

Sprawozdanie z projektu Wykrywanie naczyń siatkówki oka

Eryk Szpostański 136811

1 Skład grupy

Eryk Szpostański nr indeksu 136811

2 Zastosowany język programowania oraz dodatkowe biblioteki

Python:

- streamlit (do GUI)
- skimage (przetwarzanie obrazów)
- sklearn (drzewo decyzyjne oraz ocena skuteczności)
- numpy
- tqdm (wyświetlania paska postępu w terminalu)
- numba (przyspieszanie działania, niestety nie wszędzie dało się to wykorzystać)
- matplotlib (wyświetlanie obrazów, jak tqdm dotyczy głównie gałęzi research)
- podstawowe biblioteki takie jak pickle, marshal, os, re

Wszystkie biblioteki można zainstalować np. bezpośrednio z pomocą pip

3 Opis zastosowanych metod

3.1 Przetwarzanie obrazów

przetwarzanie wstępne (klasa src.ProcessImage fun preprocesing)

Rozpoczynam od zmiany temperatury obrazu (a przynajmniej czegoś na jej wzór), co sprowadza się do przemnożenia wartości koloru czerwonego

i niebieskiego odpowiednio przez 0.5 oraz 2.991, oczywiście ograniczając wynik do dopuszczalnego przedziału. Następnie zamieniam kolory na odcienie szarości, a potem poprawiam kontrast poprzez adaptacyjne wyrównanie histogramu oraz skurczanie zakresu intensywności.

Funkcje: src.ProcessImage.color_change, skimage.color.rgb2gray, skimage.exposure.equalize_adapthist, skimage.exposure.rescale_intensity
Uzasadnienie: Szczerze mówiąc zacząłem od próbowania wszystkich opcji w programie 'darktable' (tylko taki miałem dostępny) i po uzyskaniu zdowalającego efektu próbowałem odtworzyć ten efekt, lecz przez enigmatyczne opisy, które wskazywały na jedną czynność, a robiły coś innego lub pojedyncze błędy (choćby zmniejszanie temperatury obrazu powodowało, że staje się bardziej niebieski) to zadanie było utrudnione. Ostatecznie skupiłem się na kilku głównych etapach i udało mi się uzyskać zbliżony efekt, który nie najgorzej podkreśla naczynia siatkówki.

metoda "podstawowe - Meijering" (src.SimpleMethod2 fun calculate)

Biorę maxima z minimów, przy czym pole dla którego wyszukuje minima jest większe od tego dla maximów. Dzięki temu obraz jest bardziej ujednolicony kolorowo i delikatnie zwiększone są obszary czarne/cienie, które odpowiadają aktualnie za naczynia. Następnie używam filtra meijering i uzyskany wynik binaryzuje. wartości powyżej 0.15 otrzymuję wartość 1, a pozostałe 0.

Uzasadnienie: Przez wykonanie $\max(\min())$ zmniejszam ilość artefaktów i innych zakłóceń, kosztem pogrubienia naczyń. Obraz przepuszczam przez filtr, aby znaleźć naczynia, a następnie rzutuje wartości na 1 i 0, tak aby wynik przedstawiał klasyfikację każdego z pikseli.

metoda "podstawowe - progowanie Sauvola"

(src.SimpleMethod fun calculate)

Obliczam próg Sauvola, a następnie sprawdzam, gdzie nie został przekroczyony. Filtruje wynik filtrem gaussowskim (z sigmą odpowiadającą rozmiarowi obrazu) i na koniec sprowadzam wartości do 1, gdy przekroczą 0.235, i 0 w przeciwnym przypadku.

Uzasadnienie: Wybrałem okurat próg Sauvola, po przetestowaniu kilku analogicznych rozwiązań (między innymi otsu). Stosuje filtr Gaussa, aby pozbyć się pojedynczych punktów, które również przedostały się przez próg. Argument sigma jest wybierany na podstawie regresji uzyskanej z kilku punktów, dla których uznałem, że rozmycie jest odpowiednie. Analogicznie rzutuję wartości na 1 i 0, z tą różnicą, że wykorzystuje również maskę (uzyskaną z src.ProcessImage fun get_mask), jeśli w pierwotnym obrazie wartość koloru czerwonego była mniejsza niż 0.12 to True, gdzie dla wartości True piksel nie może uzyskać 1.

3.2 Uczenie maszynowe

przygotowanie danych (learn.prepareFiles fun prepare_files)

Rozpoczynam od ręcznego przeniesienia części zdjęć dna oka (wszystkich

oprócz 5 użytych w kolejnej części sprawozdania, czyli 88 zdjęć) do osobnego folderu, z którego następnie podany skrypt będzie czytał, a później będą one użyte do uczenia i walidacji. Skrypt ten zaczyna od przeskakowania obrazów (orginalnego, jak i tego po wstępnym przetworzeniu) tak aby zachowały proporcję, ale uzyskały długość 500 pikseli. Tworzę maskę odzwierciedlającą gdzie obraz wskazuje na ramkę. Następnie losuje 23% (wszystkich) pikseli, ze zbioru tych zawierających obraz oka i oddalonych od brzegu obrazu o co najmniej 3 i dla każdego wyliczam wariancje kolorów, momenty centralne (do 4 stopnia) oraz momenty Hu dla fragmentu obrazu 7x7 z wylosowanym środkowym pikselem. Zapisuje wszystko w pliku.

wstępne przetwarzanie zbioru uczącego (learn.prepareFiles fun read_data)

Odczytując dane z pliku odrzucam te, które zawierają wartości Nan, gdyż powodowałyby one błędy. Przed samym procesem uczenia podzielenie zbioru na uczący i testujący w 7:3

zastosowane metody uczenia (learn.learnTree)

Wykorzystałem drzewo decyzyjne (z biblioteki sklearn, a dokładniej DecisionTreeClassifier) testowałem kilka ustawień, które uznałem za znaczące, jak również integrowałem w sam zbiór danych (zwiększałem stosunek ilości naczyń do ilości tła, co przynosiło polepszenie w wyniku czułości, lecz przetworzony obraz za pomocą takiego drzewa był zdecydowanie gorszy, miał mnóstwo artefaktów), głębokość drzewa pogarszała wyniki, gdy zeszła ponizej pewien próg, także ostatecznie zmieniłem domyślne kryterium oceny podziału na entropię, co dodawało niecały punkt procentowy do czułości, resztę parametrów pozostawiając bez zmian. Wcześniej optymalizowałem jedynie ogólną trafność i próbowałem tylko dość brutalnie zbalansować zbiór danych, co skutkowało zapewne jego dużym zmniejszeniem i w konsekwencji brakiem poprawy wizualnej jakości. Teraz spróbowałem użyć RandomizedSearchCV na DecisionTreeClassifier optymalizującego F lub zbalansowaną dokładność (czyli według sklearn średnią czułości uzyskanych przez każdą klasę, jeśli się nie myleć to dla klasyfikacji binarnej jest to średnia z czułości i swoistości)

wyniki wstępnej oceny klasyfikatora

Testy hold-out:

dokładność - 97.6756606%

czułość - 30.4650092%

swoistość - 98.8854901%

średnia arytmetyczna czułości i swoistości - 64.67524964%

Otrzymane drzewo decyzyjne ma głębokość 71, a pogorszenie czułości można zaobserwować koło głębokości 42.

Dla drzewa uzyskanego z RandomizedSearchCV optymalizującego F:

dokładność - 97.5128404%

czułość - 32.7975732%

swoistość - 98.6977953%

średnia arytmetyczna czułości i swoistości - 65.7476843%

Dla drzewa uzyskanego z RandomizedSearchCV optymalizującego zbalansowaną dokładność:

dokładność - 76.9039581%

czułość - 77.1395985%

swoistość - 76.8996766%

średnia arytmetyczna czułości i swoistości - 77.0196376%

Drzewo to mierzy jakość za pomocą entropii, rozdziela dane wybierając najlepszy losowy podział, ma największą głębokość 10 oraz wagami klasify jest procentowy udział w zbiorze trenującym drugiej klasy (dla 0: 0.018, dla 1: 0.982, gdzie 1 to naczynie).

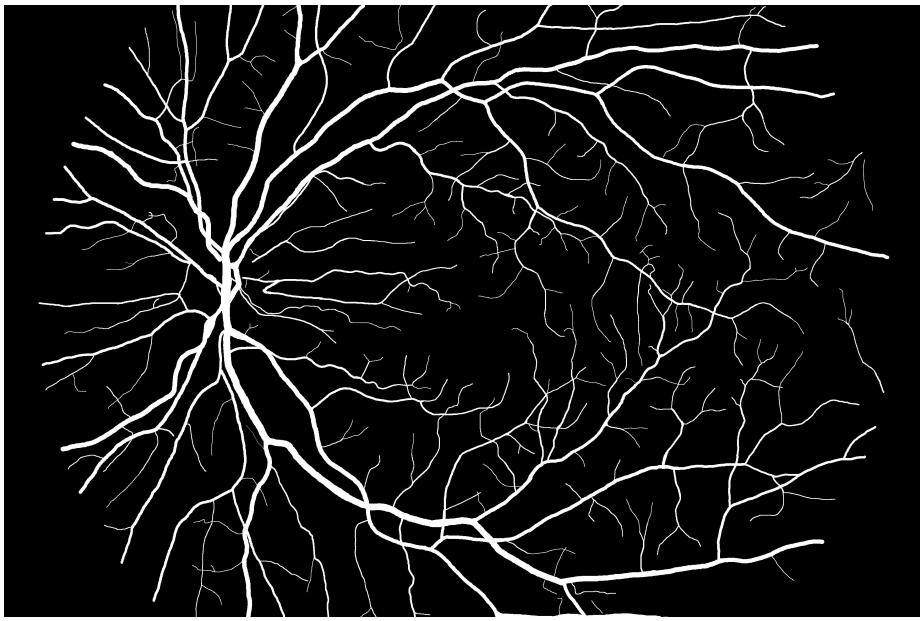
uzasadnienie

Wielkość zbioru danych wyniosła trochę ponad 4 miliony pikseli z obliczonymi momentami i wariancjami. W pliku CSV zajmuje on 2.6GB, a w pickle o połowę mniejszej. Podczas przetwarzania i uczenia zajętość pamięci podręcznej sięgała do 75% (z włączonym PyCharmem), więc w teorii mógłbym jeszcze trochę zwiększyć rozmiar bazy, lecz samo jej uzyskanie i tak trwało kilkudziesiąt minut, więc ostatecznie nie zdecydowałem się na jej zwiększanie. (Szczególnie, że z rozpiędu zacząłem robić klasyfikator kNN to już przy tej wielkości bazy nie doczekałem się końca nauczania, pomimo kilkudziesięciu minut oczekiwania)

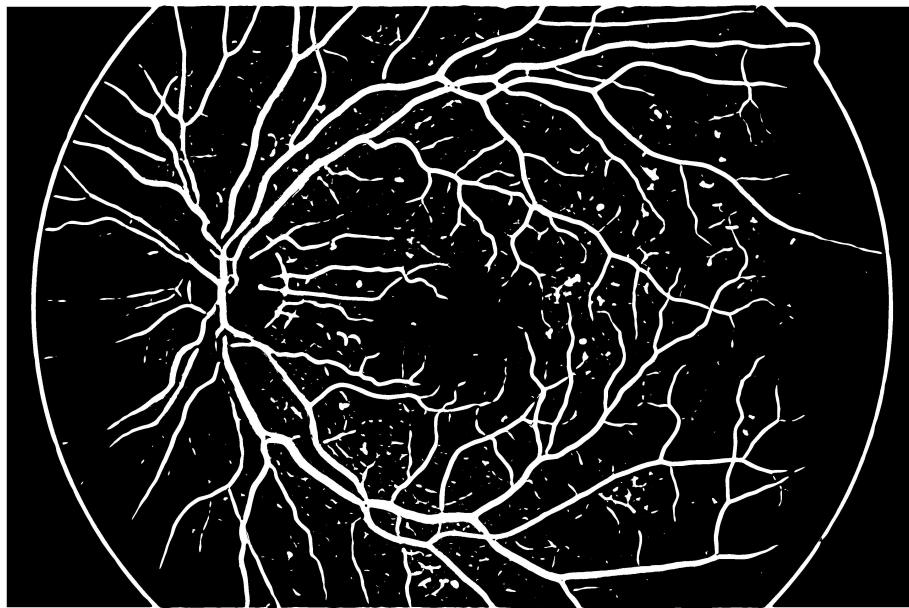
4 Wizualizacja wyników

4.1 Obraz 1

Maska ekspercka:



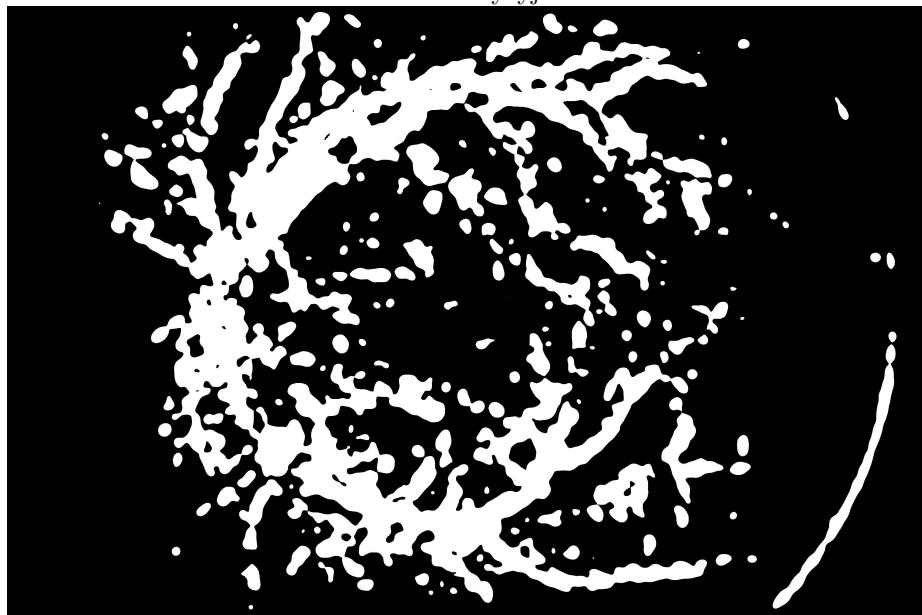
Meijering:



progowanie Sauvola:

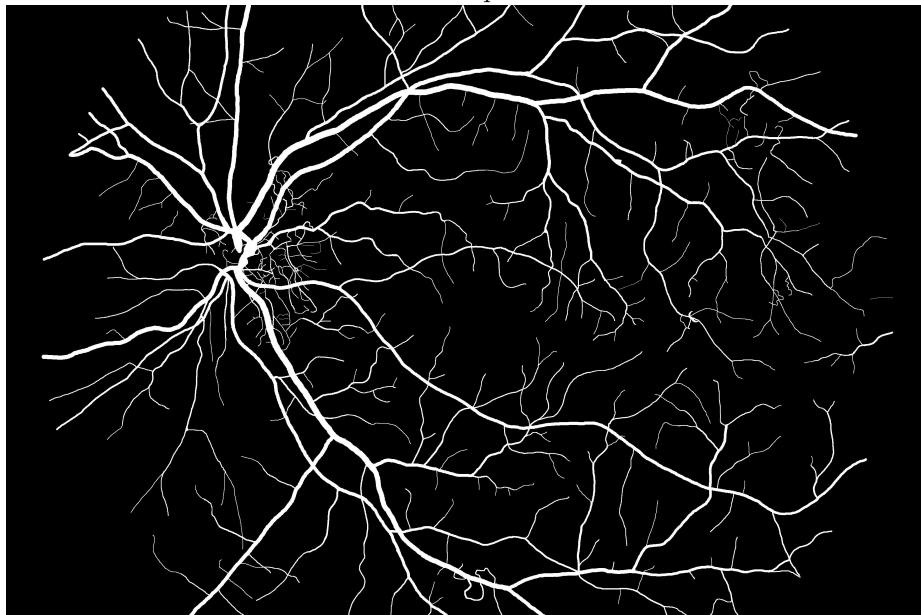


Drzewo decyzyjne:

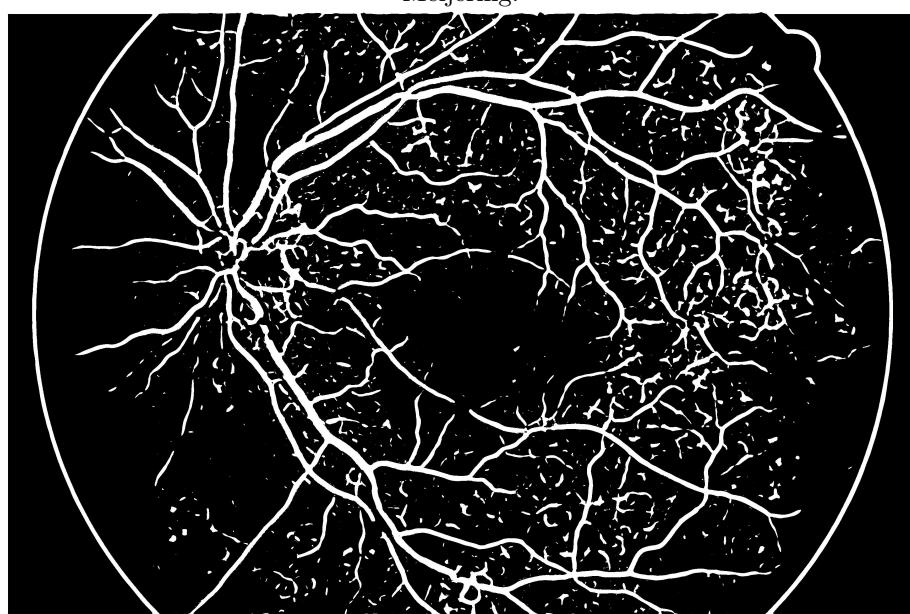


4.2 Obraz 2

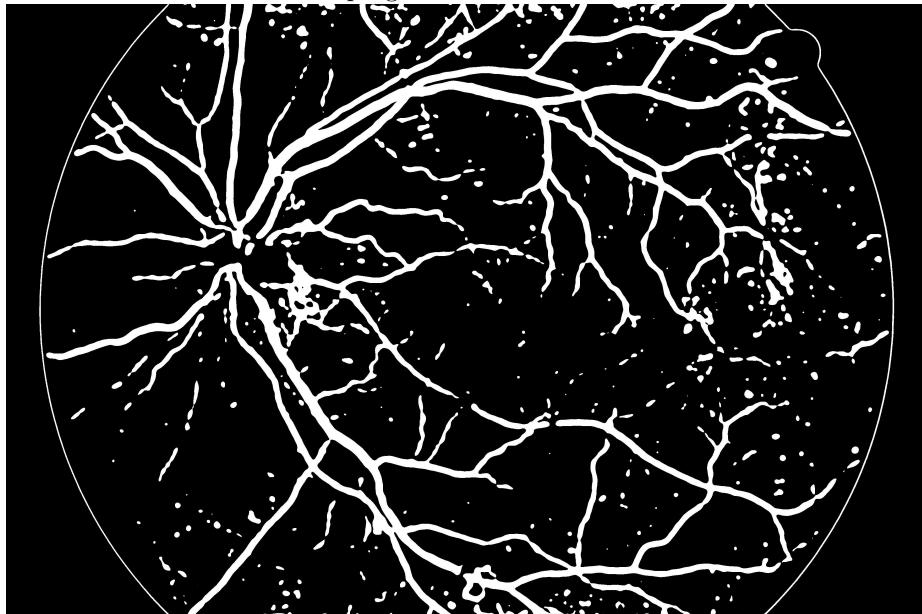
Maska ekspercka:



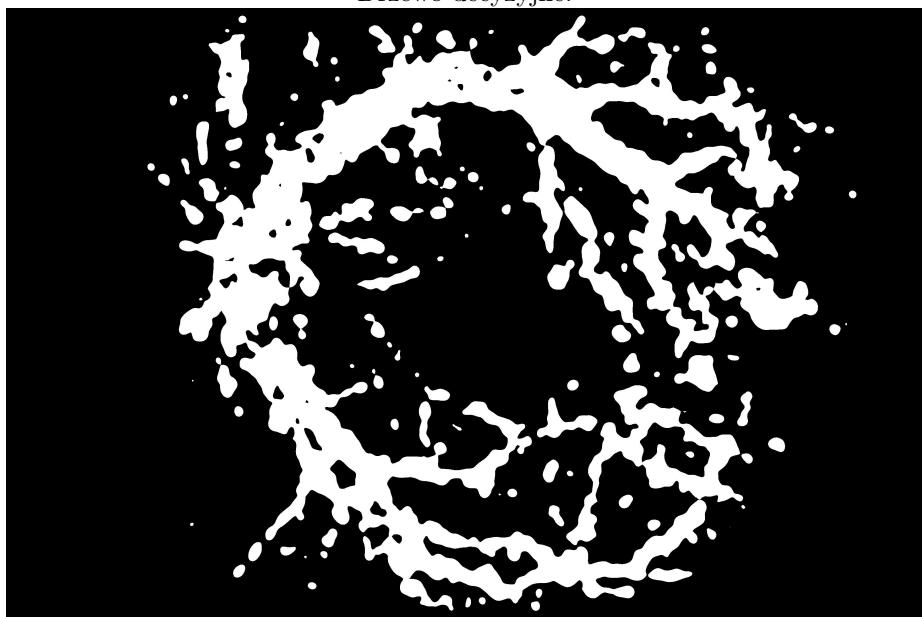
Meijering:



progowanie Sauvola:

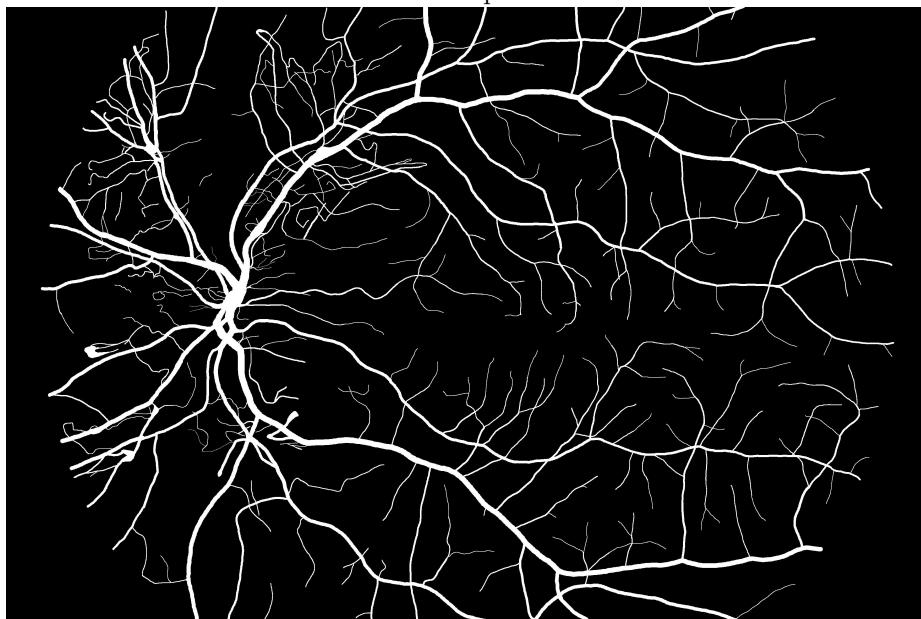


Drzewo decyzyjne:

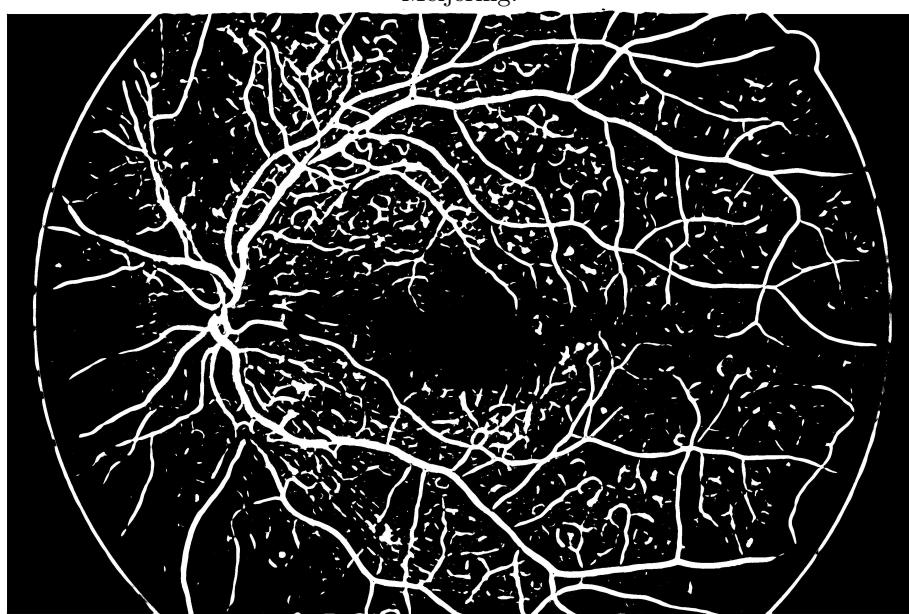


4.3 Obraz 3

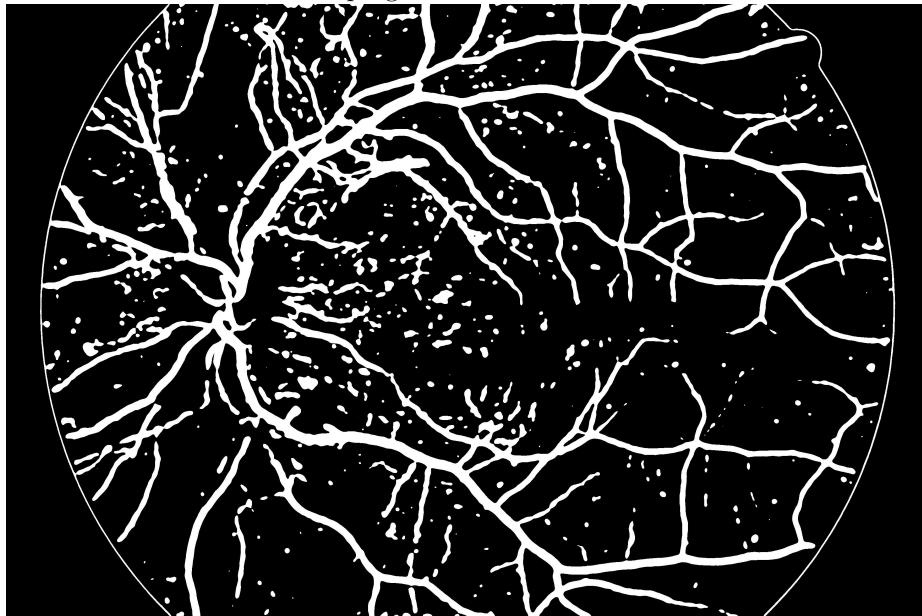
Maska ekspercka:



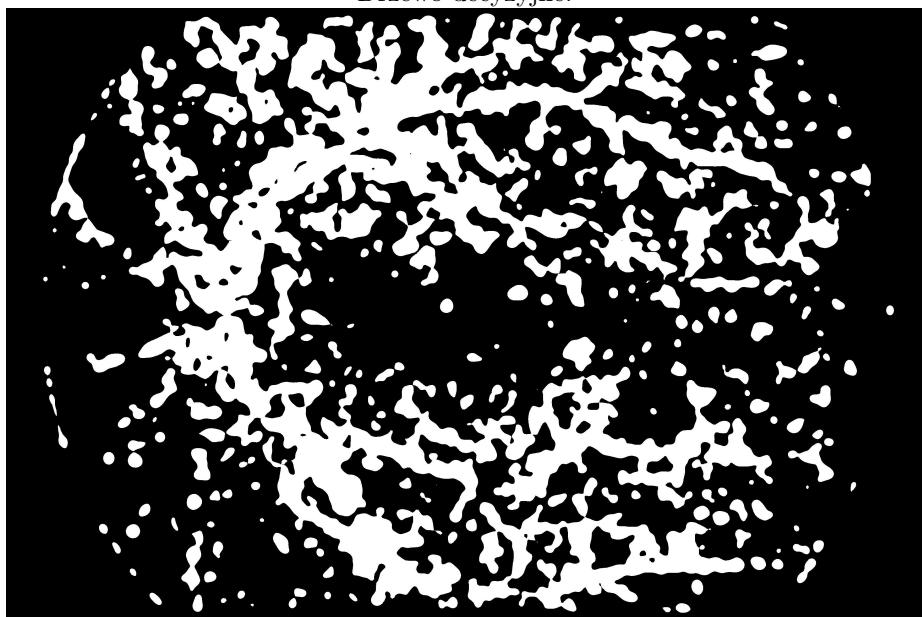
Meijering:



progowanie Sauvola:

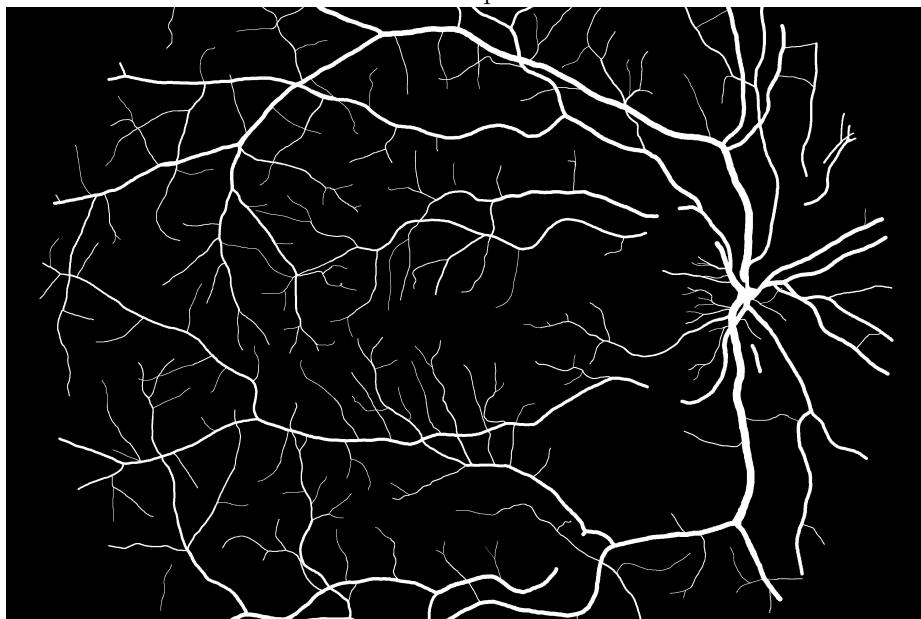


Drzewo decyzyjne:

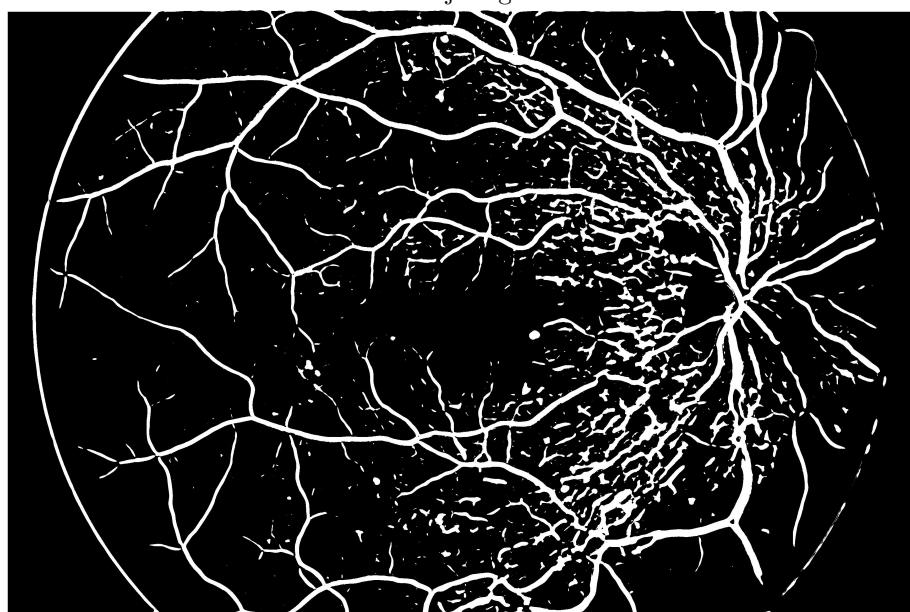


4.4 Obraz 4

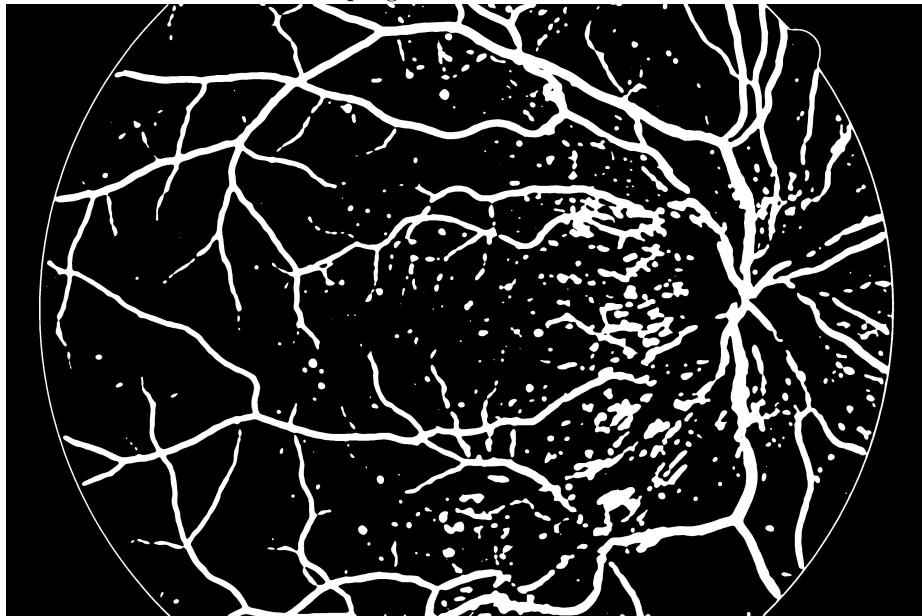
Maska ekspercka:



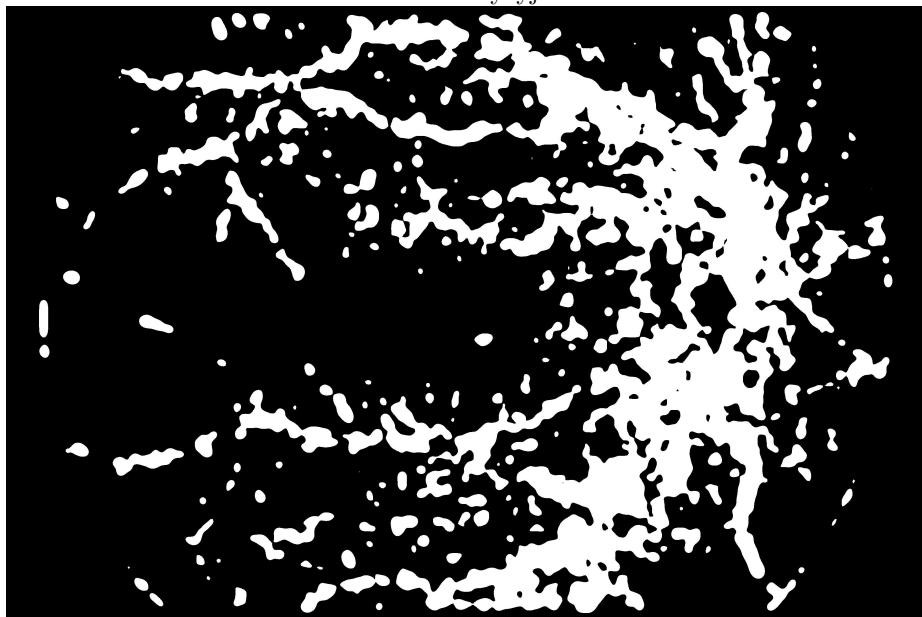
Meijering:



progowanie Sauvola:

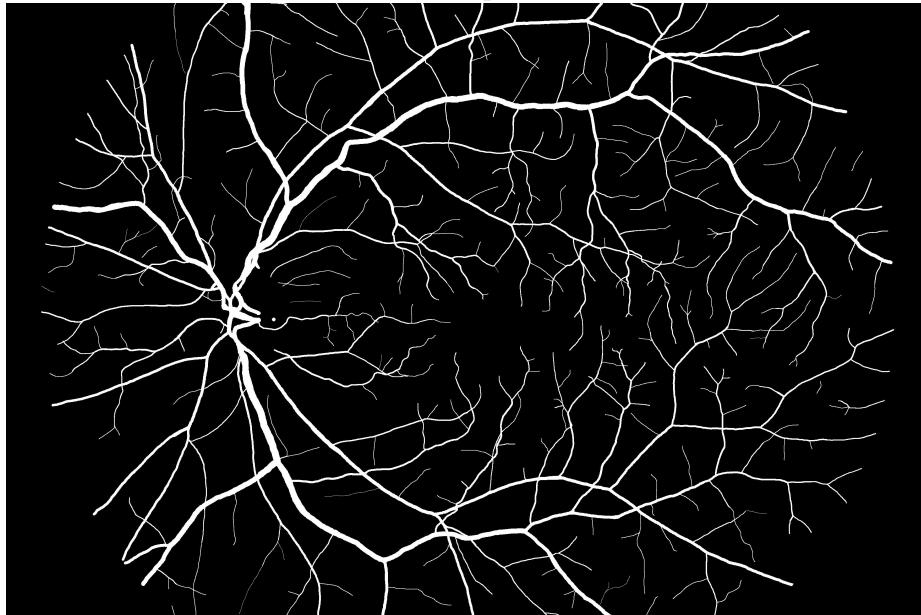


Drzewo decyzyjne:

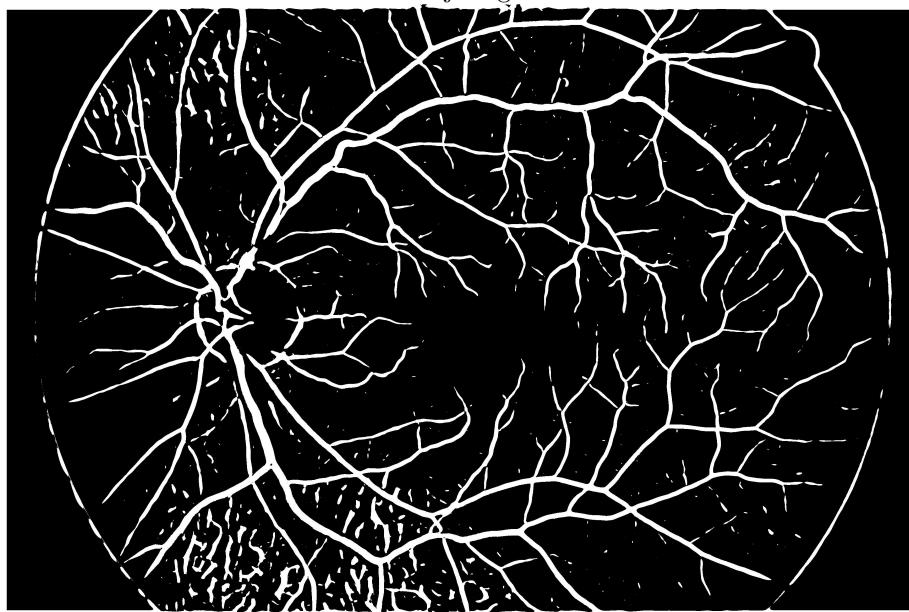


4.5 Obraz 5

Maska ekspercka:



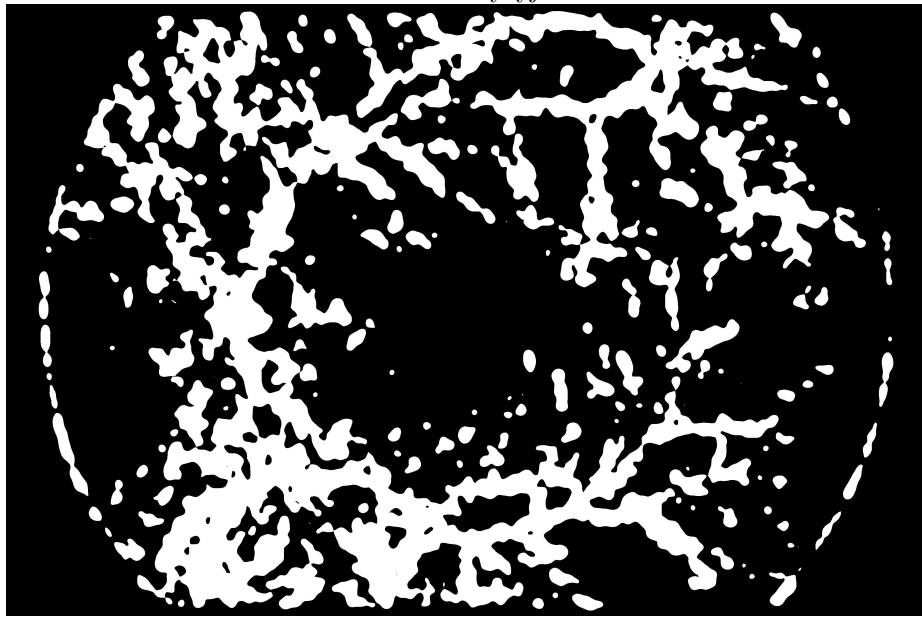
Meijering:



progowanie Sauvola:



Drzewo decyzyjne:



5 Analiza wyników

5.1 Obraz 1

	trafność	swoistość	czułość	średnia z swoistości i czułości
Meijering	89.791351%	90.563831%	80.627602%	85.595716%
Progowanie Sauvola	90.647296%	91.024317%	86.174785%	88.599551%
Drzewo decyzyjne	79.099742%	81.170687%	54.532583%	67.851635%

Obraz wejściowy, jako jedyny z tych pięciu, jest bez większych skaz, mimo to i tak na obu podstawowych algorytmach widać pewne zakłócenia i to nie zawsze wynikające z częściowego wychwytywania mniejszych naczyń. Za to od razu jest widoczna porażka drzewa decyzyjnego, pomimo prawie 70% średniej ze swoistości i czułości, uzyskany wynik jedynie delikatnie zarysuje to jak układają się naczynia.

5.2 Obraz 2

	trafność	swoistość	czułość	średnia z swoistości i czułości
Meijering	88.279234%	89.017541%	78.162428%	83.589985%
Progowanie Sauvola	91.143304%	91.993008%	79.500075%	85.746542%
Drzewo decyzyjne	80.905530%	82.541350%	58.490383%	70.515866%

Na tym przykładzie dobrze widać, że pomimo sporych zakłóceń po prawej stronie obrazu, pierwszy algorytm całkiem dobrze poradził sobie ze znalezieniem naczyń, choć wyświetla nadal sporo zakłóceń. Za to drugi lepiej odfiltrował zakłócenia, co dobrze obrazuje prawie o 3 punkty procentowe większa swoistość. Drzewo decyzyjne nadal nie wygląda dobrze.

5.3 Obraz 3

	trafność	swoistość	czułość	średnia z swoistości i czułości
Meijering	87.292592%	88.033251%	78.332181%	83.182716%
Progowanie Sauvola	89.714225%	90.319027%	82.397397%	86.358212%
Drzewo decyzyjne	73.750425%	74.925017%	59.540336%	67.232677%

Analogiczna sytuacja jak poprzednio, pierwszy algorytm wykrył naczynia, ale nie odfiltrował reszty, przez co ponownie progowanie uzyskało lepsze wyniki.

5.4 Obraz 4

	trafność	swoistość	czułość	średnia z swoistości i czułości
Meijering	88.744932%	89.42168%	79.068411%	84.245047%
Progowanie Sauvola	89.564751%	89.927520%	84.377704%	87.152612%
Drzewo decyzyjne	78.145964%	79.505381%	58.708365%	69.106873%

Ten przykład jest kolejnym na którym progowanie zdecydowanie lepiej radzi sobie blisko krawędzi oka. W połączeniu z minimalnie lepszym odfiltrowaniem artefaktów zyskuje aż 5 punktów procentowych nad poprzednią metodą.

5.5 Obraz 5

	trafność	swoistość	czułość	średnia z swoistości i czułości
Meijering	90.623436%	91.241117%	82.140478%	86.690797%
Progowanie Sauvola	91.759112%	92.559293%	80.76978%	86.664539%
Drzewo decyzyjne	75.61910%	76.63167%	61.712908%	69.172292%

Spoglądając na średnią to jedyny przypadek, gdy pierwszy algorytm triumfuje. Przy nie dużym szumie, lepiej wykrywa mniejsze naczynka.

6 Wnioski

Gdyby przyjąć, że badane oko będzie zdrowe oraz otrzymamy zdjęcie dobrej jakości to metoda wykorzystująca filtr meijering może się okazać dokładniejsza, lecz w bardziej ogólnym przypadku ustępuje ona progowaniu.

Za to liczyłem, że drzewo decyzyjne będzie zdecydowanie lepiej radzić sobie z rozróżnianiem naczyń od tła, a nastąpiła totalna porażka. Być może użyłem zbyt małej bazy danych lub problem tkwi w tym, że użyłem danych ze wszystkich podanych linków. Lub też popełniłem inny karygodny błąd, którego nie zauważam, przez moją niezbyt obszerną wiedzę na temat uczenia maszynowego.

7 Informacje dodatkowe

GUI uruchamiane poprzez komendę **streamlit run gui.init.py** w głównej gałęzi repozytorium. Skrypt ten znajduje się w folderze src.