömböt. Ezt követően a minta adatokat véletlenszerű sorrendbe rendezzük annak érdekében, hogy a modellnél elkerüljük a túltanulás jelenségét.

Modell létrehozása és tanítása

Ezt követően a modell létrehozásra kerül. A *Tensorflow Sequential API*-t használjuk a neurális háló létrehozására. Ez több rétegből áll, az első a laposító réteg (Flatten), ez a réteg átalakítja a bemeneti képeket egydimenziós tömbökké. Ezt a réteget kettő darab sűrű réteg (Dense) amelyek egyenként 128 neuronnal és *ReLU* (Rectified Linear Units)-al rendelkeznek. Végül pedig egy aktivációs réteg zárja a sort, amely 10 neuront tartalmaz (10 db felismerhető osztályunk van) és egy softmax aktivációs függvényt. Ezt a függvényt használja a modell a valószínűségi osztályzásra. Ezek után a modell összeállításra kerül. *Adam* optimalizációval, egy veszteségfüggvénnyel és metrikákkal.

Miután a modell létre lett hozva elkezdődhet a tanítása. Ezt a *fit()* függvény végzi amely paraméterként megkapja az *epoch-ok* számát. Ennek a paraméternek az értéke határozza meg, hogy

a tanítási folyamat hányszor fog végigmenni a rendelkezésre álló adatokon. A tanítás folyamán a konzolban megjelenik a tanítás aktuális státusza, valamint a becsült hátralévő idő. A tanítás befejeztével a modell mentésre kerül egy .h5 formátumú fájlban és a főprogramba betölthető.

Github

A projekt GitHub repository-a: https://github.com/Levente011/Sodoku_Solver/tree/main

Felhasznált irodalmak

- https://stackoverflow.com/questions/57636399/how-to-detect-sudoku-grid-board-in-opencv
- https://www.geeksforgeeks.org/detect-an-object-with-opencv-python/
- https://medium.com/analytics-vidhya/sudoku-backtracking-algorithm-and-visualization-75adec8e860c
- https://www.inf.u-szeged.hu/~rfarkas/ML20/deep_learning.html
- https://www.tensorflow.org/api_docs/python/tf/keras/Model

Források

[1] „NumPy - About Us”. Elérés: 2024. május 3. [Online]. Elérhető: https://numpy.org/about/

[2] „Matplotlib documentation — Matplotlib 3.8.4 documentation”. Elérés: 2024. május 3. [Online]. Elérhető: https://matplotlib.org/stable/index.html

[3] „About”, OpenCV. Elérés: 2024. május 3. [Online]. Elérhető: https://opencv.org/about/

[4] „Introduction to TensorFlow”. Elérés: 2024. május 3. [Online]. Elérhető: https://www.tensorflow.org/learn