AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
WYDZIAŁ FIZYKI I INFORMATYKI STOSOWANEJ

Dokumentacja techniczna

Klasyfikacja rodzaju guza na podstawie analizy zdjęć z rezonansu magnetycznego

Autor: Joanna Blaszka, Dawid Justyna, Zuzanna Wachulak
Kierunek studiów: Fizyka Medyczna
Przedmiot: Analiza Obrazów

Kraków, 2022

Spis treści

1 Opis programu

Cancer Detector to program umożliwiający rozpoznanie rodzaju guza na podstawie analizy zdjęcia z rezonansu magnetycznego, które do niego wczytujemy. Program wyświetla dane pacjenta (imię, nazwisko, wiek) oraz wskazuje który rodzaj guza jest najbardziej prawdopodobny. Wyświetla również prawdopodobieństwa występowania pozostałych rodzajów.

2 Autorzy projektu

- **Joanna Blaszka**:
  Przygotowanie interfejsu użytkownika.
- **Dawid Justyna**:
  Wytrenowanie sieci.
- **Zuzanna Wachulak**:
  Napisanie dokumentacji, tworzenie danych przykładowych pacjentów.

3 Uruchamianie programu

3.1 Instalacja potrzebnych programów

W celu uruchomienia programu należy posiadać zainstalowaną najnowszą wersję *Python'a*. Jeśli potrzebujesz go zainstalować można to zrobić na tej stronie `https://www.python.org/downloads/`.

3.2 Instalacja potrzebnych bibliotek

Należy posiadać również zainstalowane następujące biblioteki:

Pillow, **xlrd** wersja 1.2.0, **pypiwin32**, **CustomTkinter**,
NumPy, **OpenCV-Python**, **sklearn**, **Joblib**, **os**, **tkinter**

Jeśli potrzebujesz je zainstalować można to zrobić wpisując w oknie wiersza poleceń następujące komendy:

```
pip install pillow
pip install xlrd==1.2.0
pip install pypiwin32
pip install customtkinter
pip install numpy
pip install opencv-python
pip install sklearn
pip install joblib
```

3.3 Uruchamianie

Jeśli posiadasz już wszystkie potrzebne programy i biblioteki możesz uruchomić *Cancer Detector*. Wystarczy, że w oknie wiersza poleceń wpiszesz

```
python Cancer_Detector.py
```

Pamiętaj, żeby podczas wykonywania powyższego polecenia znajdować się w folderze zawierającym wywoływany program.

3.4 Dane wejściowe

Dane wejściowe muszą być zapisane w formacie *.xls*. W pliku takim powinny znajdować się:

- Zdjęcie do analizy
- Imię pacjenta
- Nazwisko pacjenta
- Wiek pacjenta

Każda z tych danych powinna znajdować się w kolejnych kolumnach pierwszego wiersza arkuszu.

3.5 Dane wyjściowe

Po załadowaniu danych wejściowych dane wyjściowe wyświetlone będą w formie zbliżonej do pokazanej na rys. nr (2).

4 Trenowanie sieci

Do rozpoznawania rodzajów guza użyliśmy biblioteki **Sklearn** z metodą **SVC()** - Support Vector Classifier. Metoda ta rozdziela nasze dane na hiperpłaszczyzny klasyfikując je do poszczególnych kategorii. W naszym przypadku są to 4 kategorie:

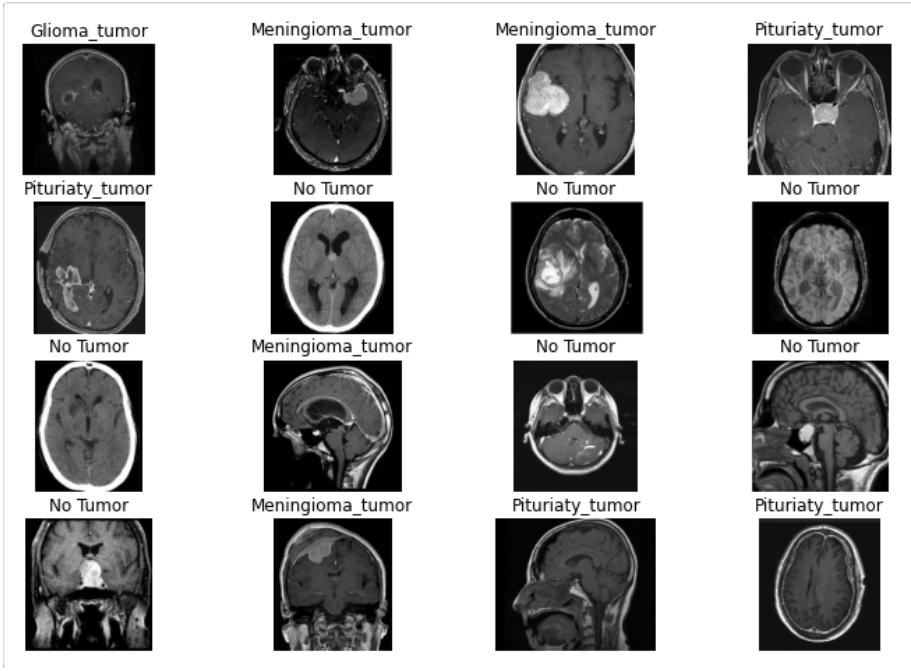
- Brak guza - No_tumor
- Guz przysadki - $Pituitary\_tumor$
- Glejak - $Glioma\_tumor$
- Oponiak mózgu - $Meningioma\_tumor$

Zbiór treningowy (znajduje się w katalogu training) do uczenia pobraliśmy ze strony: `https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri?select=Training` Do wczytywania, konwersji do odcieni szarości i ustalania rozmiarów obrazów wykorzystaliśmy bibliotekę **OpenCV** i dostępnie w niej funkcję **imread()** i **rezsize()**. Następnie przygotowaliśmy nasze dane do uczenia. Do przetwarzania obrazu wykorzystaliśmy bibliotekę **NumPy** i jej funkcję **reshape()**, która nasz obraz zapisany w macierzy dwuwymiarowej sprowadza do jednego wymiaru. Kolejnym krokiem była normalizacja danych. Wiemy, że każdy piksel może przyjmować wartości od 0-255. My

chcemy jednak, żeby te wartości były z zakresu 0-1, dlatego cały zbiór dzielimy przez wartość maksymalną, czyli 255. Kolejnym krokiem było rozdzielenie naszych danych na testowe i treningowe wykorzystując funkcję **train_test_split**. Dane podzieliśmy w stosunku 80% dane do wytrenowania i 20% dane do testowania. Następnie korzystając z biblioteki **sklearn.svm** wywołaliśmy funkcję **SVC()**, która będzie naszym modelem uczenia, ustawiając parametr *probability* na wartość *True*. Spowodował on, że nasz model nie tylko zwróci nam rodzaj guza, ale również prawdopodobieństwo występowania każdego z nich. Użyliśmy tego, ponieważ nie chcemy, żeby nasz program stawiał konkretne diagnozy, ale był pomocny dla lekarza. Otrzymaliśmy następujący wynik trenowania:

```
Training Score: 93.9  %
Testing Score: 84.49  %
```

Widzimy, że dla nieznanych danych algorytm osiągnął 84.49 % skuteczności wykrywania poprawnego rodzaju guza lub jego braku.
Następnie wyświetliliśmy kilka przykładowych obrazów oraz to jak algorytm rozpoznał rodzaje guza:



Rysunek 1: Przykłady trenowania.

Zapisaliśmy nasz model, aby móc go wykorzystywać w głównym programie. Do zapisu wykorzystaliśmy bibliotekę **joblib** z funkcją **dump()**. Zapisany model nazywa się **trained_model.sav**

5 Interfejs użytkownika

Do stworzenia interfejsu użytkownika wykorzystaliśmy bibliotekę **Custom tkinter**, która jest oparta na bibliotece **Tkinter**. Umożliwia ona stworzenie aplikacji o nowoczesnym wyglądzie.
Interfejs naszego programu jest zbudowany z dwóch paneli. Jeden służy do wyświetlania zdjęć, a drugi do wyświetlania informacji o pacjencie oraz wyników pochodzących z sieci na temat wczytywanego pliku.
Oprócz tego mamy dwa przyciski, jeden służy do załadowania danych pacjenta, a drugi umożliwia wyjście z aplikacji, na dole po prawej stronie znajduje się suwak, który umożliwia wyłączenie trybu ciemnego aplikacji.

6 Testowanie programu

Przy testowaniu programu chcieliśmy sprawdzić czy dane ładują się w poprawny sposób, a także czy wgrany model działa poprawnie. Dla przykładu:

6.1 Test 1

Rysunek 2: Odpowiedź programu na wgranie pierwszych danych testowych.

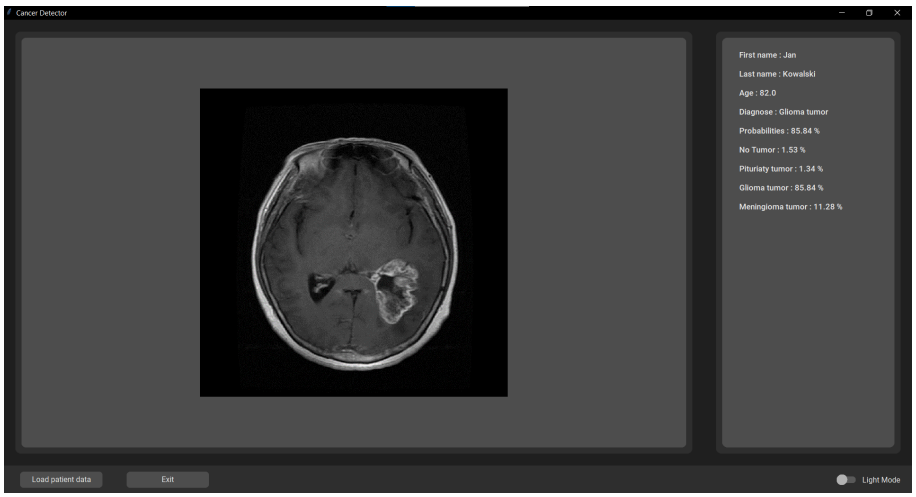
Załadowanie i rozmieszczenie danych zostało wykonanie poprawnie. Model poprawnie wykrył również diagnozę, ponieważ w tym przypadku załadowaliśmy zdjęcie, ze znanej kategorii "Glioma_Tumor", taką też otrzymaliśmy.

6.2 Test 2

Rysunek 3: Odpowiedź programu na wgranie drugich danych testowych.

W drugim przykładzie podaliśmy już nieznane zdjęcie, dlatego nie jesteśmy w stanie ocenić poprawności modelu, reszta programu w dalszym ciągu działa poprawnie.

6.3 Test 3



Rysunek 4: Odpowiedź programu na maksymalizowanie ekranu.

Sprawdziliśmy także, czy program dobrze zareaguje na zmianę rozmiarów okna. Widzimy na obrazku (4), że interfejs odpowiednio dostosował rozmiary poszczególnych paneli.

7 Podsumowanie

Celem naszego projektu było rozpoznawanie ze zdjęć rezonansu magnetycznego rodzaju guza mózgu oraz obliczanie rozkładu prawdopodobieństwa, a następnie wyświetlanie tych informacji. Główne założenia projektu zostały spełnione, a nasz program działa poprawnie. Są jednak w nim rzeczy, które chcielibyśmy w przyszłości poprawić. M.in. chcielibyśmy stworzyć bazę danych, która będzie przetrzymywać dane pacjentów, obecnie dane te przechowywane są w Excelu. Kolejną rzeczą, która wymaga poprawy jest algorytm uczenia. Obecnie poprawne wykrycie rodzaju guza występuje w 84.49% przypadków. Mimo dobrej skuteczności naszym celem jest osiągnięcie jeszcze wyższej wartości. Być może nabierając większego doświadczenia w klasyfikacji danych, spróbujemy dopasować lepszy algorytm. Rozważamy również, czy w przyszłości nie rozszerzyć aplikację o możliwość detekcji innych rodzajów guzów.
Pomimo tego, że jesteśmy świadomi kilku niedociągnięć jesteśmy zadowoleni z naszej aplikacji. Uważamy, że jej dalszy rozwój mógłby przyśpieszyć prace lekarzom i być dla nich wskazówką do dalszej diagnozy.

8 Źródła pomocnicze

Podczas tworzenia projektu korzystaliśmy z poniższych źródeł:

- `https://github.com/akd6203/brain-tumor-detection`
- `https://github.com/TomSchimansky/CustomTkinter`