Prepoznavanje registrskih tablic z uporabo strojnega učenja

Miha Štih (89221324), Luka Uršič (89221145)

Repozitorij projekta

`https://github.com/urluur/Registration-Plate-Recognition`

Cilj projekta

Cilj najinega projekta je bil razviti sistem za prepoznavanje in identifikacijo avtomobilskih tablic iz slik. Delovati bi moral tako, da bi mu dali sliko, program pa bi v primeru, da je na sliki uspel najti avtomobilsko tablico, vrnil registrsko številko s tablice in označil lokacijo tablice na sliki.

Izvedba projekta

Projekt sva začela tako, da sva najprej izbrala dataset, ki ga bova uporabila za projekt. Izbrala sva si dataset *"Car License Plate Detection"* z spletne strani Kaggle. Kot se je pozneje izkazalo, nisva izbrala najbolj optimalnega dataseta, saj je bil ta sestavljen iz samo 433 slik avtomobilov, kar pa je premalo, da bi se najin model lahko dobro naučil.

Ker so bile slike dataseta ločene od podatkov, sva začela z zbiranjem podatkov. S tem namenom sva pripravila `utils.py` datoteko, ki vsebuje 3 različne funkcije za ekstrakcijo podatkov, saj so bili relevantni podatki podani v XML datotekah za vsako izmed slik posebej. Tako torej pridobimo `bbox` podatke iz XML datotek oziroma `xmin`, `xmax`, `ymin`, in `ymax`, ki določijo kote pravokotnika avtomobilske tablice na sliki. Za pretvorbo slik v uporabne podatke pa sva uporabila `numpy`.

Uporaba OpenCV in Pytesseract

Ko so funkcije za pridobitev podatkov delovale, sva poskusila uporabiti OpenCV orodje za manipulacijo s slikami. Naučila sva se, kako se bere slike, kako riše pravokotnik in kako lahko napiševa številke tablice kot tekst na sliko. Poskusila sva uporabiti tudi `pytesseract` knjižnico, ki uporablja Googlov Tesseract OCR Engine za branje teksta iz slik in jo tudi uporabljala skoraj do končne verzije projekta.

Delitev podatkov in učenje modela

Nato sva uporabila funkcijo `train_test_split` iz `scikit-learn` knjižnice za naključni split podatkov na tiste, ki bodo uporabljeni za učenje in tiste, katere bova uporabila za testiranje. Za velikost test seta sva izbrala samo 15% od vseh podatkov, saj je bil izbran dataset že tako ali tako majhen, tako da nama ostane čim več podatkov za učenje modela.

Model in transformacija slik

Najin naslednji korak je bil sestavljanje in učenje modela na datasetu. Za to sva uporabila `keras`, saj je preprost za uporabo. Kot model sva izbrala najpreprostejšo verzijo `Sequential` in dodala nekaj slojev. Najin model je narejen tako, da poskuša napovedati lokacije pravokotnika, kjer naj bi bila avtomobilska tablica (lokacijo na sliki). Začela sva z tem, da sva slike transformirala do velikosti 224x224px in učila model z naslednjimi sloji.

```python
def build_custom_model(input_shape):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),

        Conv2D(64, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),
        Conv2D(128, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),
        Conv2D(256, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),
        Conv2D(512, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),
        Conv2D(256, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),
        Conv2D(128, (3,3), activation='relu', padding='same'),
        MaxPooling2D(2,2),
        Flatten(),
        Dense(1024, activation='relu'),
        Dropout(0.5),

        Dense(4, activation='sigmoid')
    ])
    optimizer = Adam(learning_rate=0.001)
    model.compile(optimizer=optimizer, loss='mean_squared_error')

    return model
```

Figure 1: Začetna sestava modela

Izboljšave modela

Ker nama model ni dal želenih rezultatov, sva poskusila več različnih kombinacij slojev in `loss` funkcij, kot nama je priporočil asistent. Izkazalo se je, da nama `mean_squared_error` še vedno daje najboljše rezultate. Poskusila sva še `IoULoss` in `GIoULoss`, a se nobena ni izkazala bolje kot `mean_squared_error`.

Struktura slojev se je medtem spremenila v naslednje:

```python
def build_custom_model(input_shape):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(3,3),

        Conv2D(64, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        AveragePooling2D(16),

        Flatten(),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(16, activation='relu'),
        Dense(8, activation='relu'),

        Dense(4, activation='sigmoid')
    ])
    optimizer = Adam(learning_rate=0.01)
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model
```

Figure 2: Končna sestava modela

Za boljše učenje sva tudi spremenila velikost slik, ki jih uporabiva za učenje in torej iz 224x224px v 500x500px, kar misliva da je malce pomagalo.

Med ugibanjem slojev pa sva napisala še funkcijo `load_and_predict`. Deluje tako, da najprej pretovori podano sliko v pravilno velikost, torej 500x500px in nato preko prej omenjenega modela poskuša najti lokacijo avtomobilske tablice. Ko lokacijo oz pravokotnik najde, pretvori sliko v črno-belo in uporabi pytesseract OCR za prepoznavo znakov z tega dela slike. Za vsak slučaj pa sva dodala še nekaj maneverskega prostora tako da prebere sliko, ki je na vsako stran za 10px večja od pravokotnika.

Po nekaj preizkusih modela sva na primeru dobila naslednji rezultat:

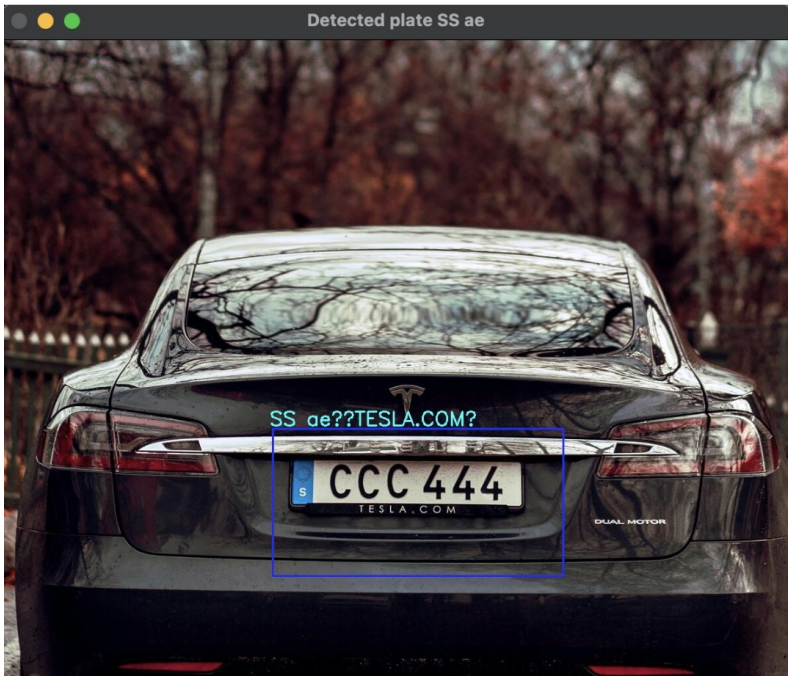

Figure 3: Začetna zaznava tablice

Kar nama je pokazalo, da prepoznavanje lokacije tablic vsaj približno deluje, medtem ko OCR ne deluje tako kot bi moral. Ker OCR ni deloval po pričakovanjih sva se odločila da ga raje zamenjava z EasyOCR, ki se je izkazal dosti bolje in sedaj bere tablice kot bi moral. To je lahko vidno na spodnji sliki (v primerjavi z prejšnjo sliko).

Figure 4: Končna zaznava tablice

Nadgradnje

Ker je na tej točki projekt deloval kot bi moral je bil premaknjen iz navadnih python datotek v Jupyter notebook, da se ga bo dalo lažje uporabljati ter nadgrajevati.

Kot dodatno sva uporabila tudi sklearn in matplotlib knjižnici za lepši prikaz delovanja najinega modela. Prikazani so Confusion Matrix, graf napačno pozitivnih v primerjavi z pravilnimi in graf točnosti.

Za konec je Luka že začel s prvo dejansko nadgradnjo, ki omogoča, da programu ni treba podati slike avtomobila, pač pa lahko z uporabo kamere tablice beremo v realnem času.

Zaključek

Projekt naju je malo presenetil, saj je bil težji od pričakovanj za izvedbo. Ker še vedno ne deluje optimalno kot predvideno, misliva, da obstaja nekaj možnih izboljšav. Poudarek bi dala na zamenjavo dataseta, kot nama je svetoval že asistent. Trenutno uporabljen dataset je premajhen, da bi se model učinkovito naučil iskati lokacije tablic na slikah in bi bil tako večji dataset več kot dobrodošel.

Skratka, vidiva tudi več drugih možnih izboljšav in nadgradenj za projekt, saj ima potencialno veliko praktično uporabnost na več različnih področjih. Vse od nadzora prometa pa do parkirišč in radarjev.

Povezave/viri

- Uporabljen dataset
- OpenCV
- Pytesseract
- Keras losses
- Train_test_split funkcija
- Numpy
- EasyOCR
- MatPlotLib
- XML parsing