



**SZÉCHENYI
EGYETEM**
UNIVERSITY OF GYŐR



**INFORMATIKA
TANSZÉK**
DEPARTMENT OF INFORMATICS

Széchenyi István Egyetem
Gépészmérnöki, Informatikai és Villamosmérnöki Kar
Informatika Tanszék

Képfeldolgozás (GKNB_INTM152)

Rendszámtábla felismerés

Novák Gergő László
FIJXQQ

<https://github.com/gera0719/LicencePlateRecognition>

Tartalomjegyzék

Tartalom

Tartalomjegyzék	2
A projekt ismertetése	3
Program indítása	3
Fejlesztői dokumentáció	4
Adathalmaz	4
Használt képfeldolgozási algoritmusok	4
Tanítási eredmények összehasonlítása	4
A rendelkezésre álló scriptek	5
Mappaszerkezet	6
Az elkészített scriptek működése	6
detect_plate.py	6
detect_ocr.py	7
detect.py	7
train.py	8
fine_tune.py	8
preprocess_pipeline.py	9
preprocess_train.py	10
pascal_to_yolo.py	11
separate_dataset.py	12
Tesztfuttatás	12
Irodalomjegyzék	15

A projekt ismertetése

A projekt lényege, ahogy a nevéből is adódik, rendszámtáblák felismerése, majd a felismert rendszámtáblákon található felirat leolvasása.

A projekt Python nyelven íródott, a rendszámtábla detektálásra egy feltanított *YOLOv8* modellt, a szövegfelismerésre pedig *Tesseract OCR*-t használva.

Az alábbi könyvtárak kerültek felhasználásra:

- *os*
- *pytesseract*
- *pillow*
- *ultralytics*
- *pathlib*
- *xml*
- *cv2*
- *numpy*
- *typing*
- *shutil*
- *random*

A program egy bemeneti kép alapján automatikusan megkeresi és azonosítja a képen látható jármű rendszámtábláját. Ehhez egy korábban betanított *YOLO* alapú objektumdetektáló modellt használ, amely képes a rendszámtábla pozícióját pontosan meghatározni. A detektálás után a képből kivágott táblarészletet előfeldolgozza (például élesítés, zajszűrés), majd karakterfelismerést hajt végre rajta egy *OCR*-algoritmussal. Az eredményként a terminálra kiírja az értelmezett rendszámtábla karaktereit, és opcionálisan menti is a felismerési eredményeket.

Program indítása

Feltételezve, hogy a felhasználó *WSL Ubuntu* disztribúciót használ.

1. Repository klónozása

```
git clone  
https://github.com/gera0719/LicencePlateRecognition
```

2. Függőségek telepítése

```
pip install requirements.txt
```

3. Teszt képek (*data/test/testing.zip*) kicsomagolása

4. Felismerő program indítása

```
python src/detect.py
```

Fejlesztői dokumentáció

Adathalmaz

A projekt elkészítéséhez a *Car Licence Plate Detection*[1] nevű *Kaggle* oldalon található adathalmazt használtam, mely 433 képet tartalmazott autókról. Az adathalmazban szerepeltek európai-, és nem európai szabványú rendszámtáblák. Az adathalmazhoz tartozó címkék *Pascal VOC* formátumban voltak.

Használt képfeldolgozási algoritmusok

A projektben több lépcsős képfeldolgozási eljárást alkalmaztam annak érdekében, hogy a rendszámtáblák detektálását és karakterfelismerését előkészítsem és javítsam.

A tanítóképek előfeldolgozása során első lépésként *CLAHE*[2] módszert alkalmaztam a képek világosság-komponensére, amely hatékonyan növeli a kontrasztot még egyenetlen fényviszonyok mellett is. Ezt követte egy bilaterális szűrés[3], amely a képek éleit megőrzi, a zaj eltávolítása mellett. Végül egy *unsharp masking* technikával kiemelttem a képen található éleket, hogy a rendszámtábla kontúrjai még jobban elkülönüljenek a háttértől.

A detektált rendszámtáblák optikai karakterfelismerésére történő előkészítéséhez egy külön képfeldolgozási pipeline készült. Ennek első lépéseként a képeket 2-szeresére nagyítottam, így javítva a karakterek részletességét. Ezután *Hough-transzformációval* történő egyenesdetektálás segítségével megállapítottam a kép dőlésszögét, és ezt követően kiegyenesítettem a képet. Ez különösen fontos olyan esetekben, ahol a jármű oldalról látható és a rendszámtábla ferdén jelenik meg, mivel az optikai karakterfelismerés nagyon nehezen ismeri fel a nem teljesen vízszintesen elhelyezkedő szöveget.

A képek ezután zajszűrésen estek át, majd átalakításra kerültek szürkeárnyalatossá. Az *Otsu-féle küszöbölést*[4] használtam *binarizálásra*, amely automatikusan választott küszöbértékkel fehér-fekete képet hoz létre, így javítva a karakterek kontrasztját a háttérhez képest. A pipeline záró lépéseként a legnagyobb összefüggő fehér régió automatikusan kivágásra kerül, így az OCR már csak a releváns tartalommal dolgozik.

Ezek a képfeldolgozási eljárások jelentős mértékben javítottak a felismerés pontosságán.

Tanítási eredmények összehasonlítása

Az alábbiakban látható két tanítási eredmény összehasonlítása. Az első a legelső tanítás az adathalmaz feldolgozatlan képein, a második az első modell finomhangolása az adathalmaz feldolgozott képeivel.

Metrika	Eredeti modell	Finomhangolt modell	Megjegyzés
Confusion matrix	licence_plate: 0.82	licence_plate 0.96	Jelentős javulás a felismerésben, csökkent a tévesztés
F1	max F1 \approx 0.82 @ conf 0.267	max F1 \approx 0.93 @ conf 0.405	Erősebb F1 érték, jobb konfidencia
Precision-Confidence	Precision \approx 1.00 @ conf 0.882	Precision \approx 1.00 @ conf 0.831	Precizitás hasonló, de alacsonyabb conf mellett is megmarad
Precision - Recall	mAP@0.5 = 0.819	mAP@0.5 = 0.943	Jobb általános detekciós teljesítmény
Recall confidence	Recall \approx 0.96 @ conf 0	Recall \approx 0.97 @ conf 0	Kis növekedés, nem ez a fő javulási terület
Loss	Magasabb és instabilabb vesztség értékek, lassabb konvergencia	Stabilabb és alacsonyabb veszteség, jobb generalizáció	Javult a stabilitás és a validációs teljesítmény

1. táblázat - A tanítások eredményeinek összehasonlítása

Összességében elmondható, hogy a képfeldolgozás végrehajtása után történt finomhangolás nagyban javította a rendszámtábla felismerő modell értékeit.[5]

A rendelkezésre álló scriptek

- *detect_plate.py*
- *detect_ocr.py*
- *detect.py*
- *train.py*
- *fine_tune.py*

- *preprocess_pipeline.py*
- *preprocess_train.py*
- *pascal_to_yolo.py*
- *separate_dataset.py*

Mappaszerkezet

```

├── README.md
├── data
│   ├── processed
│   ├── raw
│   ├── result
│   ├── split
│   └── test
├── models
│   ├── stepzeroftune
│   └── stepzerotest
├── requirements.txt
├── src
│   ├── detect.py
│   ├── detect_ocr.py
│   ├── detect_plate.py
│   ├── fine_tune.py
│   ├── pascal_to_yolo.py
│   ├── preprocess_pipeline.py
│   ├── preprocess_train.py
│   ├── separate_dataset.py
│   └── train.py
└── yolov8n.pt

```

Az elkészített scriptek működése

detect_plate.py

Ez a modul végzi a rendszám táblák detektálását a feltanított YOLO modell segítségével. A scriptben a modell-, a bemeneti mappa-, és a kimeneti (projekt) mappa elérési útvonala statikusan van megadva.

Lépések:

1. Feltanított modell betöltése
2. Predikció végzése, az *ultralytics* modulban megtalálható alapértelmezett függvénnyel a megadott bemeneti könyvtárban található képekre, 0.5 minimum konfidenciaszint értékkel

3. Felcímkézett képek mentése a megadott projekt mappába, egy új (eredmény) mappába
4. Kivágott rendszámablák mentése egy az eredmény mappában található másik (kivágott) mappába
5. Az új mappa elérési útvonalának visszaadása visszatérési értéként

Azért van szükség erre, mivel a predikciós függvény minden futtatáskor egy új mappát hoz létre a projekt mappában, és így megkaphatjuk a tényleges kimeneti útvonalat.

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban lévő bemeneti mappában vannak képek.

Amennyiben ez teljesül,

```
python src/detect_plate.py
```

detect_ocr.py

Ez a modul végzi a kivágott rendszámábla képeken az optikai karakterfelismerést.

Lépések:

1. Beolvassa a paraméterként megadott mappában található képfájlokat
2. Képfeldolgozást végez minden képen a *preprocess_pipeline* nevű modul használatával
3. *Pytesseract* segítségével felismeri a képen lévő nagybetűket, illetve számokat
13-as értékű *oldaltagolási módszert* használ, ami azt mondja meg a modellnek, hogy a kép egyetlen nyers szövegsor tartalmaz
4. Kiírja a felismerés eredményét a terminálra minden képfájlhoz

Bár van megadva statikusan egy alapértelmezett bemeneti könyvtár, az a tesztelésre szolgál.

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban lévő bemeneti mappában vannak képek.

Amennyiben ez teljesül,

```
python src/detect_ocr.py
```

detect.py

Ez a script végzi el a teljes detektálási folyamatot.

Lépések:

1. *detect_plates* eljárás meghívása

2. *detect_plates* eljárás visszatérési értékének mentése a *plates_path* változóba
3. *perform_ocr* függvény meghívása a kivágott képeket tartalmazó mappára

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban lévő modulok bemeneti mappáiban vannak képek.

Amennyiben ez teljesül,

```
python src/detect.py
```

train.py

Ez a script tanít fel egy *YOLOv8* neurális hálót rendszám-tábla felismerésére.

Az alapértelmezett tanító scriptnek utat mutató *dataset.yaml* fájl és a kimeneti könyvtár statikusan van megadva.

Lépések:

1. Alap *yolov8n* előtanított modell betöltése
2. Tanítás megkezdése az alábbi paraméterekkel:
 - a. *epoch*: 60 körön keresztül tanít
 - b. *imgsz*: 640x640-es méretre alakítja a képeket a tanításhoz
 - c. *batch*: 16 képpel tanul egyszerre
3. A betanított modell kiértékelése validációs adatokon

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban megadott *dataset.yaml* fájl létezik.

Amennyiben ez teljesül,

```
python src/train.py
```

fine_tune.py

Ez a script a már korábban betanított modell finomhangolására szolgál.

Az alapértelmezett tanító scriptnek utat mutató *dataset.yaml* fájl és a kimeneti könyvtár itt is statikusan van megadva. A *dataset.yaml* fájlban olyan könyvtárhoz tartozó elérési út szerepel, melyben a korábban használt képek, a *preprocess_train* modul segítségével feldolgozott változatai találhatók.

Lépések:

1. Meglévő modell betöltése
2. Tanítás megkezdése az alábbi paraméterekkel:

- a. *epoch*: 30 körön keresztül tanít
 - b. *imgsz*: 640x640-es méretűre alakítja a képeket
 - c. *batch*: 16 képen tanul egyszerre
 - d. *lr0*: 0.001-es tanulási ráta, ami megfelelő a finomhangoláshoz
3. A betanított modell kiértékelése validációs adatokon

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban megadott *dataset.yaml* fájl létezik.

Amennyiben ez teljesül,

```
python src/fine_tune.py
```

preprocess_pipeline.py

A modul egy kapott kép feldolgozását végzi különböző módszerekkel. A cél, hogy olyan adathalmazt lehessen nyújtani ezzel a modell finomhangolásához, melynek eredményeképp egy pontosabb modellt kapunk.

Lépések:

1. Kép beolvasása, betöltése
2. Kép kétszerezésre növelése, a részletek megőrzésével
3. A kép forgatása úgy, hogy a rendszámtáblán található felirat egyenes legyen, a rendszámtábla két oldalsó egyenesének detektálása és *Hough-transzformáció* segítségével
4. Kép tisztítása *bilaterális szűrővel*
5. Az RGB kép *szürkeárnyalatossá* való konvertálása
6. A kép fekete-fehérré alakítása *Otsu-módszerrel*
7. A képen található legnagyobb fehér régió (amely nagy valószínűséggel a rendszámtábla) kivágása
8. A feldolgozott kép visszaadása eredményül

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban megadott kép elérési útja helyes.

Amennyiben ez teljesül,

```
python src/preprocess_pipeline.py
```



1. ábra - Feldolgozás előtti felismert rendszámtábla



2. ábra - Feldolgozás utáni felismert rendszámtábla

preprocess_train.py

Ez a script egy képfeldolgozási folyamatot valósít meg, melyet a modell finomhangolásához használt képeken kerül alkalmazásra. A feldolgozás mellett a megfelelő fájlok kezelését, mentését is végzi.

Lépések:

1. Képfeldolgozás
 - a. *CLAHE* kontrasztjavítás alkalmazása a képek részleteinek kiemelésére
 - b. *Bilaterális szűrés* alkalmazása a képek zajszűrésére, az élek megőrzésével
 - c. Élek kiemelése *Gauss elmosás* segítségével
2. A kapott bemeneti mappában történő képek feldolgozása a fenti módszerrel, majd a kapott képek mentése
3. A képekhez tartozó címkék másolása

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban megadott mappa tartalmaz-e képeket.

Amennyiben ez teljesül,

```
python src/preprocess_train.py
```



3. ábra - Tanítás előtti feldolgozás nélküli kép egy járműről



4. ábra - Tanítás előtti feldolgozott kép egy járműről

pascal_to_yolo.py

A használt adathalmaz *Pascal VOC* annotációt használt, ami a modellhez nem megfelelő. Ez a script átalakítja megfelelő (YOLO) formátumba a címkéket.

Lépések:

1. Bemeneti és kimeneti mappák beállítása
2. Címkék feldolgozása az *XML* fájlokon való iterálással:
 - a. Képméret és keretméret kiolvasása
 - b. Átkonvertálás *YOLO* formátumba
3. Címkék mentése a megfelelő helyre, a megfelelő néven

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban megadott mappa szerkezete megfelelő, tartalmaz-e *Pascal VOC* formátumú címkéket.

Amennyiben ez teljesül,

```
python src/pascal_to_yolo.py
```

separate_dataset.py

Ez a script egy meglévő adathalmazt oszt fel, a megadott arányoknak megfelelően, tanító, teszt és validálási halmazokra.

Lépések:

1. Bemeneti és kimeneti elérési utak beállítása
2. Arányok definiálása
3. Fájlok véletlenszerű összekeverése, majd szétosztása
4. A képek és a címkék megfelelő mappákba másolása

Futtatása:

Először is bizonyosodjunk meg, hogy a kódban megadott mappa tartalmaz-e képeket.

Amennyiben ez teljesül,

```
python src/separate_dataset.py
```

Tesztfuttatás

A tesztet a GitHub repository-ban található *testing.zip* tömörített állományban található képek közül 9 db-on fogom elvégezni.

Először a program lefuttatja a predikciót a nyers képekre, melyeken autók szerepelnek.



5. ábra - Teszteléshez használt képek

Eredményül megkapjuk a felcímkézett képeket, illetve a képekről kivágott rendszámtáblákat.



6. ábra - Felcímkézett képek a predikciós algoritmus lefuttatása után



7. ábra - Eredményül kapott kivágott képek

Láthatjuk, hogy a képen, amelyen két rendszámtábla szerepelt, ott két címkézett részt is kaptunk. Ennek eredményeképp két kivágott képet is kaptunk.



3. ábra - Eredményül kapott kivágott képek, ahol több rendszámtábla található a képen

Ezután a rendszámtáblákról kapott képeken lefut a képfeldolgozási eljárás, melynek eredményéül a 2. ábrához hasonló eredmények keletkeznek.

Végül a feldolgozott képeken lefut az optikai karakterfelismerés, melynek eredménye a következő:

1. [RAP235]
2. [3SZ9372]
3. [M5SK1339]

4. [WA146KICI]
5. [AL3AG67]
6. [AKFM9G3G]
7. [PO]
8. [WSMB1746E]
9. [AIKA850]

Az eredményekből láthatjuk, hogy az optikai karakterfelismerés viszonylag pontosan működik a tiszta és éles képeknél, ahol teljesen vízszintesen helyezkedik el a rendszám. A homályos, vagy ferde feliratoknál viszont gyakran semmit, vagy csak néhány karaktert ismer fel, azt is tévesen.

Gyakori hiba, hogy ahol a felségjelzést nem sikerült teljesen eltüntetni, ott plusz karakternek érzékeli. Többször is előfordult a 0 karakternek, a G karakterrel való felcserélése.

Az előbbi hibát úgy lehetne orvosolni, ha a képfeldolgozási eljárással még pontosabban kivágható lenne a szükséges terület. Az utóbbin segíthetne egy saját OCR modell tanítása a rendszám táblán szereplő karakterekkel.

Irodalomjegyzék

- [1] Larxel, „Car License Plate Detection”. [Online]. Elérhető: <https://www.kaggle.com/datasets/andrewmvd/car-plate-detection>
- [2] P. Marimuthu, „Image Contrast Enhancement Using CLAHE”. [Online]. Elérhető: https://www.analyticsvidhya.com/blog/2022/08/image-contrast-enhancement-using-clahe/?utm_source=chatgpt.com
- [3] S. Paris, P. Kornprobst, J. Tumblin, és F. Durand, „Bilateral filtering: Theory and applications”, *Foundations and Trends in Computer Graphics and Vision*, köt. 4, sz. 1, o. 12–16, 2009, doi: 10.1561/06000000020.
- [4] T. D. Piyadasa, „Image Binarization in a Nutshell”. [Online]. Elérhető: <https://medium.com/%40tharindad7/image-binarization-in-a-nutshell-b40b63c0228e>
- [5] Ultralytics Inc., „Performance Metrics Deep Dive”. [Online]. Elérhető: <https://docs.ultralytics.com/guides/yolo-performance-metrics/#how-can-validation-metrics-from-yolo11-help-improve-model-performance>