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. Program naj bi ob podani sliki vrnil registrsko številko in označil lokacijo tablice na sliki, če bi bila ta uspešno prepoznana.

Izvedba projekta

Najprej sva začela z izbiro podatkovnega nabora (dataset), ki ga bova uporabila za projekt. Izbrala sva dataset *"Car License Plate Detection"* s spletne strani Kaggle. Kasneje se je izkazalo, da to ni bila najbolj optimalna izbira, saj dataset vsebuje le 433 slik avtomobilov, kar je premalo za učinkovito učenje najinega modela.

Ker so bile slike v datasetu ločene od podatkov, sva se lotila zbiranja teh podatkov. S tem namenom sva pripravila datoteko `utils.py`, ki vsebuje tri različne funkcije za ekstrakcijo podatkov, saj so bili relevantni podatki podani v XML datotekah za vsako izmed slik posebej. Tako pridobiva podatke `bbox` iz XML datotek, natančneje `xmin`, `xmax`, `ymin` in `ymax`, ki določajo kote pravokotnika avtomobilske tablice na sliki. Za pretvorbo slik v uporabne podatke sva uporabila `numpy`.

Uporaba OpenCV in Pytesseract

Ko so funkcije za pridobivanje podatkov delovale, sva poskusila uporabiti OpenCV orodje za manipulacijo s slikami. Naučila sva se, kako brati slike, risati pravokotnike in kako zapisati številke tablice kot tekst na sliko. Poskusila sva uporabiti tudi knjižnico `pytesseract`, ki uporablja Googlov Tesseract OCR Engine za branje teksta s slik, in jo uporabljala skoraj do končne različice projekta.

Delitev podatkov in učenje modela

Nato sva uporabila funkcijo `train_test_split` iz knjižnice `scikit-learn` za naključni razrez podatkov na učno in testno množico. Za velikost testnega seta sva izbrala 15% vseh podatkov, saj je bil izbran dataset že tako ali tako majhen, tako da nama je ostalo čim več podatkov za učenje modela.

Model in transformacija slik

Najin naslednji korak je bil sestaviti in naučiti model na izbranem 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 zasnovan tako, da poskuša napovedati lokacije pravokotnika, kjer naj bi bila avtomobilska tablica (lokacijo na sliki). Začela sva tako, da sva slike transformirala na velikost 224x224px in model učila z naslednjimi sloji.

```
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
```

Slika 1: Začetna sestava modela

Izboljšave modela

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

Struktura slojev se je medtem spremenila v naslednje:

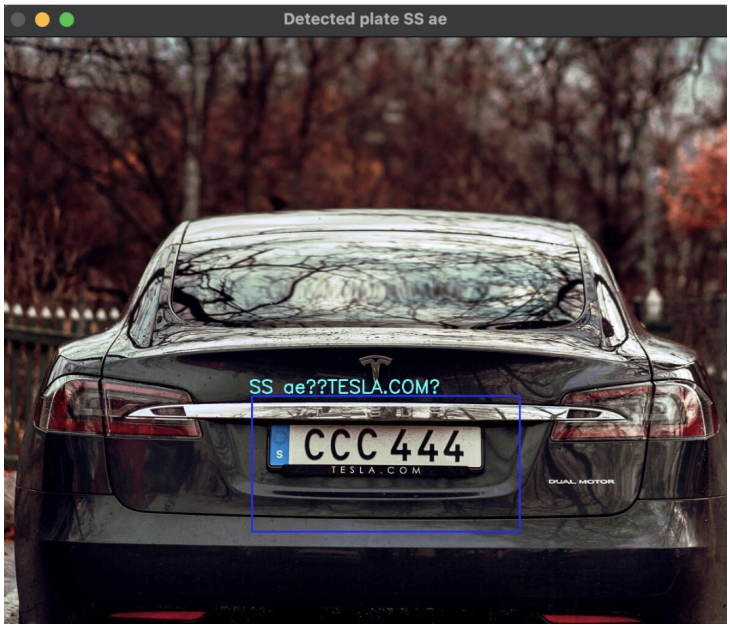
```python
def build_custom_model(input_shape):
    model = Sequential([
        Input(shape=input_shape),
        Conv2D(32, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(3,3),

        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
```

Slika 2: Končna sestava modela

Za boljše učenje sva tudi spremenila velikost slik, ki jih uporabljava za učenje, in sicer iz 224x224px na 500x500px, kar meniva, da je malce pomagalo.

Med optimizacijo slojev sva napisala še funkcijo `load_and_predict`. Deluje tako, da najprej pretvori podano sliko na pravilno velikost, torej 500x500px, nato pa z uporabo modela poskuša najti lokacijo avtomobilske tablice. Ko lokacijo oziroma pravokotnik najde, sliko pretvori v črno-belo in uporabi `pytesseract` OCR za prepoznavo znakov na tem delu slike. Za vsak slučaj sva dodala še nekaj manevrskega prostora, tako da se prebere slika, ki je na vsako stran za 10px večja od pravokotnika.

Po nekaj preizkusih modela sva na primeru dobila naslednji rezultat:



Slika 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 za dosti bolj učinkovitega in sedaj bere tablice tako, kot bi moral. To je razvidno na spodnji sliki (v primerjavi s prejšnjo sliko).

Slika 4: Končna zaznava tablice

Nadgradnje

Ker je na tej točki projekt deloval kot bi moral, sva ga premaknila iz običajnih Python datotek v Jupyter Notebook, da bo lažje za uporabo in nadgradnjo.

Kot dodatno izboljšavo sva uporabila tudi knjižnici `scikit-learn` in `matplotlib` za lepšo vizualizacijo delovanja najinega modela. Prikazani so Confusion Matrix, graf napačno pozitivnih primerov v primerjavi s pravilnimi ter graf točnosti.

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

Zaključek

Projekt naju je presenetil, saj se je izkazal za težjega, kot sva pričakovala. Ker še vedno ne deluje optimalno, meniva, da obstaja nekaj možnih izboljšav. Poudarek bi dala na zamenjavo podatkovnega nabora, kot nama je svetoval že asistent. Trenutno uporabljen dataset je premajhen, da bi se model učinkovito naučil iskati lokacije tablic na slikah, zato bi bil večji dataset več kot dobrodošel.

Poleg tega vidiva še druge možne izboljšave in nadgradnje projekta, saj ima ta potencialno veliko praktično uporabnost na več različnih področjih, vse od nadzora prometa do parkirišč in radarjev.

Povezave/viri

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