Analiza Obrazów - Dokumentacja Projektu

Gosia Makieła, Marta Figurska, Karol Kozłowski, Bartosz Konopka

1 Opis aplikacji i algorytmów

Program składa się z aplikacji i dokumentacji obejmującej projekt z przedmiotu Analiza Obrazów, semestru piątego, kierunku Informatyka Stosowana. Aplikacja napisana w języku Python, ma za zadanie rozpoznawanie i kategoryzację zdjęcia logo samochodów. Interfejs aplikacji prezentuje trafność wytrenowanego modelu uczenia maszynowego, gwarantuje możliwość załadowania i podejrzenia zdjęcia, które za pomocą odpowiedniego przycisku możemy za pomocą modelu ML, przypisać do jednej z kategorii. Program powinien przyjmować zdjęcie widocznego loga marki samochodowej, które zostanie rozpoznane i podpięte pod odpowiednią kategorię. Dostępne w aplikacji kategorie to: $Mercedes$, $Volkswagen$, $Skoda$, $Lexus$ oraz $None$ - kategoria w któej nie rozpoznano żadnej z podanych wcześniej marek.

Projekt składa się z plików: $GUI.py$, $main.py$, $train.py$, $data\_for\_model.py$, $data\_augmentation.py$, $create\_model.py$, $cost\_funcion.py$ oraz $categorize.py$. Każdy z tych plików implementuje odpowiednie algorytmy odpowiedzialne za prawidłowe działanie aplikacji.

- $GUI.py$: Plik zawierający klasę odpowiedzialną za całą strukturę interfejsu graifcznego aplikacji. Interejs stworzony został przy użyciu biblioteki $PyQt5$. w klasie $Ui\_MainWindow$ znajduje się metoda $setupUi$ odpowiedzialna za stworzenie wszelkich części graficznych projektu, takich jak panele, przyciski czy labele. Metoda $retranslateUi$ zawiera kod odpowiedzialny za konfigurację i przygotowanie głównego okna, oraz atrybutów interfejsu graficznego.

- $main.py$: Główny plik programu, zawierający klasę $MainWindow$, dziedziczącą po klasie $Ui\_MainWindow$ zaimportowanej z pliku $GUI.py$. W metodzie $main$ dochodzi do stworzenia głównego okna aplikacji, skonfigurowania i przetworzenia go oraz wyświetlenia intefrejsu graficznego. Metoda $setupUi$, ustawia odpowiednie parametry klasy $MainWindow$, oraz definiuje jakie funkcje wywołać po wciśnięciu odpowiednich przycisków w interfejsie graficznym użytkownika. Metoda $\_loadImage$ przypięta do przycisku $Load\ Image$ implementuje algorytm załadowania zdjęcia do klasy, oraz wyświetlenie go w interfejsie. Metoda $\_recognizeImage$ podpięta do przycisku $Categorize\ Image$ odpowiada za to, za sprawdzenie czy zdjęcie zostało załadowane, a następnie inicjalizuje algorytm odpowiedzialny za kategoryzację wcześniej wczytanego zdjęcia, przy pomocy pliku $categorize.py$. Metoda ta również po otrzymaniu odpowiedniej kategorii zdjęcia, wypisuje tą kategorię w interfejsie użytkownika.

- $train.py$: Główny plik odpowiedzialny za przygotowanie zdjęć znajdujących się w datasecie (przy pomocy funkcji znajdujących się w plikach $data\_for\_model.py$ oraz $data\_augmentation.py$), oraz za stworzenie modelu (przy pomocy funkcji zjadującej się w pliku $create\_model.py$). Po wykonaniu tych operacji i zdefiniowaniu dostępnych kategorii, dane z przerobionych zdjęć dzielone są na dane na których model będzie trenowany i na dane na których model zostanie przetestowany. Ostatecznie następuje trenowanie modelu, oraz zapisanie go w strukturze plików, dzięki czemu może on zostać wykorzystany w głównej aplikacji. Wyświetlany również zostaje wykres straty modelu (przy pomocy funkcji znajdującej się w pliku $cost\_funcion.py$).

- $data\_for\_model.py$: Plik ten zawiera funkcję $prepare\_data\_for\_model$ odpowiedzialną za obróbkę zdjęć wykorzystywanych do trenowania modelu. Tworzona jest pusta

lista *images* na przechowywanie przetworzonych obrazów oraz pusta lista *labels* na przechowywanie odpowiadających im etykiet. Dla każdego katalogu z listy directories: Obliczana jest liczba obrazów w danym katalogu, znajdywana jest minimalna liczba obrazów we wszystkich katalogach. Następnie, dla każdego katalogu z listy *directories*: Pętla przechodzi przez obrazy w danym katalogu, przycinając ich liczbę do minimalnej liczby obrazów wśród katalogów. Każdy obraz jest otwierany i konwertowany do skali szarości. Rozmiar obrazu jest zmieniany na $128x128$ pikseli. Obraz jest normalizowany przez podzielenie wartości pikseli przez 255.0. Numeryczna reprezentacja obrazu (macierz) jest dodawana do listy *images*. Odpowiadająca mu etykieta jest przypisywana na podstawie nazwy katalogu i dodawana do listy *labels*. Na koniec, lista images jest konwertowana na macierz *image_array*, a wymiary są odpowiednio dostosowane do formatu oczekiwanego przez model konwolucyjny. Macierz *image_array* oraz lista *labels* są zwracane przez funkcję. Ostatecznie, funkcja ta przygotowuje dane, które mogą zostać wykorzystane do treningu modelu konwolucyjnego stworzonego wcześniej.

- *data_augmentation.py*: Ten kod używa biblioteki $TensorFlow$ do przetwarzania obrazów w celu generowania danych do augmentacji. Augmentacja danych to technika stosowana w uczeniu maszynowym, która polega na tworzeniu nowych przykładów treningowych przez zastosowanie różnych transformacji do istniejących danych treningowych, co pomaga zwiększyć różnorodność zbioru danych. Na początku tworzony jest obiekt $ImageDataGenerator$ z określonymi parametrami augmentacji, takimi jak zakres rotacji, przesunięcia szerokości i wysokości, zakres pochylenia, zakres przybliżenia, poziome odbicie lustrzane itp. Następnie przy pomocy funkcji *create_augmented_images* która operuje na bibliotekach datasetu, dzięki pętlom które przechodzą po wszystkich zdjęciach, zapisuje je do macierzy obrazu i augmentuje zdjęcia, dzięki czemu otrzymujemy o wiele więcej danych potrzebnych do wytrenowania modelu.

- *create_model.py*: Ten kod definiuje model konwolucyjnej sieci neuronowej przy użyciu biblioteki $TensorFlow$ i modułu $Keras$.

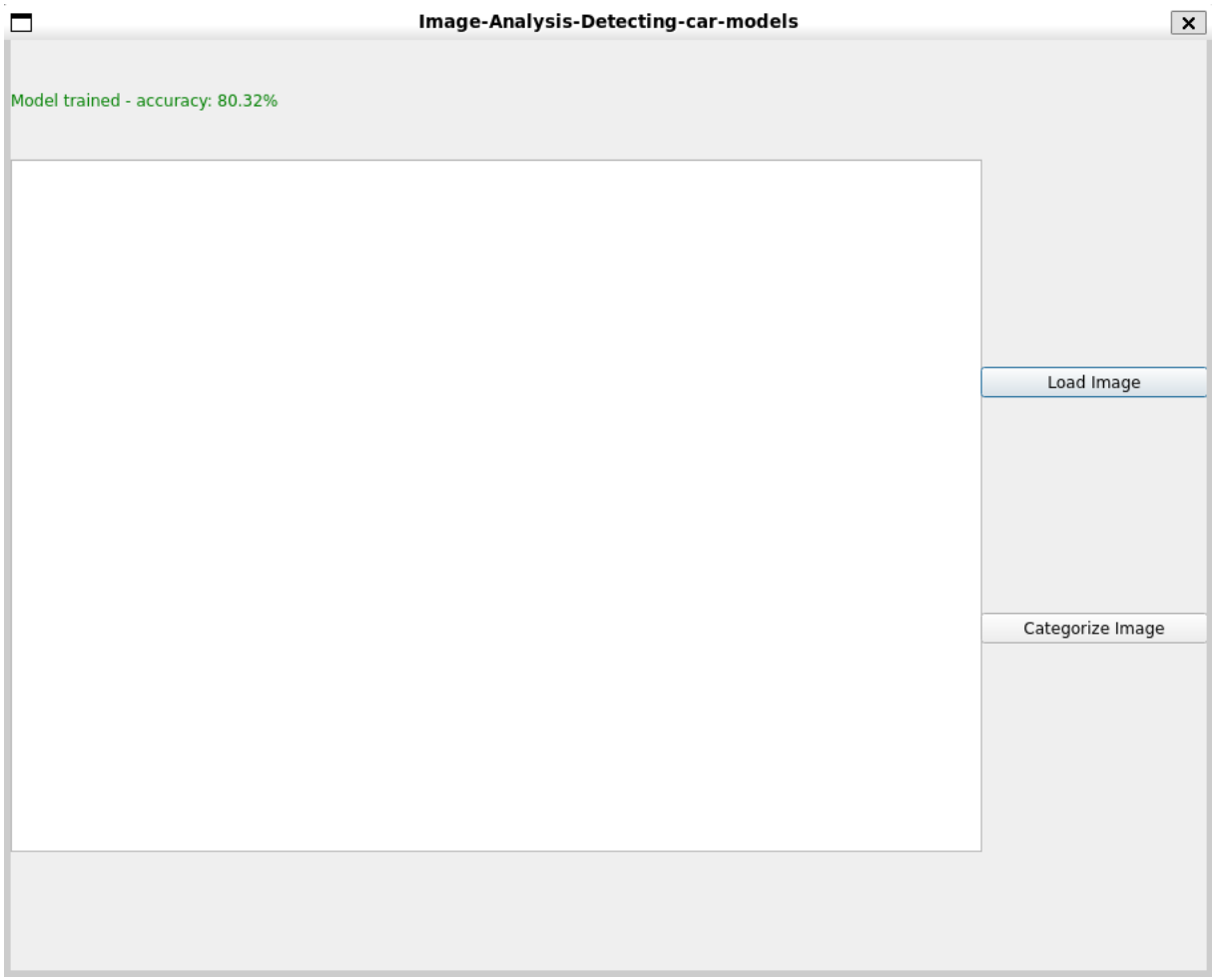
- *cost_funcion.py*:Ten kod definiuje funkcję *plot_cost_function*, która jest używana do stworzenia wykresu ewolucji funkcji straty podczas treningu modelu. Model ten jest zaprojektowany do zadania klasyfikacji obrazów na trzy klasy, ponieważ ostatnia warstwa ma 3 neurony z funkcją aktywacji $softmax$. Warto zauważyć, że dane wejściowe są oczekiwane w formie $(128, 128, 1)$, co mówi, że modele są przystosowane do obsługi obrazów o rozmiarze $128x128$ pikseli z jednym kanałem (na przykład obrazy w odcieniach szarości).

- *categorize.py*: Plik odpowiedizalny za kategoryzację wcześniej wczytanego zdjęcia z pliku *main.py*, przy pomocy wytrenowanego modelu uczenia maszynowego. Na początku ładowany jest z plików model, a zdjęcie zostaje odpowiednio załadowane i przetworzone - rozmiar zdjęcia jest zmieniany do rozmiarów $128x128$, zdjęcie konwertwane jest do odcieni szarości, oraz zostaje ono znormalizowane do wartości double, tak żeby zares informacji o zdjęciu znajdował się w przedziale $[0; 1]$. Ostatecznie przy pomocy modelu następuje podpięcie zdjęcia pod jedną z instiejących kategorii, a wynik działania algorytmu zwracany jest do pliku *main.py*.

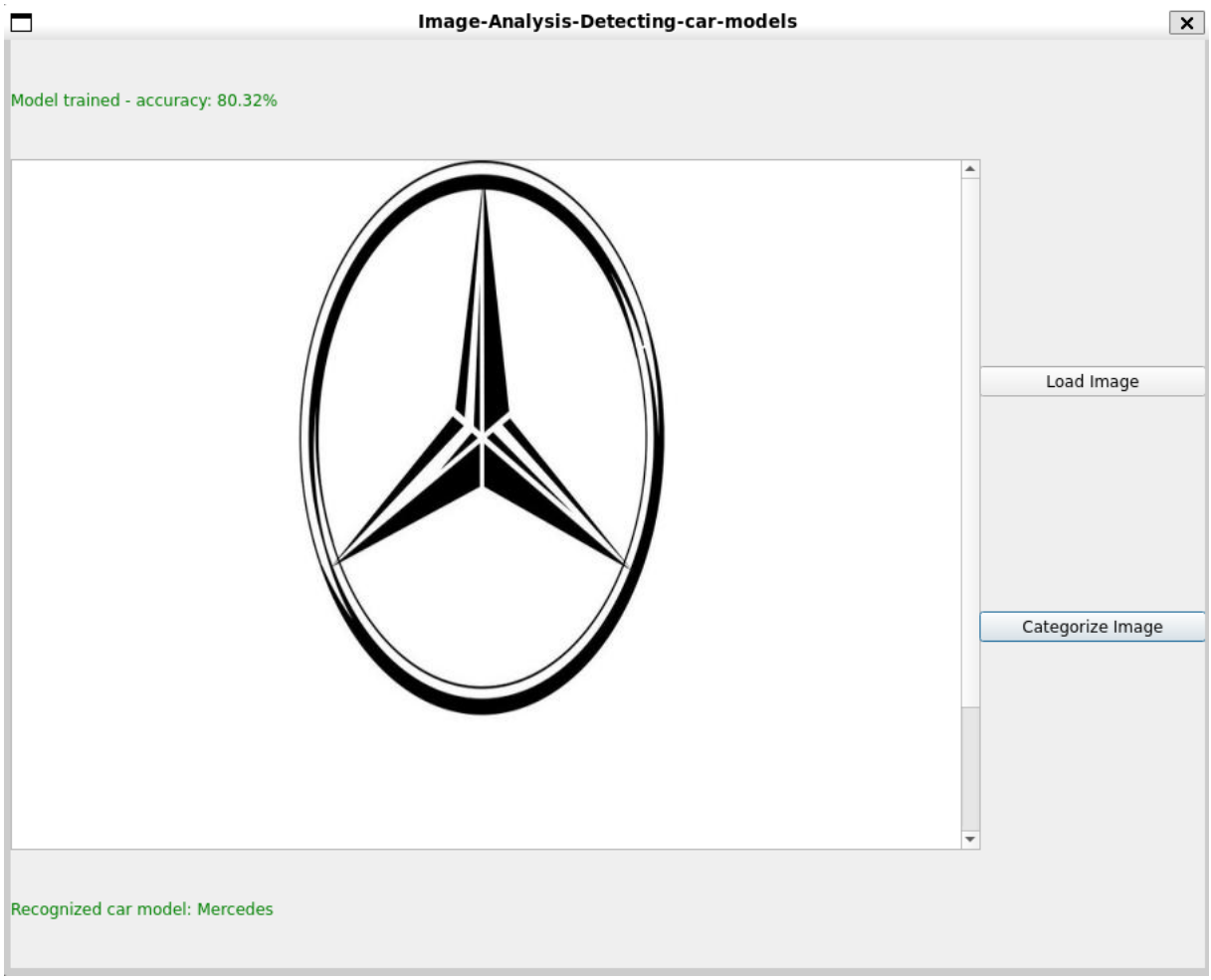
Pliki projektu zawierają również model uczenia maszynowego $model1.h5$, wytrenowany w celu kategoryzacji zdjęć loga samochodów. W plikach można również znaleźć folder

test_images, w którym znaleźć można zdjęcia pomocne w trakcie testowania działania aplikacji.

2 Instrukcja obsługi aplikacji



Rysunek 1: Wygląd interfejsu graficznego aplikacji, bez wczytanego i rozpoznanego zdjęcia



Rysunek 2: Wygląd interfejsu graficznego aplikacji, ze wczytanym i rozpoznanym zdjęciem

Obsługa aplikacji przy wytrenowanym modelu jest stosunkowo prosta. Aby uruchomić aplikację, należy użyć komendy *python3 main.py*, pamiętając o posiadaniu wszystkich potrzebnych bibliotek. Po otworzeniu interfejsu graficznego, w górnej części okna aplikacji otrzymujemy informację potwierdzającą istnienie wśród plików wytrenowanego modelu ML potrzebnego do działania aplikacji, oraz informację o dokładności działania modelu w procentach.

Środkowy panel składa się z miejsca odpowiedzialnego za graficzne przedstawienie załadowanego zdjęcia, oraz z dwóch przycisków: *Load Image* oraz *Categorize Image*. Pierwszy przycisk ma za zadanie wczytanie z dysku zdjęcia w dostępnych formatach: *.png*, *.jpg*, *.jpeg*, oraz wyświetlenie tego zdjęcia w odpowiednim obszarze. Drugi przycisk wywołuje odpowiedni algorytm, który wczytane wcześniej zdjęcie za pomocą istniejącego modelu, kategoryzuje, oraz wyświetla pod załadowanym zdjęciem informację o tym, jaka kategoria została rozpoznana na obrazku. Jeżeli zdjęcie nie zostało wczytane, a użytkownik wcisnął przycisk *Categorize Image*, wyświetlona zostanie informacja o tym, że zdjęcie nie zostało wczytane.

W momencie, gdy model nie został wytrenowany, w górnej części interfejsu graficznego aplikacji wyświetlony zostanie komunikat o braku niezbędnego do działania programu pliku z modelem uczenia maszynowego. W takim przypadku, należy pobrać odpowiedni

dataset znajdujący się pod tym linkiem: `https://drive.google.com/drive/folders/1ENMmsX_74hF0vcwBLtJv53shgnVJ9E_L?usp=sharing` i umieścić go w folderze z plikami aplikacji. Następnie należy uruchomić plik *train.py*, za pomocą komendy *python*3 *train.py*, pamiętając o zainstalowaniu odpowiednich bibliotek. Wywołanie tego pliku powinno w odpowiedni sposób przetworzyć zdjęcia znajdujące się w datasecie, a następnie na ich podstawie wytrenować model uczenia maszynowego, który zostanie zapisany. Po zapisaniu modelu, możemy korzystać z aplikacji w normalny sposób, opisany powyżej.

3 Podział obowiązków

Każdy z członków zespołu brał czynny udział w projektowaniu i implementacji projektu. Podział obowiązków prezentuje się następująco:

- Gosia Makieła: Trenowanie i rozbudowanie modelu uczenia maszynowego, pomoc w przygotowaniu dokumentacji, zbieranie zdjęć do datasetu.
- Marta Figurska: Trenowanie i rozbudowa modelu uczenia maszynowego, przetestowanie prawidłowego działania aplikacji, pomoc w implementacji algorytmów odpowiedzialnych za obróbkę zdjęć.
- Karol Kozłowski: Przygotowanie plików odpowiedzialnych za algorytmy wykonujące prawidłowe przetworzenie zdjęć w celu wytrenowania modelu uczenia maszynowego, utworzenie dockera, zadbanie o międzyplatformowość aplikacji.
- Bartosz Konopka: Budowa graficznego interfejsu użytkownika, wstępne przygotowanie skryptu trenującego model uczenia maszynowego, napisanie dokumentacji projektu, pomoc w zbieraniu zdjęć do datasetu.

4 Wadliwe elementy aplikacji

Jedną z wad aplikacji jest brak możliwości wczytania innych formatów plików graficznych niż *.png*, *.jpg*, *.jpeg*. Inne formaty zdjęć nie są obsługiwane przez aplikację. Problematiczną częścią programu jest również brak rozbudowy modelu na więcej kategorii samochodów. Aplikacja niestety działa tylko i wyłącznie na podanych wcześniej modelach aut oraz wymaga widocznego loga pojazdu do prawidłowego działania. Projekt nie został rozszerzony na więcej kategorii z powodu bardzo dużej ilości zdjęć w datasecie, potrzebnych do prawidłowego wytrenowania modelu uczenia maszynowego. Większa ilość kategorii nie byłaby możliwa do przetworzenia w obecnej fazie. W aplikacji również trzeba ręcznie ustawić poprawność działania modelu uczenia maszynowego przy każdym nowym wytrenowaniu modelu, ponieważ informacja o dokładności (accuracy) modelu nie jest w nim zapisywana. Jednym z problemów jest rónież dokładność wytrenowanego modelu w wysokości $75,19\%$. Dokładność nie jest abrdzo wysoka, więc program może czasem działać błędnie, przypisując inną, nieprawidłową kategorię.

Pomimo paru wad w działaniu aplikacji, uczestnicy projektu starali się, aby była ona jak najbardziej przejrzysta dla użytkownika i zgodna z wymaganiami projektowymi.