Rozpoznawanie Znaków Drogowych
Dokumentacja Projektu

Analiza obrazów

Andrzej Brzyski, Błażej Czaicki
Urszula Nowak, Dawid Sroka

22 stycznia 2023

Spis treści

1 Opis oraz założenia wstępne

Projekt został wykonany w ramach przedmiotu "Analiza obrazów". Tematem projektu jest rozpoznawanie znaków drogowych z wczytanego przez użytkownika obrazu.
Przy wstępnych założeniach program powinien poprawnie rozpoznawać znaki drogowe z wczytanych plików graficznych. Przy pomocy uczenia maszynowego program uczy się rozpoznawania znaków.

2 Użyte narzędzia

2.1 Język Python

Program został w całości wykonany w języku Python z użyciem odpowiednich bibliotek. Ważniejsze z nich zostały opisane w kolejnych podrozdziałach. Biblioteki niezbędne do uruchomienia programu zostały zapisane do osobnego pliku w celu prostego uruchomienia programu.

Biblioteka PySimpleGUI

Biblioteka przeznaczona do tworzenia prostych interfejsów w języku Python. Została użyta do utworzenia okna dialogowego programu oraz wczytania plików przez użytkownika. Zostały użyte z niej podstawowe funkcje do utworzenia okna, napisów, przycisków oraz wyświetlenia grafik.

```
import PySimpleGUI as sg

...

# result gui
result_gui = [
    [sg.Text(key="SignName")],
    [sg.Image(size=(size, size), key="Result")]
]

layout = [
    [
        sg.Column(gui, vertical_alignment="left", justification="left"),
        sg.VSeperator(),
        sg.Column(result_gui, vertical_alignment="center", justification="center")
    ]
]
```

Listing 1: Przykład użycia PySimpleGUI

Biblioteka Keras

Biblioteka do uczenia maszynowego przeznaczona dla języka Python. Wykorzystując jej przeznaczenie w projekcie m.in. utworzono model oraz dodano do niego warstwy, co zostało przedstawione na listingu poniżej. W pliku *GuI.py* do odwołania się do modelu, w celu klasyfikacji znaków, użyto funkcji *load_model*. Pierwsza z nich służy do konwersji formatu obrazu, natomiast druga pozwala na jego wczytanie.

```
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

...

model.add(Conv2D(32, (3,3), activation = 'relu', input_shape= x_train.shape[1:]))
model.add(MaxPool2D((2,2)))
model.add(Conv2D(filters = 64, kernel_size = (3,3), activation = 'relu'))
model.add(MaxPool2D(pool_size = (2,2)))
model.add(Flatten())
model.add(Dense(256, activation = 'relu'))
```

```
model.add(Dense(amount_of_diffrent_signs, activation = 'softmax'))
model.compile(loss ='categorical_crossentropy', optimizer = 'adam', metrics = ['
    accuracy'])
```

Listing 2: Przykład użycia Keras

Biblioteka OpenCV

Biblioteka zawierająca standardowe operacje wykorzystywane przy przetwarzaniu obrazów. Zostały uzyte z niej dwie funkcje: *imencode* oraz *imread*.

```
import cv2

...

res, img_to_show = cv2.imencode(".png", cv2.imread(image_path_reference_sign))
```

Listing 3: Przykład użycia OpenCV

Biblioteka Scikit-learn

Scikit-learn jest biblioteką do uczenia maszynowego, która umożliwia w języku Python przeprowadzać algorytmy klasyfikacji, regresji i klastrowania. W przedstawionym projekcie użyto z niej funkcji *train_test_split*, która umożliwia podział danych testowych na zbiory w różnych proporcjach.

```
from sklearn.model_selection import train_test_split

...

x_train, x_test, y_train, y_test = train_test_split(data, labels, test_size=0.2,
    random_state=20)
```

Listing 4: Przykład użycia Scity-learn

3 Instalacja i uruchomienie

Instalacja na systemie Ubuntu 22.04 LTS. Poniższe komendy utworzą wirtualne środowisku w celu lokalnego zainstalowania niezbędnych bibliotek oraz uruchomienia programu.

```
python3 -m venv .venv
python -m pip install -r ./requirements.txt
```

Trenowanie modelu:

```
python ./Main.py
```

Uruchomienie aplikacji:

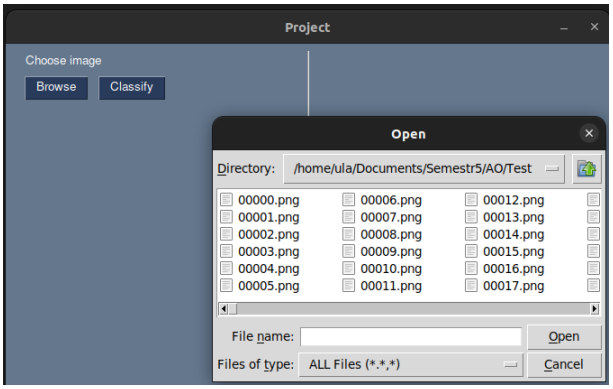
```
python ./GuI.py
```

4 Działanie programu

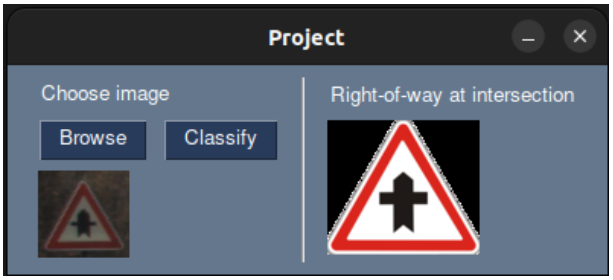
Po uruchomieniu programu wyświetli się proste okno aplikacji przedstawione na 1. Umożliwia nam ono wczytanie grafiki oraz jej klasyfikację. Program daje możliwość wczytywania dowolnych obrazów z urządzenia użytkownika. Okno podzielone jest na dwie części. Po lewej znajduje się miejsce na wczytaną grafikę, natomiast po prawej wyświetli się nazwa oraz grafika znaku, który został dopasowany.

Rysunek 1: Okno aplikacji po uruchomieniu

Przycisk *"Browse"* pozwala na wczytanie obrazu. Gdy to nastąpi należy kliknąć *"Classify* w celu klasyfikacji obrazu. Sposób wczytania grafiki oraz wynik końcowy przedstawiono poniżej.



Rysunek 2: Wczytywanie obrazu



Rysunek 3: Okno aplikacji po klasyfikacji znaku

5 Możliwości rozwoju

Ścieżką rozwoju dla prezentowanego programu mogłaby być implementacja całego zasobu znaków drogowych. Wiąże się to z wydłużonym czasem trenowania modelu, co daje również możliwość zoptymalizowania tego procesu. Przedstawiony system rozpoznawania wykrywa znaczną większość znaków na grafikach, jednak istnieje możliwość lepszego dostosowania modelu w celu zwiększenia poprawności analizy wczytywanych grafik.

5.1 Co nie działa

Poniżej przedstawiono niedoskonałości aplikacji, które występują dla tak wyszkolonego modelu.

- Program rozpoznaje grupę 20 znaków, a nie całego całego ich zbioru.
- Nie są rozpoznawane znaki skierowane po kątem.
- Problem występuje również przy znakach, które przedstawione są z dalekiej odległości. Po ich przybliżeniu program działa poprawnie.
- Może się zdarzyć problem z wczytaniem zdjęcia.

6 Podsumowanie

W projekcie zrealizowano wstępne założenia. Problem stanowi jedynie rozpoznawanie znaków z odległości lub pod pewnym kątem. W dokumentacji przedstawiono części programu sprawiające problem oraz możliwość rozwoju aplikacji.

6.1 Podział pracy

- GUI
  Dawid Sroka
- Algorytm rozpoznawania znaków
  Błażej Czaicki, Andrzej Brzyski
- Testowanie modelu
  Błażej Czaicki, Andrzej Brzyski
- Dokumentacja
  Urszula Nowak