

Retinal Vessel Segmentation preprocessing, Frangi, kNN, RF, U-Net

Alicja Augustyniak

Spis treści

1	Treść projektu	3
2	Niezbędne biblioteki	3
3	Przetworzenie obrazu	4
3.1	Importy bibliotek	4
3.2	Funkcje pomocnicze	5
3.2.1	Funkcja <code>load_ppm_gz(path_to_gz)</code>	5
3.2.2	Funkcja <code>apply_preprocessing(img_rgb)</code>	5
3.2.3	Funkcja <code>create_field_of_view_mask(gray)</code>	5
3.2.4	Funkcja <code>postprocess_mask(bin_mask)</code>	6
3.2.5	Funkcja <code>get_metrics(gt, pred)</code>	6
3.3	Główna pętla	6
3.4	Wizualizacje	7
3.4.1	Wizualizacja dla <code>top3</code>	7
3.4.2	Wizualizacja dla <code>bad2</code>	7
3.5	Zbiorcze metryki	7
3.6	Uzyskane wyniki	8
4	Klasyfikator k-NN	8
4.1	Importy bibliotek	8
4.2	Funkcje pomocnicze	9
4.2.1	Funkcja <code>load_ppm_gz(path_to_gz)</code>	9
4.2.2	Funkcja <code>apply_preprocessing(img_rgb)</code>	9
4.2.3	Funkcja <code>create_fov(gray)</code>	9
4.2.4	Funkcja <code>get_metrics(gt, pred)</code>	9
4.2.5	Funkcja <code>extract_features_from_patch(patch)</code>	9
4.2.6	Funkcja <code>extract_data(gray, mask, patch_size=5, stride=1)</code>	10
4.2.7	Funkcja <code>postprocess_mask(mask_bin)</code>	10
4.3	Trening klasyfikatora K-NN	10
4.4	Automatyczny dobór progu decyzyjnego	11
4.5	Segmentacja i ocena jednego obrazu	12
4.6	Ocena i wizualizacja	12
5	Klasyfikator Random Forest	12
5.1	Trening klasyfikatora Random Forest	12
5.2	Segmentacja i ocena jednego obrazu	13
5.3	Ocena i wizualizacja	13
5.4	Podsumowanie i różnice k-NN a Random Forest	14
6	Sieć neuronowa działająca na patchach	14
6.1	Importy Bibliotek	14
6.2	Funkcje Pomocnicze	14
6.2.1	Funkcja <code>extract_patches(gray, mask, patch_size=5, stride=1)</code>	14
6.2.2	Funkcja <code>build_patch_cnn(patch_size=5)</code>	14
6.3	Przygotowanie danych	15
6.4	Trening modelu CNN	15
6.5	Automatyczny dobór progu decyzyjnego	15
6.6	Segmentacja i ocena jednego obrazu	15
6.7	Ocena i wizualizacja	15
7	Sieć neuronowa U-Net	16
7.1	Proces ładowania i preprocessingu	16
7.2	Parametry dodatkowe	16
7.3	Podział danych	17
7.4	Architektura modelu U-Net	17
7.5	Trening modelu	18
7.6	Ewaluacja	19
7.6.1	Dobór progu	19
7.6.2	Testowanie progu na zbiorze testowym i obliczanie metryk	19

7.7	Wizualizacja wyników	19
7.8	Trudności, testowane podejścia, wnioski	22

1 Treść projektu

Przygotowanie aplikacji, która dla zadanego obrazu wejściowego przedstawiającego dno siatkówki oka automatycznie wykrywa naczynia krwionośne.

2 Niezbędne biblioteki

Na początku skryptu importowane są niezbędne biblioteki Python, które zapewniają funkcjonalność w zakresie operacji na plikach, kompresji danych, przetwarzania obrazów, obliczeń numerycznych, wizualizacji, analizy danych i metryk uczenia maszynowego:

- `os`: do interakcji z systemem plików, umożliwia operacje takie jak łączenie ścieżek (`os.path.join`) czy listowanie plików w katalogu (`os.listdir`);
- `gzip`: do obsługi skompresowanych plików `.gz`, co pozwala na wczytywanie obrazów i masek zapisanych w tym formacie;
- `cv2` (OpenCV): podstawowa biblioteka do przetwarzania obrazów, wykorzystywana do wczytywania obrazów (`cv2.imread`), konwersji przestrzeni barw (`cv2.cvtColor`), operacji morfologicznych (`cv2.morphologyEx`, `cv2.getStructuringElement`), progowania (`cv2.threshold`) oraz zmiany rozmiaru obrazów (`cv2.resize`);
- `numpy` (`np`): do obliczeń numerycznych i manipulacji tablicami;
- `matplotlib.pyplot` (`plt`): do tworzenia wykresów i wizualizacji, używane do wyświetlania obrazów oryginalnych, masek referencyjnych i predykcji oraz nakładek;
- `skimage.filters.frangi`: implementacja filtra Frangiego z biblioteki `scikit-image`;
- `skimage.measure`: moduł z biblioteki `scikit-image`, zawierający funkcje do mierzenia właściwości regionów obrazu;
 - `moments_central`: do obliczania momentów centralnych obrazu;
 - `moments_hu`: do obliczania momentów Hu, zestawu siedmiu niezmienników momentowych, które są niezmiennicze na przesunięcie, skalowanie i rotację, co czyni je użytecznymi jako cechy kształtu;
- `sklearn.preprocessing.StandardScaler`: z biblioteki `scikit-learn`, używany do standardyzacji cech, co oznacza skalowanie danych do średniej równej 0 i odchylenia standardowego równego 1;
- `sklearn.pipeline.make_pipeline`: z biblioteki `scikit-learn`, służy do tworzenia potoków uczenia maszynowego, co pozwala na sekwencyjne łączenie kroków przetwarzania danych i modelowania w jeden obiekt;
- `sklearn.neighbors.KNeighborsClassifier`: z biblioteki `scikit-learn`, implementuje algorytm klasyfikatora K-Najbliższych Sąsiadów (K-NN), prosty algorytm klasyfikacji oparty na odległości do sąsiadów;
- `sklearn.ensemble.RandomForestClassifier`: z biblioteki `scikit-learn`, implementuje algorytm losowego lasu do klasyfikacji, będący metodą zespołową budującą wiele drzew decyzyjnych w celu poprawy dokładności i odporności na przeuczenie;
- `sklearn.metrics`: moduł z biblioteki `scikit-learn`, zawiera funkcje do obliczania metryk oceny klasyfikacji, takich jak:
 - `confusion_matrix`: do generowania macierzy pomyłek,
 - `accuracy_score`: do obliczania dokładności,
 - `recall_score`: do obliczania czułości (recall),
 - `roc_curve`: do obliczania punktów krzywej ROC;
- `sklearn.model_selection.train_test_split`: z biblioteki `scikit-learn`, do podziału danych na zbiory treningowy i testowy;
- `pandas` (`pd`): do analizy i manipulacji danymi tabelarycznymi, używany do tworzenia podsumowań wyników w formie DataFrame;
- `matplotlib.patches` (`mpatches`): do tworzenia niestandardowych elementów legendy na wykresach Matplotlib;

- `tensorflow`: podstawowa biblioteka do uczenia maszynowego, zapewniająca fundamenty dla tworzenia i trenowania modeli sieci neuronowych;
- `tensorflow.keras.models.Model`: z Keras (API w TensorFlow), do definiowania modeli sieci neuronowych, w tym modeli funkcyjnych;
- `tensorflow.keras.layers`: moduł z Keras, zawierający definicje warstw sieci neuronowych:
 - `Input`: do definiowania warstwy wejściowej modelu,
 - `Conv2D`: warstwa konwolucyjna 2D, używana do ekstrakcji cech z obrazów,
 - `MaxPooling2D`: warstwa poolingu 2D, zmniejsza wymiary przestrzenne, redukując liczbę parametrów i kontrolując przeuczenie,
 - `Flatten`: przekształca wejście wielowymiarowe w jednowymiarowy wektor,
 - `Dense`: warstwa gęsta (w pełni połączona), gdzie każdy neuron jest połączony z każdym neuronem z poprzedniej warstwy,
 - `Dropout`: warstwa regularyzacji, losowo "wyłącza" neurony podczas treningu, aby zapobiec przeuczeniu,
 - `Conv2DTranspose`: warstwa transponowanej konwolucji (często nazywana "dekonwolucją"), używana do upsamplingu w modelach takich jak U-Net,
 - `concatenate`: do łączenia wyjść z różnych warstw wzduż określonej osi (np. w architekturze U-Net),
- `tensorflow.keras.optimizers.Adam`: optymalizator Adam, jeden z najpopularniejszych algorytmów optymalizacji gradientowej do trenowania sieci neuronowych;
- `tensorflow.keras.callbacks`: moduł z Keras, zawierający narzędzia do kontrolowania procesu treningu:
 - `EarlyStopping`: zatrzymuje trening, gdy metryka monitorowania przestaje się poprawiać, aby zapobiec przeuczeniu,
 - `ModelCheckpoint`: zapisuje najlepszy model (na podstawie monitorowanej metryki) podczas treningu,
 - `ReduceLROnPlateau`: zmniejsza szybkość uczenia, gdy metryka monitorowania przestaje się poprawiać, co pomaga w osiągnięciu lepszej konwergencji;
- `tensorflow.keras.utils.to_categorical`: do konwersji etykiet klas na format one-hot encoding, niezbędny dla klasyfikacji wieloklasowej w sieciach neuronowych.

3 Przetworzenie obrazu

Celem jest segmentacja naczyń krwionośnych w obrazach dna oka, podejście wykorzystuje klasyczne techniki przetwarzania obrazów do wstępnego przetwarzania, detekcji cech (naczyń) za pomocą filtra Frangiego, post-processingu, a także standardowe metryki do oceny jakości segmentacji. Cały proces jest zautomatyzowany i przystosowany do efektywnego przetwarzania wielu obrazów, włączając w to wizualizację wyników dla wybranych przypadków oraz zbiorcze podsumowanie.

3.1 Importy bibliotek

Na początku skryptu importowane są niezbędne biblioteki Python, które zapewniają funkcjonalność w zakresie operacji na plikach, kompresji danych, przetwarzania obrazów, obliczeń numerycznych, wizualizacji, analizy danych i metryk uczenia maszynowego:

- `os`,
- `gzip`,
- `cv2` (OpenCV),
- `numpy` (`np`),
- `matplotlib.pyplot` (`plt`),
- `skimage.filters.frangi`,
- `scikit-image`,

- `sklearn.metrics`,
- `pandas (pd)`,
- `matplotlib.patches (mpatches)`,

3.2 Funkcje pomocnicze

W kodzie zdefiniowano zmienne przechowujące ścieżki do katalogów z danymi.

3.2.1 Funkcja `load_ppm_gz(path_to_gz)`

```
1 def load_ppm_gz(path_to_gz):
2     with gzip.open(path_to_gz, 'rb') as f:
3         buf = f.read()
4     arr = np.frombuffer(buf, dtype=np.uint8)
5     return cv2.imdecode(arr, cv2.IMREAD_UNCHANGED)
```

Listing 1: Funkcja wczytująca skompresowane obrazy

Odpowiada za wczytywanie obrazów skompresowanych w formacie .ppm.gz. Funkcja otwiera plik gzip w trybie binarnym, odczytuje jego zawartość, konwertuje ją na tablicę bajtów NumPy, a następnie używa funkcji `cv2.imdecode` do zdekodowania tej tablicy na obraz. Argument `cv2.IMREAD_UNCHANGED` zapewnia, że obraz jest wczytywany w swoim oryginalnym formacie.

3.2.2 Funkcja `apply_preprocessing(img_rgb)`

```
1 def apply_preprocessing(img_rgb):
2     gray = cv2.cvtColor(img_rgb, cv2.COLOR_RGB2GRAY)
3     clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
4     return clahe.apply(gray)
```

Listing 2: Funkcja wstępniego przetwarzania obrazu

Wykonuje wstępne przetwarzanie obrazu wejściowego `img_rgb`. Konwertuje obraz z przestrzeni barw **RGB na skalę szarości** (`cv2.COLOR_RGB2GRAY`). Stosuje **CLAHE** i poprawia kontrast w obrazach. Parametry `clipLimit=2.0` (ogranicza wzmocnienie kontrastu, aby zapobiec szumowi) i `tileGridSize=(8,8)` (określa rozmiar siatki, na której wykonywana jest lokalna equalizacja histogramu) są typowymi wartościami dla tego zastosowania.

3.2.3 Funkcja `create_field_of_view_mask(gray)`

```
1 def create_field_of_view_mask(gray):
2     _, th = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
3     kern = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15,15))
4     close = cv2.morphologyEx(th, cv2.MORPH_CLOSE, kern)
5     open_ = cv2.morphologyEx(close, cv2.MORPH_OPEN, kern)
6     return (open_ > 0).astype(np.uint8)
```

Listing 3: Funkcja tworząca maskę pola widzenia (FOV)

W użytej bazie nie była dodatkona maska FOV, dlatego załączona funkcja generuje binarną maskę pola widzenia, która identyfikuje obszar oka na obrazie:

- wykonuje **progowanie Otsu** (`cv2.THRESH_OTSU`) na obrazie w skali szarości, metoda Otsu automatycznie znajduje optymalny próg do binarnej segmentacji;
- wykonuje operacje **morfologiczne**:
 - **zamknięcie (MORPH_CLOSE)**: operacja dylatacji, a następnie erozji, pomaga wypełnić małe dziury i połączyć bliskie sobie obiekty;
 - **otwarcie (MORPH_OPEN)**: operacja erozji, a następnie dylatacji, pomaga usunąć małe obiekty i wygładzić granice;
 - użycie dużego jądra eliptycznego (15,15) sprawia, że te operacje są agresywne i mają na celu wyizolowanie głównego, dużego obszaru oka;
- konwertuje wynik do binarnej maski (wartości 0 lub 1) typu `np.uint8`, maska ta służy do ograniczenia dalszego przetwarzania tylko do obszaru oka.

3.2.4 Funkcja postprocess_mask(bin_mask)

```
1 def postprocess_mask(bin_mask):
2     kern = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3,3))
3     close = cv2.morphologyEx(bin_mask, cv2.MORPH_CLOSE, kern)
4     open_ = cv2.morphologyEx(close, cv2.MORPH_OPEN, kern)
5     return open_
```

Listing 4: Funkcja post-processingu maski segmentacji

Wygładza i oczyszcza binarną maskę segmentacji naczyń, stosuje operacje **zamknięcia** (wypełnienie małych przerw w naczyniach) i **otwarcia** (usunięcie drobnych artefaktów/szumu) z mniejszym jądrem eliptycznym (3,3). Celem jest poprawa spójności i redukcja niepożądanych pikseli w końcowej masce naczyń.

3.2.5 Funkcja get_metrics(gt, pred)

```
1 def get_metrics(gt, pred):
2     y_true = gt.flatten()
3     y_pred = pred.flatten()
4     tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
5     acc = accuracy_score(y_true, y_pred)
6     sens = recall_score(y_true, y_pred)
7     spec = tn / (tn + fp) if (tn+fp)>0 else 0
8     return acc, sens, spec, (sens+spec)/2, (tn, fp, fn, tp)
```

Listing 5: Funkcja obliczająca metryki segmentacji

Oblicza standardowe metryki oceny jakości segmentacji, porównując maskę referencyjną (**gt**) z maską predykcji (**pred**):

- spłaszcza maski do jednowymiarowych wektorów dla ułatwienia obliczeń,
- oblicza absolutnie podstawowe metryki do oceny skuteczności:
 - **macierz pomyłek** (`confusion_matrix`), zwracającą liczbę TN, FP, FN i TP,
 - **accuracy (dokładność)**: odsetek poprawnie sklasyfikowanych pikseli,
 - **sensitivity (czułość/recall)**: zdolność modelu do znajdowania wszystkich pozytywnych przypadków (ile prawdziwych naczyń zostało wykrytych),
 - **specificity (swoistość)**: zdolność modelu do poprawnego identyfikowania negatywnych przypadków (ile obszarów tła zostało poprawnie zidentyfikowanych jako tło),
 - **balanced score (zbalansowany wynik)**: średnia arytmetyczna czułości i swoistości, dobra miara, gdy klasy są niezbalansowane,
- funkcja zwraca te metryki oraz samą macierz pomyłek.

3.3 Główna pętla

Dla każdego pliku obrazu w katalogu `star_images`, pętla wykonuje poniższe kroki.

Wczytanie obrazu i masek GT

Wczytuje oryginalny obraz i maski referencyjne (Ground Truth) od ekspertów (AH i jeśli jest to VK). Jeśli maska VK istnieje, jest ona łączona z maską AH (logiczne OR), tworząc jedną maskę GT, gdzie piksel jest uznany za naczynie, jeśli którykolwiek z ekspertów tak go oznaczył.

Wstępne przetwarzanie obrazu

Obraz jest konwertowany do formatu RGB (jeśli nie jest już w tym formacie), następnie przetwarzany wstępnie (kontrast CLAHE) i maskowany za pomocą maski pola widzenia (FOV).

Detekcja naczyń (Filtr Frangiego)

Filtr Frangiego jest stosowany do obrazu w skali szarości (przekonwertowanego na wartości zmiennoprzecinkowe w zakresie 0-1). Filtr ten wzmacnia struktury, takie jak naczynia, dając mapę prawdopodobieństwa. Następnie ta mapa prawdopodobieństwa jest progowana wartością 0.03, tworząc wstępnią binarną maskę naczyń.

Post-processing i maskowanie FOV

Wstępna binarna maska jest poddawana post-processingowi (`postprocess_mask`) w celu usunięcia szumu i wypełnienia przerw, a następnie mnożona przez maskę FOV, aby usunąć segmentację poza obszarem oka.

Metryki

Dla każdego obrazu obliczane są metryki i macierz pomyłek za pomocą funkcji `get_metrics`. Wyniki są dodawane do listy `all_results` oraz sumowane do `total_cm` w celu uzyskania zbiorczych statystyk.

3.4 Wizualizacje

Po przetworzeniu wszystkich obrazów, kod wybiera i wizualizuje wyniki dla 3 najlepszych (`top3`) oraz 2 najgorszych (`bad2`) obrazów, bazując na wartości `balanced_score`.

3.4.1 Wizualizacja dla top3

```
1 for res in top3:
2     # (...) wczytania i przetwarzania
3     tn_mask = (gt_mask==0) & (vessel_bin_masked==0)
4     fp_mask = (gt_mask==0) & (vessel_bin_masked==1)
5     fn_mask = (gt_mask==1) & (vessel_bin_masked==0)
6     tp_mask = (gt_mask==1) & (vessel_bin_masked==1)
7
8     overlay = img_rgb.copy()
9     overlay[fp_mask] = [255,255,0]
10    overlay[fn_mask] = [0,0,255]
11    overlay[tp_mask] = [0,255,0]
12
13    magenta_patch = mpatches.Patch(color='yellow', label='FP')
14    blue_patch = mpatches.Patch(color='blue', label='FN')
15    green_patch = mpatches.Patch(color='green', label='TP')
16
17    fig, axs = plt.subplots(1,4,figsize=(20,5))
18    axs[0].imshow(img_rgb); axs[0].set_title('orygina'); axs[0].axis('off')
19    axs[1].imshow(gt_mask, cmap='gray'); axs[1].set_title('maska ekspercka'); axs[1].axis('off')
20
21    axs[2].imshow(overlay); axs[2].set_title('wynik: overlay TP/FP/FN'); axs[2].axis('off')
22    axs[2].legend(handles=[green_patch, magenta_patch, blue_patch], loc='lower right')
23    axs[3].imshow(vessel_bin_masked, cmap='gray'); axs[3].set_title('wynik: maska seg.'); axs[3].axis('off')
24
25    plt.suptitle(f"TN:{tn} FP:{fp} FN:{fn} TP:{tp} | acc:{acc:.3f} sens:{sens:.3f} spec:{spec:.3f} bal:{bal:.3f}", fontsize=12, y=1.02)
    plt.tight_layout()
    plt.show()
```

Listing 6: Fragment kodu wizualizacji dla najlepszych wyników

Dla każdego z 3 obrazów o najwyższej `balanced_score`:

- obraz jest ponownie wczytywany i przetwarzany, aby uzyskać maski True Positives (TP), False Positives (FP), False Negatives (FN) i True Negatives (TN) poprzez logiczne porównanie maski GT i predykcji;
- tworzona jest wizualizacja `overlay`: oryginalny obraz jest modyfikowany tak, aby:
 - **FP** są pokazywane na **żółto** ([255,255,0]),
 - **FN** są pokazywane na **niebiesko** ([0,0,255]),
 - **TP** są pokazywane na **zielono** ([0,255,0]);
- wyświetlane są cztery panele: oryginalny obraz, maska ekspercka (GT), obraz z nałożonym overlayem TP/FP/FN oraz uzyskana binarna maska segmentacji;
- nad wykresem wyświetlane są również wartości metryk i macierzy pomyłek dla danego obrazu, co ułatwia ocenę wizualną.

3.4.2 Wizualizacja dla bad2

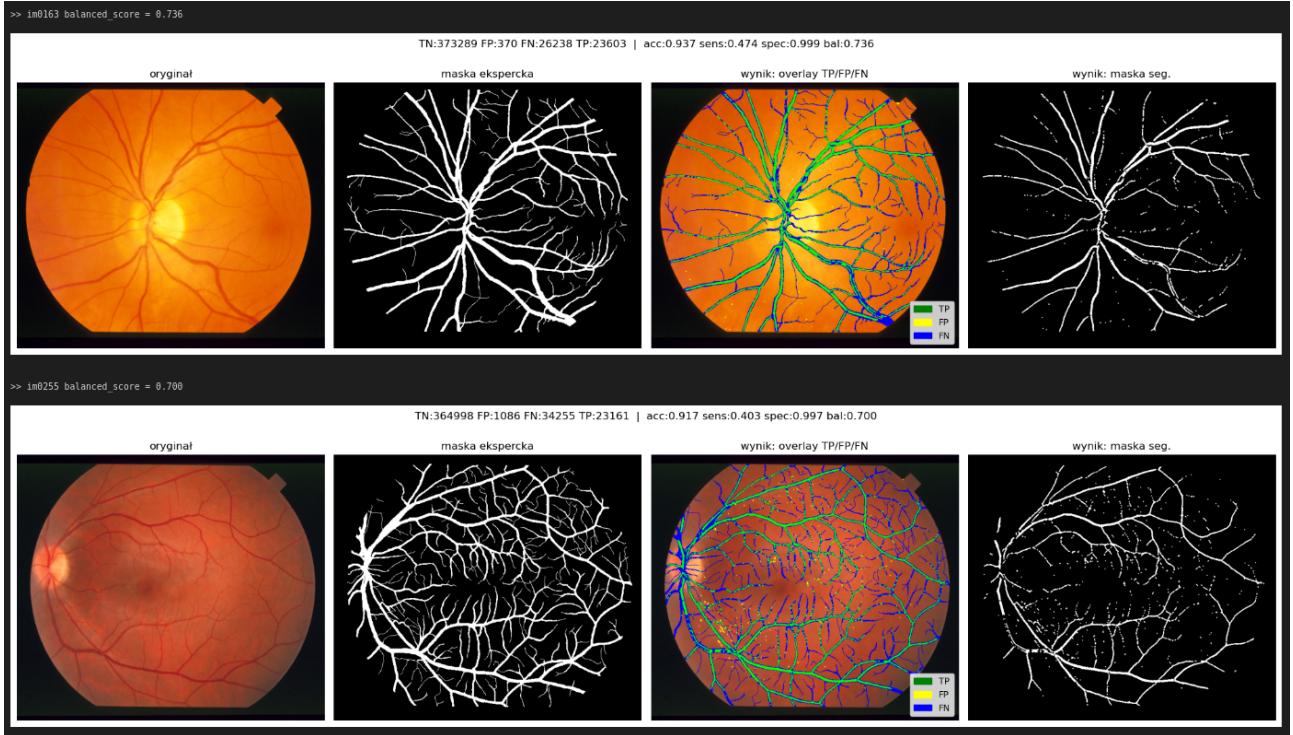
Sekcja dla `bad2` jest analogiczna do `top3`.

3.5 Zbiorcze metryki

Na koniec, kod generuje zbiorcze podsumowanie wyników dla całego zestawu danych:

- tworzony jest DataFrame Pandas z wynikami dla wszystkich obrazów, co pozwala na łatwą analizę statystyczną;
- wyświetlane są **statystyki opisowe** (średnia, odchylenie standardowe, minimum, maksimum) dla `accuracy`, `sensitivity`, `specificity` i `balanced_score` dla całego zbioru danych;
- obliczana i wyświetlana jest **łączna macierz pomyłek** dla wszystkich pikseli ze wszystkich obrazów;
- na podstawie łącznej macierzy pomyłek wyliczane są i wyświetlane globalne wartości `accuracy`, `sensitivity`, `specificity` i `balanced score` dla całego zbioru.

3.6 Uzyskane wyniki



Rysunek 1: Wizualizacja segmentacji dla dwóch obrazów o najwyższym wyniku (balanced score). Od lewej: oryginalny obraz, maska ekspercka (ground truth), nakładka z wynikami segmentacji (zielony - TP, żółty - FP, niebieski - FN) oraz uzyskana binarna maska segmentacji.

Tabela 1: Statystyki dla wszystkich obrazów

	Accuracy	Sensitivity	Specificity	Balanced Score
Mean	0.910214	0.233222	0.997268	0.615245
Std	0.018406	0.124003	0.004610	0.061634
Min	0.872357	0.037384	0.978887	0.518585
Max	0.941216	0.473566	0.999884	0.736288

Tabela 2: Łączna macierz pomyłek i zbiorcze metryki

Parametr	Wartość
Łączna macierz pomyłek	
True Negatives (TN)	7464124
False Positives (FP)	20508
False Negatives (FN)	739983
True Positives (TP)	245385
Zbiorcze metryki	
Accuracy	0.9102
Sensitivity	0.2490
Specificity	0.9973
Balanced Score	0.6231

4 Klasyfikator k-NN

4.1 Importy bibliotek

Kod dodatkowo wykorzystuje następujące biblioteki:

- `skimage.measure.moments_central`, `moments_hu`,
- `sklearn.preprocessing.StandardScaler`,
- `sklearn.pipeline.make_pipeline`,
- `sklearn.neighbors.KNeighborsClassifier`,
- `sklearn.model_selection.train_test_split`.

Pliki obrazów są dzielone w prosty sposób: pierwsze 10 plików służy do treningu, a reszta do testowania. Jest to prosty podział, z powodu bardzo małej liczby danych.

4.2 Funkcje pomocnicze

4.2.1 Funkcja `load_ppm_gz(path_to_gz)`

Funkcja ta została szczegółowo opisana w poprzednich sekcjach sprawozdania.

4.2.2 Funkcja `apply_preprocessing(img_rgb)`

Funkcja ta została szczegółowo opisana w poprzednich sekcjach sprawozdania. Wykonuje konwersję do skali szarości i aplikuje CLAHE.

4.2.3 Funkcja `create_fov(gray)`

```

1 def create_fov(gray):
2     _, t = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU + cv2.THRESH_BINARY)
3     k      = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (15,15))
4     m      = cv2.morphologyEx(t, cv2.MORPH_CLOSE, k)
5     m      = cv2.morphologyEx(m, cv2.MORPH_OPEN, k)
6     cnts, _ = cv2.findContours(m, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
7     if not cnts:
8         return (m > 0).astype(np.uint8)
9     cnt = max(cnts, key=cv2.contourArea)
10    mask = np.zeros_like(gray, dtype=np.uint8)
11    cv2.drawContours(mask, [cnt], -1, 255, thickness=-1)
12    return (mask > 0).astype(np.uint8)

```

Listing 7: Funkcja tworząca maskę pola widzenia (FOV)

Ta funkcja została zmodyfikowana w porównaniu do poprzednich wersji. Nadal wykorzystuje progowanie Otsu i operacje morfologiczne (zamknięcie, otwarcie) z dużym jądrelem, ale dodatkowo:

- wykrywa **zewnętrzne kontury** na otrzymanej binarnej masce;
- wybiera **największy kontur**, zakładając, że reprezentuje on pole widzenia siatkówki;
- tworzy nową, pustą maskę i **wypełnia największy kontur**.

4.2.4 Funkcja `get_metrics(gt, pred)`

Funkcja ta została szczegółowo opisana w poprzednich sekcjach sprawozdania. Oblicza metryki `accuracy`, `sensitivity`, `specificity` oraz macierz pomyłek. W tej wersji dodano `zero_division=0` do `recall_score`, aby uniknąć błędów w przypadku braku prawdziwych pozytywów.

4.2.5 Funkcja `extract_features_from_patch(patch)`

```

1 def extract_features_from_patch(patch):
2     p = patch.astype(np.float64)
3     v = np.var(p)
4     mean_intensity = np.mean(p)
5     gx = cv2.Sobel(p, cv2.CV_64F, 1, 0, ksize=3)
6     gy = cv2.Sobel(p, cv2.CV_64F, 0, 1, ksize=3)
7     grad_mean = np.mean(np.sqrt(gx**2 + gy**2))
8     if v == 0.0:
9         hu = [0.0]*7
10    else:
11        mc = moments_central(p)
12        hu = moments_hu(mc)
13        hu = [0.0 if np.isnan(x) or np.isinf(x) else x for x in hu]

```

```
14     return [v, mean_intensity, grad_mean] + hu
```

Listing 8: Funkcja ekstrakcji cech z fragmentu obrazu (patch)

Jest to kluczowa, nowa funkcja w tym podejściu, ekstrahuje zestaw cech z danego fragmentu obrazu (`patch`) o rozmiarze `patch_size` (domyślnie 5x5 pikseli):

- **wariancja intensywności** mierzy rozproszenie pikseli w patchu. Obszary z naczyniami mogą mieć większą wariancję na granicy naczynia/tła;
- **średnia intensywność** czyli średnia wartość pikseli w patchu;
- **średnia wielkość gradientu** obliczana za pomocą filtrów Sobela w kierunku X i Y, a następnie średnia wielkość wektora gradientu. Duże wartości gradientu wskazują na krawędzię (w tym naczynia);
- **momenty Hu** czyli zestaw siedmiu niezmienników momentowych, które są niezmiennicze na przesunięcie, skalowanie i rotację, kod zawiera obsługę przypadku, gdy wariancja jest zerowa, co mogłoby prowadzić do błędów w obliczeniach momentów.

Zwarcane cechy są łączone w jedną listę.

4.2.6 Funkcja `extract_data(gray, mask, patch_size=5, stride=1)`

```
1 def extract_data(gray, mask, patch_size=5, stride=1):
2     X, y, coords = [], [], []
3     off = patch_size // 2
4     h, w = gray.shape
5     for i in range(off, h-off, stride):
6         for j in range(off, w-off, stride):
7             if mask[i, j] not in (0, 1):
8                 continue
9             patch = gray[i-off:i+off+1, j-off:j+off+1]
10            feats = extract_features_from_patch(patch)
11            X.append(feats)
12            y.append(mask[i, j])
13            coords.append((i, j))
14    return np.array(X), np.array(y), coords
```

Listing 9: Funkcja do ekstrakcji danych (cech i etykiet) z obrazu

- Ta funkcja iteruje po obrazie w skali szarości i masce, ekstrahując dane do treningu/testowania klasyfikatora:
- dla każdego piksela w obszarze FOV (z pominięciem brzegów równych `patch_size // 2` i z uwzględnieniem `stride`):
 - wyodrębnia `patch` obrazu wokół tego piksela,
 - wywołuje `extract_features_from_patch` do ekstrakcji cech z tego patcha,
 - zapisuje wyekstrahowane cechy (`X`), etykietę piksela z maski GT (`y`) oraz współrzędne piksela (`coords`);
 - zwraca te dane jako tablice NumPy.

4.2.7 Funkcja `postprocess_mask(mask_bin)`

Funkcja ta została szczegółowo opisana w poprzednich sekcjach sprawozdania. Wygładza binarną maskę segmentacji poprzez operacje morfologiczne otwarcia i zamknięcia. W tej wersji użyto jądra 5x5.

4.3 Trening klasyfikatora K-NN

```
1 Xtr, ytr = [], []
2 for fn in train_f:
3     # kod wczytania, preprocessingu obrazu i maski
4     # kod ekstrakcji danych za pomocą extract_data
5     Xtr.extend(Xf)
6     ytr.extend(yf)
7
8 Xtr = np.nan_to_num(np.array(Xtr))
9 ytr = np.array(ytr)
10
11 # balansowanie klas
12 pos_idx = np.where(ytr == 1)[0]
13 neg_idx = np.where(ytr == 0)[0]
```

```

14 neg_sel = np.random.choice(neg_idx, size=len(pos_idx), replace=False)
15 sel = np.concatenate([pos_idx, neg_sel])
16 Xbal, ybal = Xtr[sel], ytr[sel]
17
18 # train/val i trenowanie k-NN
19 X_train, X_val, y_train, y_val = train_test_split(
20     Xbal, ybal, test_size=0.3, random_state=42, stratify=ybal
21 )
22
23 clf = make_pipeline(
24     StandardScaler(), # Standaryzacja cech
25     KNeighborsClassifier(n_neighbors=1, weights='distance')
26 )
27 clf.fit(X_train, y_train)

```

Listing 10: Trening i walidacja klasyfikatora K-NN

Ekstrakcja danych treningowych

Kod iteruje po plikach treningowych, wczytuje obrazy i maski GT, wykonuje wstępne przetwarzanie i ekstrakcję cech z patchów. Wszystkie cechy i etykiety są zbierane do list `Xtr` i `ytr`.

Obsługa NaN

Użyto `np.nan_to_num` do zamiany ewentualnych wartości ‘NaN’ lub ‘Inf’ w cechach na wartości liczbowe (np. 0 lub bardzo duże liczby), co zapobiega błędem w klasyfikatorze.

Balansowanie klas

Pikseli tła jest znacznie więcej niż pikseli naczyń (nierównowaga klas), wykonano **undersampling** klasy negatywnej (tła). Losowo wybiera się taką samą liczbę pikseli tła, ile jest pikseli naczyń, ma to na celu zapobieganie dominacji klasy tła w treningu, co mogłoby prowadzić do modelu o bardzo wysokiej swoistości, ale niskiej czułości.

Podział na treningowy i walidacyjny zbiór

Zbalansowany zbiór danych (`Xbal`, `ybal`) jest dzielony na podzbiory treningowy (`X_train`, `y_train`) i walidacyjny (`X_val`, `y_val`) z podziałem 70/30 (`test_size=0.3`). Parametr `stratify=ybal` zapewnia, że proporcje klas są zachowane w obu podzbiorach.

Budowa i trening klasyfikatora K-NN:

- tworzony jest **potok** (`make_pipeline`), który najpierw skaluje cechy za pomocą `StandardScaler`, a następnie aplikuje `KNeighborsClassifier`,
- `n_neighbors=1` klasyfikator szuka tylko 1 najbliższego sąsiada,
- `weights='distance'` waga sąsiadów jest odwrotnie proporcjonalna do ich odległości, co oznacza, że bliżsi sąsiedzi mają większy wpływ.

4.4 Automatyczny dobór progu decyzyjnego

```

1 probs_val = clf.predict_proba(X_val)[:,1]
2 fpr, tpr, thr = roc_curve(y_val, probs_val)
3 best_idx = np.argmax(tpr - fpr)
4 best_thr = thr[best_idx]
5 print(f"Wybrany próg z ROC (Youden's J): {best_thr:.3f}")

```

Listing 11: Dobór progu decyzyjnego z krzywej ROC

Po wytrenowaniu modelu, konieczne jest ustalenie optymalnego progu decyzyjnego dla klasyfikacji binarnej. Odbiera się to na zbiorze walidacyjnym:

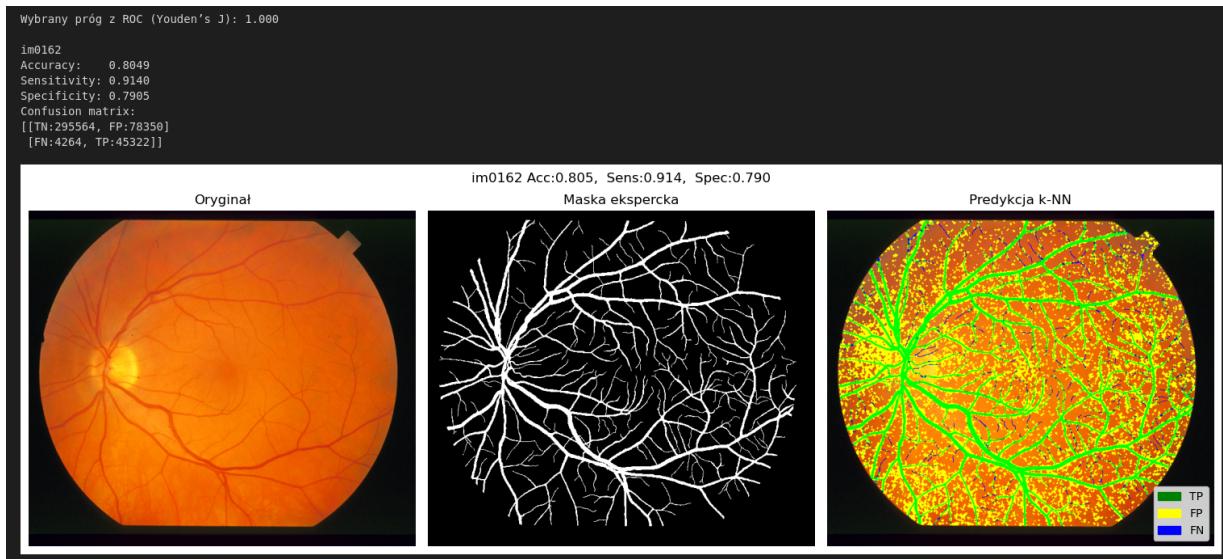
- klasyfikator przewiduje **prawdopodobieństwa** dla klasy pozytywnej (naczynia) na zbiorze walidacyjnym;
- obliczana jest **krzywa ROC**, która pokazuje zależność między czułością (True Positive Rate TPR) a swoistością (False Positive Rate FPR) dla różnych progów;
- wybierany jest próg, który maksymalizuje **indeks Youdena (Youden's J)**, zdefiniowany jako $J = \text{TPR} - \text{FPR}$, indeks Youdena szuka punktu na krzywej ROC, który jest najbardziej oddalony od linii losowego klasyfikatora i reprezentuje optymalny balans między czułością a swoistością;
- wybrany próg (`best_thr`) jest następnie wykorzystywany do binarnej klasyfikacji.

4.5 Segmentacja i ocena jednego obrazu

Obraz testowy i jego maska GT są wczytywane i przetwarzane wstępnie, podobnie jak obrazy treningowe. Cechy są ekstrahowane z obrazu testowego za pomocą funkcji `extract_data`. Wytrenowany klasyfikator `clf` przewiduje prawdopodobieństwa dla każdego patcha (piksela) na obrazie testowym. Prawdopodobieństwa są progowane za pomocą wcześniej wyznaczonego `best_thr`, aby uzyskać binarną maskę predykcji (`ypred`). Maska predykcji jest rekonstruowana z płaskiej listy predykcji na dwuwymiarowy obraz, używając zapisanych współrzędnych pikseli. Zrekonstruowana maska jest poddawana post-processingowi za pomocą funkcji `postprocess_mask` w celu wygładzenia i oczyszczania.

4.6 Ocena i wizualizacja

Na koniec, kod analogicznie do poprzedniej części sprawozdania oblicz wszystkie metryki. Wyświetlany jest też oryginalny obraz, maska ekspercka (GT) oraz obraz z nałożonym overlayem predykcji.



Rysunek 2: Wyniki segmentacji dla jednego obrazu z zestawu testowego k-NN. Nałożenie kolorów przedstawia: zielony TP, żółty FP, niebieski FN. Widoczne są również metryki acc, sens, spec, balanced score, macierz pomyłek oraz wybrany próg.

5 Klasyfikator Random Forest

Lista importowanych bibliotek jest bardzo podobna do poprzedniej wersji, z jedną kluczową zmianą:

- `sklearn.ensemble.RandomForestClassifier` nowy klasyfikator używany w tym podejściu.

Funkcje pomocnicze pozostają bez zmian. Dobór optymalnego progu decyzyjnego jest identyczny jak w poprzedniej wersji.

5.1 Trening klasyfikatora Random Forest

```
1 Xtr, ytr = [], []
2 for fn in train_f:
3     # kod wczytania, preprocessingu obrazu i maski
4     # kod ekstrakcji danych za pomocą extract_data
5     Xtr.append(Xf)
6     ytr.append(yf)
7
8 Xtr = np.nan_to_num(np.array(Xtr))
9 ytr = np.array(ytr)
10
11 # balansowanie klas
12 pos_idx = np.where(ytr == 1)[0]
13 neg_idx = np.where(ytr == 0)[0]
14 neg_sel = np.random.choice(neg_idx, size=len(pos_idx), replace=False)
15 sel      = np.concatenate([pos_idx, neg_sel])
16 Xbal, ybal = Xtr[sel], ytr[sel]
```

```

17 # train/val i tren RF
18 X_train, X_val, y_train, y_val = train_test_split(
19     Xbal, ybal, test_size=0.3, random_state=42, stratify=ybal
20 )
21
22 clf = make_pipeline(
23     StandardScaler(),
24     RandomForestClassifier(
25         n_estimators=100,
26         class_weight='balanced',
27         random_state=42,
28         n_jobs=-1
29     )
30 )
31
32 clf.fit(X_train, y_train)

```

Listing 12: Trening i walidacja klasyfikatora Random Forest

Proces zbierania danych treningowych i balansowania klas jest identyczny jak poprzednio. Główna zmiana dotyczy definicji i treningu klasyfikatora:

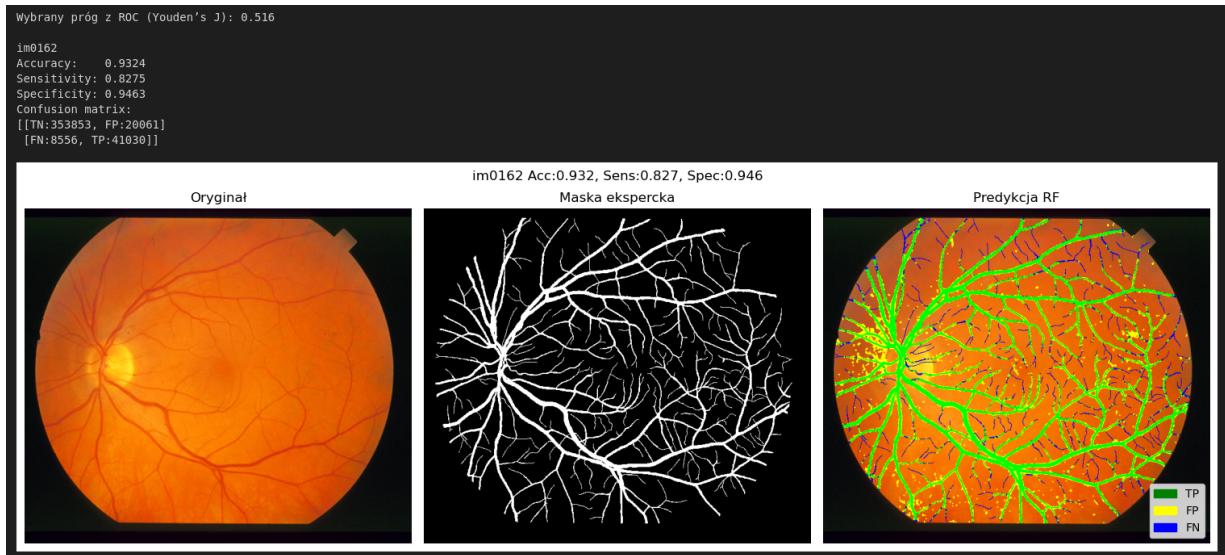
- `RandomForestClassifier` to algorytm oparty na drzewach decyzyjnych, tworzy wiele drzew decyzyjnych na różnych podzbiorach danych treningowych i cech, a następnie agreguje ich predykcje (głosowanie większościowe dla klasyfikacji), aby uzyskać bardziej stabilny i dokładny wynik;
- `n_estimators=100` definiuje liczbę drzew decyzyjnych w lesie, 100 drzew to typowa wartość początkowa;
- `class_weight='balanced'` automatycznie dostosowuje wagi klas odwrotnie proporcjonalnie do częstości występowania klas w danych wejściowych ($w_j = \frac{n_{samples}}{n_{classes} \cdot n_j}$), pomaga to RandomForestowi lepiej radzić sobie z niezbalansowanymi danymi, zwracając większą uwagę na klasę mniejszościową (naczynia).

5.2 Segmentacja i ocena jednego obrazu

Sekcja ta również jest w dużej mierze identyczna. Obraz testowy jest wczytywany, przetwarzany wstępnie, cechy są ekstrahowane, a następnie wytrenowany model Random Forest dokonuje predykcji prawdopodobieństw. Prawdopodobieństwa te są progowane z użyciem `best_thr`, aby uzyskać binarną maskę predykcji. Maska jest rekonstruowana i poddawana post-processingowi za pomocą funkcji `postprocess_mask`.

5.3 Ocena i wizualizacja

Metryki są obliczane i wyświetlane w konsoli. Wizualizacja odbywa się poprzez wygenerowanie trzech paneli, oryginalnego obrazu, maski eksperckiej oraz obrazu predykcji z nałożonymi kolorami.



Rysunek 3: Wyniki segmentacji dla jednego obrazu z zestawu testowego Random Forest. Nałożenie kolorów przedstawia: zielony TP, żółty FP, niebieski FN. Widoczne są również metryki acc, sens, spec, balanced score, macierz pomyłek oraz wybrany próg.

5.4 Podsumowanie i różnice k-NN a Random Forest

Jest to pewna ewolucja poprzedniego podejścia opartego na K-NN, zastępując go klasyfikatorem Random Forest. Kluczowe aspekty tego podejścia to:

- **klasyfikator** Random Forest, jest zazwyczaj bardziej odporny na szum i przeuczenie, a także często osiąga lepsze wyniki niż pojedyncze klasyfikatory, takie jak K-NN, zwłaszcza na danych z wieloma cechami;
- **ważenie klas** parametr `class_weight='balanced'` w `RandomForestClassifier` jest wygodnym sposobem na automatyczne radzenie sobie z nierównowagą klas bez konieczności ręcznego undersamplingu (choć w kodzie nadal występuje);
- **robustność** Random Forest jest często mniej wrażliwy na skalowanie cech niż K-NN, choć `StandardScaler` w potoku jest wciąż dobrym zabezpieczeniem;
- przy pomocy RF zostały uzyskane lepsze wyniki, zarówno w kwestii metryk postawowych jak i macierzy pomyłek; w zbiorze znajdują się dużo gorsze obrazy, na których można byłoby przetestować oba podejścia, natomiast na testowanym, algorytm poradził sobie dobrze, co potwierdza swoistość oraz czułość na wysokim poziomie, niestety występuje pewna (nadal znaczna) ilość FP, co było jednym z największych problemów podczas pracy przy projekcie.

6 Sieć neuronowa działająca na patchach

Sieć ta została utworzona w początkowym etapie zmagań z problemem, jako, że działanie na patchach jest nieoptymalne i nie powinno się go stosować kod został opisany pokrótko.

6.1 Importy Bibliotek

Kod wykorzystuje następujące biblioteki Python, z kluczowymi zmianami dotyczącymi głębokiego uczenia:

- `tensorflow.keras.models.Model`,
- `tensorflow.keras.layers`,
- `tensorflow.keras.optimizers.Adam`,
- `tensorflow.keras.callbacks`,
- `tensorflow.keras.utils.to_categorical`.

6.2 Funkcje Pomocnicze

6.2.1 Funkcja `extract_patches(gray, mask, patch_size=5, stride=1)`

Jest to zmodyfikowana wersja funkcji `extract_data` z poprzednich wersji. Kluczową różnicą jest to, że funkcja zwraca patche o rozmiarze `patch_size x patch_size`, ich etykiety (0 lub 1) oraz współrzędne środka patcha. Będą one stanowić wejście dla sieci CNN.

6.2.2 Funkcja `build_patch_cnn(patch_size=5)`

Funkcja, definiująca architekturę CNN:

- **wejście (Input)** oczekuje patchów o rozmiarze (`patchsize, patchsize, 1`), gdzie 1 to liczba kanałów (obraz w skali szarości).
- **warstwy konwolucyjne (Conv2D):**
 - pierwsza warstwa 16 filtrów o rozmiarze `3x3`, aktywacja ReLU, `padding=same` (zachowanie rozmiaru wyjścia);
 - druga warstwa 32 filtry o rozmiarze `3x3`, aktywacja ReLU, `padding=same`;
 - warstwy konwolucyjne odpowiadają za automatyczne uczenie się cech z danych wejściowych, takich jak krawędzie, tekstury itp;
- **warstwy poolingowe (MaxPooling2D)** po każdej warstwie konwolucyjnej stosowane są warstwy Max Pooling (rozmiar `2x2`, zmniejszają one wymiary przestrzenne reprezentacji, redukując liczbę parametrów i zwiększąc odporność na drobne przesunięcia);

- **spłaszczenie (Flatten)** przekształca trójwymiarową mapę cech na jednowymiarowy wektor, który może być podany na wejście do warstw w pełni połączonych (Dense).
- **warstwa gęsta (Dense)** ukryta warstwa w pełni połączona z 64 neuronami i aktywacją ReLU;
- **warstwa Dropout** z wskaźnikiem 0.3. Losowo wyłącza 30 procent neuronów podczas treningu, co pomaga zapobiegać przeuczeniu modelu;
- **warstwa wyjściowa (Dense)** to pojedynczy neuron z aktywacją sigmoid, sygnał wyjściowy (wartość od 0 do 1) jest interpretowany jako prawdopodobieństwo przynależności patcha do klasy pozytywnej (naczynia).

6.3 Przygotowanie danych

Kod iteruje po obrazach treningowych, wczytuje je, przetwarza wstępnie (RGB do szarości, FOV) i ekstrahuje patche z użyciem `extract_patches`. Patche i ich etykiety są zbierane. Wszystkie patche są łączone w jedną dużą tablicę NumPy (np.`vstack` dla X, np.`concatenate` dla y). Podobnie jak w poprzednich wersjach, wykonuje się podpróbkowanie klasy negatywnej (tła), aby zbalansować liczebność klas i zapobiec przeuczeniu modelu na klasę większościową. Wartości pikseli (0-255) są normalizowane do zakresu [0,1]. Dodawany jest dodatkowy wymiar na końcu, reprezentujący kanały obrazu (dla obrazów w skali szarości jest to 1). Keras oczekuje danych wejściowych w formacie (liczba próbek, wysokość, szerokość, kanały). Zbalansowany i przygotowany zbiór danych jest dzielony na podzbiory treningowy i walidacyjny z podziałem 80 do 20 (`test_size=0.2`). Parametr `stratify=y_sel` zapewnia utrzymanie proporcji klas w obu podzbiorach. Otrzymujemy podsumowanie rozmiarów zbiorów oraz liczebności klas, co jest przydatne do weryfikacji poprawności podziału i balansu.

6.4 Trening modelu CNN

Model jest komplikowany z optymalizatorem Adam, funkcją straty binary crossentropy, metryką accuracy. Wyświetlana jest szczegółowa architektura sieci, liczba parametrów w każdej warstwie oraz całkowitą liczbę parametrów. Zestaw funkcji zwrotnych jest konfigurowany w celu optymalizacji procesu treningu, EarlyStopping, ModelCheckpoint, ReduceLROnPlateau. Trening odbywa się przez maksymalnie 20 epok.

6.5 Automatyczny dobór progu decyzyjnego

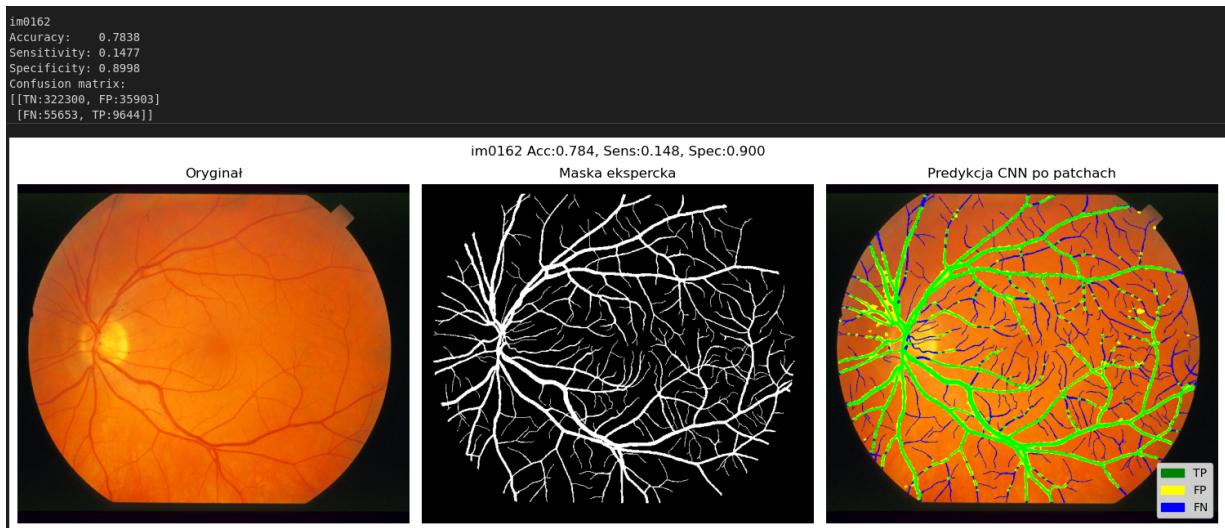
Po zakończeniu treningu (lub gdy EarlyStopping zatrzyma proces), model jest używany do predykcji prawdopodobieństw na zbiorze walidacyjnym. Następnie obliczana jest krzywa ROC, a optymalny próg decyzyjny jest wyznaczany za pomocą indeksu Youdena ($J = TPR - FPR$), co maksymalizuje równowagę między czułością a swoistością.

6.6 Segmentacja i ocena jednego obrazu

Obraz testowy i jego maska GT są wczytywane i przetwarzane wstępnie. Patche są ekstrahowane z obrazu testowego za pomocą funkcji `extract_patches`, a następnie normalizowane i przekształcane do formatu oczekiwanej przez CNN. Wytrenowany model przewiduje prawdopodobieństwa dla każdego patcha testowego. Prawdopodobieństwa te są progowane za pomocą wyznaczonego `best_thr`, aby uzyskać binarną etykietę dla każdego patcha. Binarne etykiety są używane do rekonstrukcji dwuwymiarowej maski predykcji na oryginalnym rozmiarze obrazu, wykorzystując zapisane współrzędne pikseli. Zrekonstruowana maska jest poddawana post-processingowi za pomocą funkcji `postprocess_mask`.

6.7 Ocena i wizualizacja

Na koniec, kod oblicza i wizualizuje wyniki dla przetworzonego obrazu testowego. Proces ten jest analogiczny do poprzednich wersji. W wyniku segmentacji nie otrzymano zadowalających wyników.



Rysunek 4: Wyniki segmentacji dla jednego obrazu z zestawu testowego sieci działającej na patchach. Nałożenie kolorów przedstawia: zielony TP, żółty FP, niebieski FN. Widoczne są również metryki acc, sens, spec, balanced score, macierz pomyłek.

7 Sieć neuronowa U-Net

7.1 Proces ładowania i preprocessingu

Każdy obraz jest przetwarzany w następujący sposób:

- obraz RGB jest wczytywany i konwertowany do skali szarości;
 - tworzona jest maska pola widzenia (FOV), a obraz w skali szarości jest mnożony przez tę maskę, aby wyeliminować piksele spoza obszaru oka;
 - maski eksperckie od ah i vk są wczytywane i łączone (logiczne OR), tworząc jedną maskę gruntową (Ground Truth)
 - obrazy w skali szarości i maski są skalowane do rozmiaru 256×256 pikseli, obrazy są normalizowane do zakresu $[0, 1]$, a maski konwertowane na float32;
 - do każdego obrazu i maski dodawany jest wymiar kanału (staje się kształtu ($H, W, 1$)).

7.2 Parametry dodatkowe

Nową zmienną jest `pos_weight`. Definiuje wagę przypisaną do klasy pozytywnej (pixeli naczyń krwionośnych) w funkcji straty. Przykładowo wartość 6 oznacza, że błędy w klasyfikacji pikseli naczyń będą miały sześciokrotnie większy wpływ na wartość straty niż błędy na pikselach tła. Technika została zastosowana do zaradzenia problemowi nierównowagi klas, powszechniej w zadaniach segmentacji medycznej, gdzie obiekty zainteresowania (w tym przypadku naczynia) stanowią małą część obrazu.

```
1 def weighted_bce(y_true, y_pred):
2     bce = tf.keras.losses.binary_crossentropy(y_true, y_pred)
3     y2 = tf.squeeze(y_true, axis=-1)
4     weight = 1 + (pos_weight - 1) * y2
5     return tf.reduce_mean(bce * weight)
```

Listing 13: Importy i globalna konfiguracja

`weighted_bce(y_true, y_pred)` funkcja straty, która blicza ważoną binarną entropię krzyżową (BCE). Standardowa BCE traktuje błędy dla obu klas jednakowo, ta funkcja modyfikuje BCE, mnożąc stratę dla pikseli należących do klasy pozytywnej (`y_true=1`) przez `pos_weight`, podczas gdy dla pikseli klasy negatywnej (`y_true=0`) waga pozostaje 1. To sprawia, że model jest bardziej wrażliwy na poprawne wykrywanie naczyń.

7.3 Podział danych

Ten etap jest kluczowy dla prawidłowego treningu i rzetelnej oceny modelu, polegający na podziale całego zbioru danych na trzy odrębne podzbiory.

```
1 X_tmp, X_test, Y_tmp, Y_test, Xc_tmp, Xc_test, files_tmp, files_test = train_test_split(
2     X, Y, X_color_orig, files, test_size=0.2, random_state=42 # 20% na test
3 )
4
5 X_train, X_val, Y_train, Y_val, Xc_train, Xc_val, files_train, files_val = train_test_split(
6     X_tmp, Y_tmp, Xc_tmp, files_tmp, test_size=0.2, random_state=42 # 20% z pozostałych 80%
7     na walidację
8 )
9
10 print("\nRozmiary zbiorów danych po podziale")
11 print(f"Train: {X_train.shape}, {Y_train.shape} ({len(files_train)} obrazów)")
12 print(f"Val: {X_val.shape}, {Y_val.shape} ({len(files_val)} obrazów)")
13 print(f"Test: {X_test.shape}, {Y_test.shape} ({len(files_test)} obrazów)")
```

Listing 14: Podział danych na zbiory treningowy

Funkcja `train_test_split` z biblioteki `sklearn.model_selection` jest wykorzystywana do dwustopniowego podziału danych. W pierwszym kroku, 20% całego zbioru danych jest wydzielane jako zbiór testowy (`test_size=0.2`). Pozostałe 80% danych trafia do zbiorów tymczasowych (`X_tmp`, `Y_tmp`, `Xc_tmp`, `files_tmp`), które posłużą do dalszego podziału.

W drugim kroku, zbiór tymczasowy (stanowiący 80% pierwotnych danych) jest ponownie dzielony. Tym razem 20% z niego jest przeznaczane na zbiór walidacyjny (`X_val`, `Y_val`, itd.), a pozostałe 80% na zbiór treningowy (`X_train`, `Y_train`, itd.).

`random_state=42` użycie stałej wartości dla argumentu `random_state` gwarantuje, że podział danych będzie deterministyczny i taki sam przy każdym uruchomieniu skryptu. Jest to kluczowe dla zapewnienia powtarzalności eksperymentów i porównywalności wyników.

Po podziale wyświetlane są kształty tablic NumPy dla zbiorów treningowego, walidacyjnego i testowego, a także liczbę obrazów w każdym zbiorze. Pozwala to na szybką weryfikację, czy podział został wykonany poprawnie i czy rozmiary danych są zgodne z oczekiwaniemi.

7.4 Architektura modelu U-Net

Model segmentacyjny opiera się na architekturze U-Net, która składa się z części enkoderowej i dekoderowej połączonych warstwami przeskakującymi (skip connections). Wejście do sieci ma rozmiar $256 \times 256 \times 1$ (wysokość, szerokość, kanał).

```
1 def unet(input_shape=(IMG_SIZE, IMG_SIZE, 1)):
2     inp = Input(input_shape)
3
4     c1 = Conv2D(64, 3, activation="relu", padding="same")(inp)
5     c1 = Conv2D(64, 3, activation="relu", padding="same")(c1)
6     p1 = MaxPooling2D()(c1)
7
8     c2 = Conv2D(128, 3, activation="relu", padding="same")(p1)
9     c2 = Conv2D(128, 3, activation="relu", padding="same")(c2)
10    p2 = MaxPooling2D()(c2)
11
12    c3 = Conv2D(256, 3, activation="relu", padding="same")(p2)
13    c3 = Conv2D(256, 3, activation="relu", padding="same")(c3)
14    c3 = Dropout(0.3)(c3)
15
16    u4 = Conv2DTranspose(128, 2, strides=2, padding="same")(c3)
17    m4 = concatenate([u4, c2])
18    c4 = Conv2D(128, 3, activation="relu", padding="same")(m4)
19    c4 = Conv2D(128, 3, activation="relu", padding="same")(c4)
20    c4 = Dropout(0.2)(c4)
21
22    u5 = Conv2DTranspose(64, 2, strides=2, padding="same")(c4)
23    m5 = concatenate([u5, c1])
24    c5 = Conv2D(64, 3, activation="relu", padding="same")(m5)
25    c5 = Conv2D(64, 3, activation="relu", padding="same")(c5)
26
27    out = Conv2D(1, 1, activation="sigmoid")(c5)
28
29    return Model(inp, out)
```

Listing 15: Definicja architektury sieci U-Net

Warstwa wejściowa (Input) przyjmuje obrazy o kształcie (IMG_SIZE, IMG_SIZE, 1), co oznacza obrazy o szerokości IMG_SIZE, wysokości IMG_SIZE i jednym kanale (obraz w skali szarości).

Enkoder odpowiada za ekstrakcję cech i zmniejszanie wymiarów przestrzennych obrazu.

- **Blok 1** dwie warstwy splotowe Conv2D z 64 filtrami, aktywacją ReLU i padding="same" (zachowuje rozmiar przestrzenny). Następnie warstwa MaxPooling2D, która zmniejsza rozmiar obrazu o połowę (downsampling). Wyjście tej warstwy (c1) jest zapamiętywane do użycia w skip connection.
- **Blok 2** podobnie jak Blok 1, ale z 128 filtrami. Również zawiera dwie warstwy Conv2D, po których następuje MaxPooling2D. Wyjście (c2) jest zapamiętywane.

Bottleneck jest to najniższy poziom w sieci, gdzie cechy są najbardziej abstrakcyjne. Składa się z dwóch warstw Conv2D (256 filtrów) i warstwy Dropout (z współczynnikiem 0.3), która losowo wyłącza neurony podczas treningu, pomagając zapobiegać przetrenowaniu.

Dekoder odpowiada za odbudowę maski segmentacji o oryginalnym rozmiarze, wykorzystując cechy z enkodera.

- **Blok 1** rozpoczyna się warstwą Conv2DTranspose (128 filtrów, strides=2), która wykonuje upsampling (zwiększa rozmiar o 2x). Następnie jej wyjście jest łączone (concatenate) z odpowiadającym wyjściem z enkodera (c2) to jest tzw. skip connection, dostarczające kontekstowych informacji o wysokiej rozdzielczości. Po konkatenacji następują dwie warstwy Conv2D (128 filtrów) i Dropout (0.2).
- **Blok 2** podobnie jak poprzedni blok dekodera, ale z 64 filtrami. Wyjście Conv2DTranspose jest łączone z c1 (kolejny skip connection), a następnie dwie warstwy Conv2D (64 filtry).

Warstwa wyjściowa (out) to Conv2D z 1 filtrem i aktywacją sigmoid. Generuje ona pojedynczy kanał wyjściowy (maskę segmentacji), gdzie wartości pikseli w zakresie [0, 1] reprezentują prawdopodobieństwo przynależności do klasy naczynie.

7.5 Trening modelu

```
1 model = unet(input_shape=(IMG_SIZE,IMG_SIZE,1))
2 model.compile(
3     optimizer="adam",
4     loss=weighted_bce,
5     metrics=["accuracy"]
6 )
7
8 callbacks = [
9     EarlyStopping(patience=5, restore_best_weights=True),
10    ModelCheckpoint("best_unet.keras", save_best_only=True),
11    ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=3, min_lr=1e-6)
12 ]
13
14 history = model.fit(
15     X_train, Y_train,
16     validation_data=(X_val, Y_val),
17     epochs=20,
18     batch_size=8,
19     callbacks=callbacks,
20     verbose=2
21 )
22 print("Trening zakończony")
```

Listing 16: Konfiguracja i uruchomienie treningu modelu

Model jest komplikowany z optymalizatorem Adam. Jako funkcja straty używana jest ważona binarna entropia krzyżowa (weighted_bce), gdzie piksele klasy pozytywnej (naczynia) są ważone pos_weight razy więcej niż piksele tła, co ma na celu zbalansowanie nierównowagi klas. Metryką monitorowaną podczas treningu jest accuracy. Wyświetlana jest szczegółowa architektura sieci, liczba parametrów w każdej warstwie oraz całkowitą liczbą parametrów.

Callbacks są to funkcje, które są wywoływanie w określonych punktach podczas treningu i pozwalają na dynamiczne dostosowanie jego przebiegu:

- `EarlyStopping(patience=5, restore_best_weights=True)` ten callback monitoruje wartość funkcji straty na zbiorze walidacyjnym (val_loss). Jeśli val_loss nie poprawi się (nie spadnie) przez 5 kolejnych epok (patience=5), trening zostanie automatycznie zatrzymany;

- `ModelCheckpoint("best_unet.keras", save_best_only=True)` służy do zapisywania modelu, monitoruje on również `val_loss` i zapisuje wagi modelu do pliku `best_unet.keras` tylko wtedy, gdy zostanie osiągnięty nowy najlepszy wynik na zbiorze walidacyjnym;
- `ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=3, min_lr=1e-6)` dostosowuje szybkość uczenia (*learning rate*) optymalizatora, jeśli `val_loss` nie poprawia się przez 3 kolejne epoki (`patience=3`), szybkość uczenia zostaje zmniejszona o połowę (`factor=0.5`), minimalna szybkość uczenia, do której może zostać zredukowana, wynosi 10^{-6} (`min_lr=1e-6`).

7.6 Ewaluacja

Po zakończeniu procesu treningu, model jest szczegółowo ewaluowany na niezależnym zbiorze testowym, a jego wydajność mierzona jest za pomocą standardowych metryk segmentacji.

```

1 print("\nEwaluacja na zbiorze walidacyjnym (dob r progu)")
2 probs_val_flat = model.predict(X_val).flatten()
3 y_val_flat      = Y_val.flatten()
4 fpr, tpr, th   = roc_curve(y_val_flat, probs_val_flat)
5 best_idx        = np.argmax(tpr - fpr)
6 best_thr        = th[best_idx]
7
8 print(f"Optymalny próg z ROC (Youdens J) na zbiorze walidacyjnym: {best_thr:.4f}")
9
10 print("\Testowanie progu na zbiorze testowym")
11 probs_test_flat = model.predict(X_test).flatten()
12 y_test_flat     = Y_test.flatten()
13
14 print("delta\tSpec\tSens")
15 for delta in [0.00, 0.05, 0.10, 0.15]:
16     thr    = best_thr + delta
17     y_pred = (probs_test_flat >= thr).astype(np.uint8)
18
19     tn, fp, fn, tp = confusion_matrix(y_test_flat, y_pred).ravel()
20     spec = tn / (tn + fp) if (tn+fp)>0 else 0
21     sens = tp / (tp + fn) if (tp+fn)>0 else 0
22
23     print(f"{delta:.2f}\t{spec:.3f}\t{sens:.3f}")
24
25 delta_final = 0.05
26 thr_final   = best_thr + delta_final
27 y_pred_final = (probs_test_flat >= thr_final).astype(np.uint8)
28

```

Listing 17: Ewaluacja modelu i dobór progu

7.6.1 Dobór progu

Optymalny próg decyzyjny jest wybierany identycznie względem poprzednich wersji programu (prawdopodobieństwo, krzywa ROC, indeks Youdena).

7.6.2 Testowanie progu na zbiorze testowym i obliczanie metryk

Model dokonuje przewidywań na niezależnym zbiorze testowym (`X_test`), a wyniki są spłaszczone. Pętla iteruje przez różne wartości `delta` (0.00, 0.05, 0.10, 0.15), dodając je do optymalnego progu `best_thr`. Dla każdego tak zmodyfikowanego progu, predykcje są binaryzowane (piksele z prawdopodobieństwem \geq prógów są uznawane za 1, w przeciwnym razie 0). Obliczana jest macierz pomyłek (`confusion_matrix`) oraz czułość (`sens`) i swoistość (`spec`). Wyniki są wyświetlane, co pozwala ocenić, jak małe zmiany progu wpływają na te metryki.

Finalny próg (`thr_final`), po wielu testach został ustalony jako `best_thr + 0.05`. Dla tego progu, predykcje są ostatecznie binaryzowane (`y_pred_final`). Następnie obliczane są i wyświetlane główne metryki oceny jakości segmentacji dla całego zbioru testowego i dla każdego obrazu testowego oddzielnie.

7.7 Wizualizacja wyników

Wizualizacja wyników odbywa się analogicznie do poprzednich wersji.

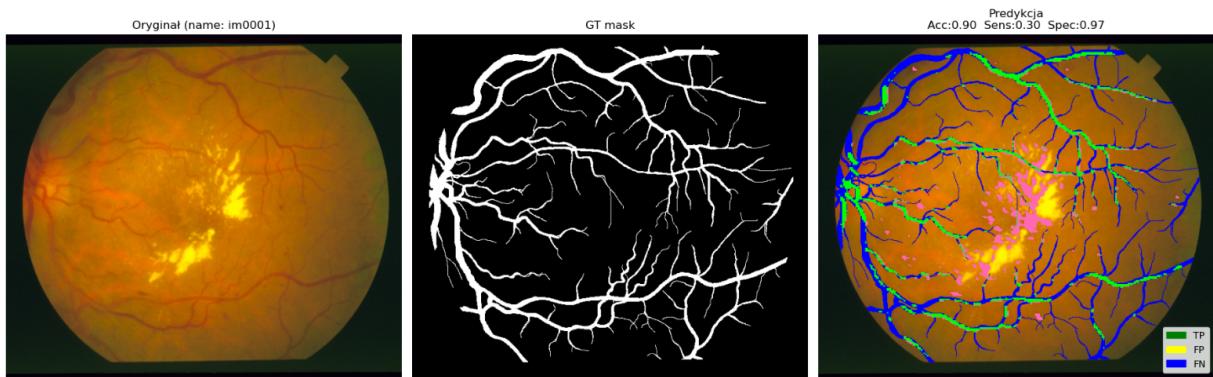
```

Epoch 2/20
2/2 - 11s - 5s/step - accuracy: 0.9114 - loss: 0.8761 - val_accuracy: 0.9083 - val_loss: 0.8633 - learning_rate: 5.0000e-04
WARNING:tensorflow:5 out of the last 13080 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x76fdc26ab060> triggered tf.function rewrites.
1/1 - 1s 738ms/step
1/1 - 1s 650ms/step
d $\delta$ ta Spec Sens
0.00 0.819 0.784
0.05 0.987 0.318
0.10 0.998 0.109
0.15 1.000 0.029

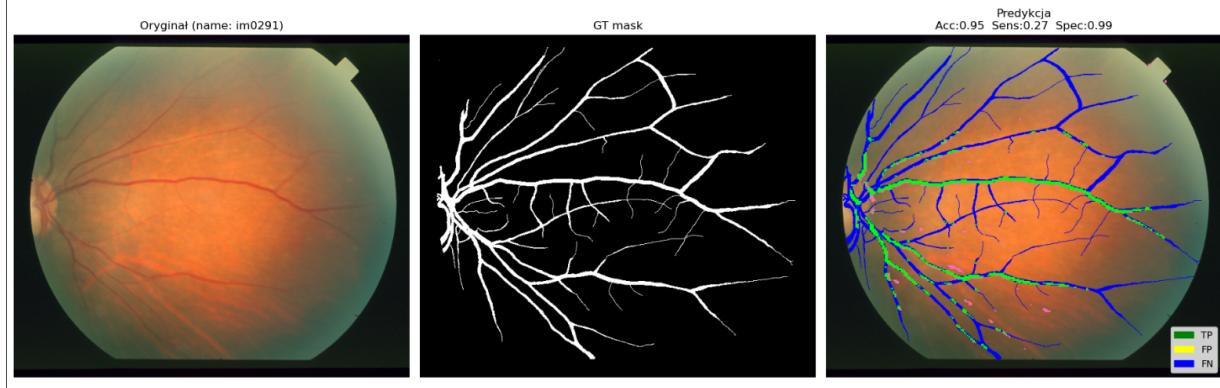
Final: Acc:0.925, Sens:0.318, Spec:0.987
Confusion matrix:
[[TN:234544, FP:2983]
 [FN:16796, TP:7821]]

Macierz pomyłek dla każdego obrazu testowego
-----
Name | TN FP FN TP
-----
im0001 | 57359 1374 4532 2271
im0291 | 60908 226 3109 1293
im0240 | 56871 150 5963 2352
im0002 | 59466 1233 3192 1705
-----
```

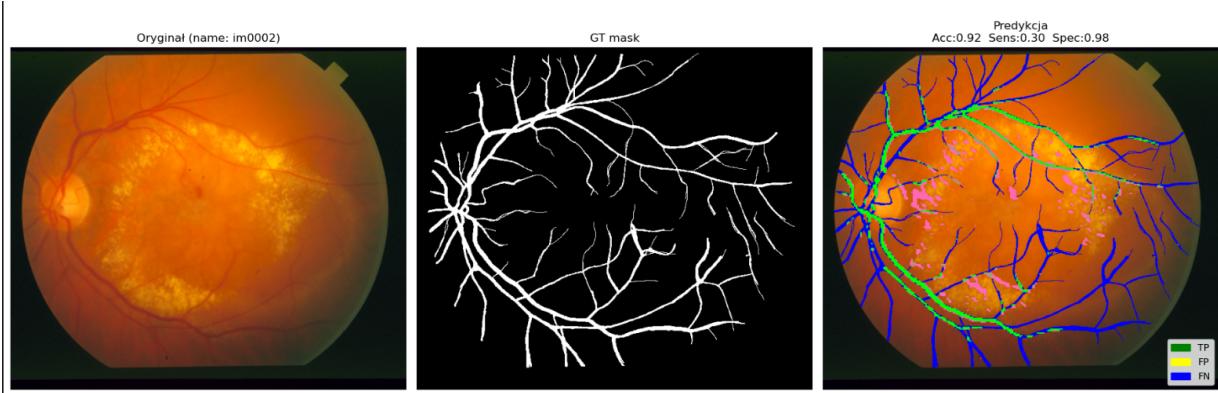
Rysunek 5: Wyniki przy parametrach: Dropout 0.3 oraz 0.2, pos_weight 6, delta 0.05. Macierz pomyłek zbiorcza oraz szczegółowa dla obrazów testowanych.



Rysunek 6: Wyniki segmentacji dla pierwszego obrazu z zestawu testowego sieci U-Net. Przy parametrach: Dropout 0.3 oraz 0.2, pos_weight 6, delta 0.05.



Rysunek 7: Wyniki segmentacji dla drugiego obrazu z zestawu testowego sieci U-Net. Przy parametrach: Dropout 0.3 oraz 0.2, pos_weight 6, delta 0.05.



Rysunek 8: Wyniki segmentacji dla trzeciego obrazu z zestawu testowego sieci U-Net. Przy parametrach: Dropout 0.3 oraz 0.2, pos_weight 6, delta 0.05.

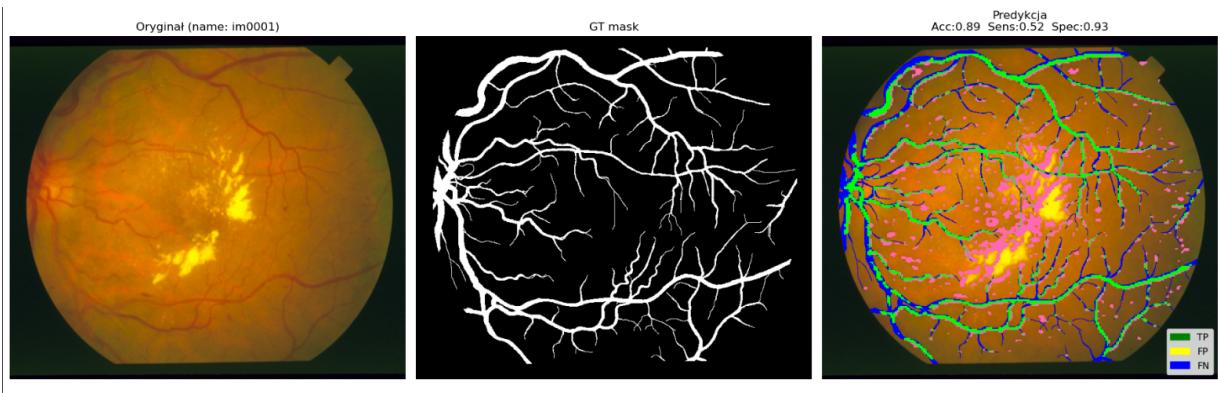
```

Epoch 19/20
2/2 - 10s - 5s/step - accuracy: 0.9201 - loss: 0.7111 - val_accuracy: 0.9195 - val_loss: 0.7313 - learning_rate: 0.0010
Epoch 20/20
2/2 - 10s - 5s/step - accuracy: 0.9689 - loss: 0.6927 - val_accuracy: 0.7841 - val_loss: 0.7169 - learning_rate: 0.0010
1/1      is 732ms/step
1/1      is 700ms/step
delta   Spec   Sens
0.00    0.831  0.801
0.05    0.963  0.578
0.10    0.988  0.360
0.15    0.996  0.206

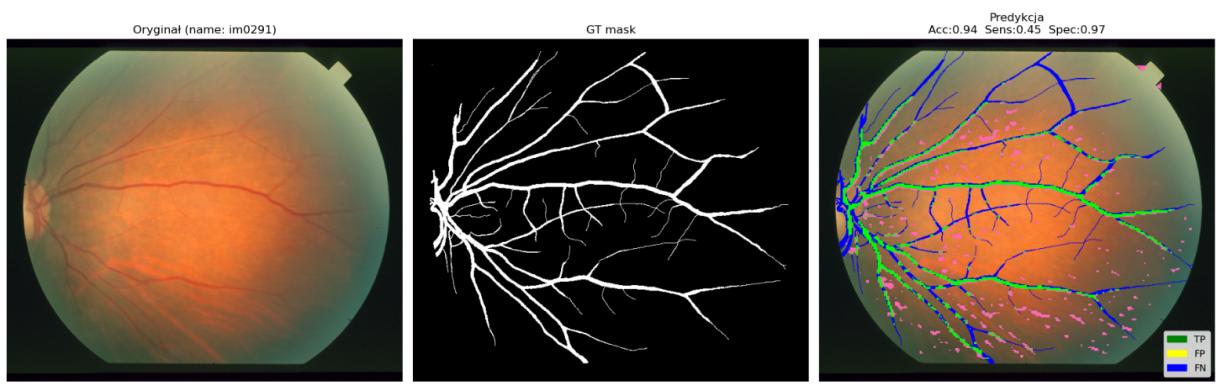
Final: Acc:0.927, Sens:0.578, Spec:0.963
Confusion matrix:
[[TN:228658, FP:8869],
 [FN:10382, TP:14235]]

Macierz pomyłek dla każdego obrazu testowego
-----
Name       | TN   FP   FN   TP
-----
im0001    | 55338 3395 2563 4240
im0291    | 59654 1480 2042 2360
im0240    | 56138 883 3878 4637
im0002    | 57528 3111 1899 2998
-----
```

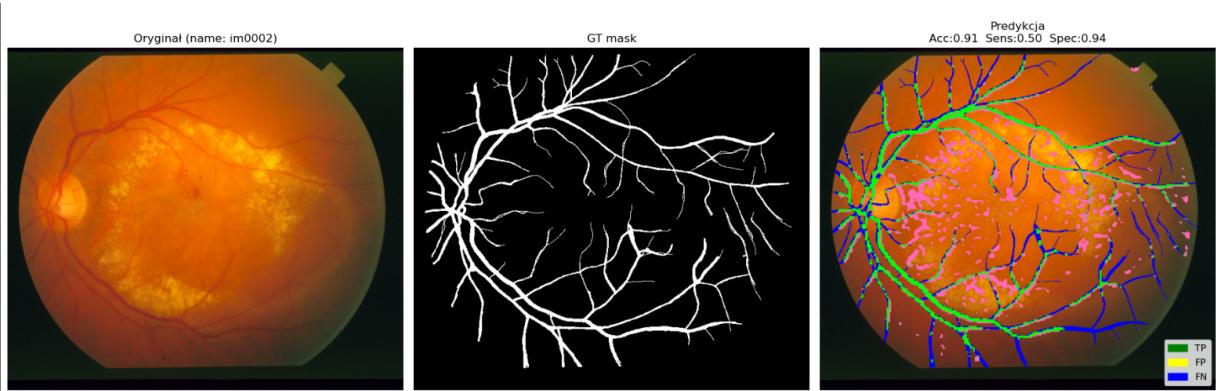
Rysunek 9: Wyniki przy parametrach: Dropout 0.2 oraz 0.1, pos_weight 5, delta 0.05. Macierz pomyłek zbiorcza oraz szczegółowa dla obrazów testowanych.



Rysunek 10: Wyniki segmentacji dla pierwszego obrazu z zestawu testowego sieci U-Net. Przy parametrach: Dropout 0.2 oraz 0.1, pos_weight 5, delta 0.05.



Rysunek 11: Wyniki segmentacji dla drugiego obrazu z zestawu testowego sieci U-Net. Przy parametrach: Dropout 0.2 oraz 0.1, pos_weight 5, delta 0.05.



Rysunek 12: Wyniki segmentacji dla trzeciego obrazu z zestawu testowego sieci U-Net. Przy parametrach: Dropout 0.2 oraz 0.1, pos_weight 5, delta 0.05.

Tabela 3: Porównanie konfiguracji

Rysunek	dropout	pos_weight	cc	Sens	Spec	Balanced Score
6 / 7 / 8	0.3 → 0.2	6	0.90 – 0.95	0.27 – 0.30	0.97 – 0.99	0.6325
10 / 11 / 12	0.2 → 0.1	5	0.89 – 0.94	0.45 – 0.52	0.93 – 0.97	0.7175

Konfiguracja 6/7/8 (dropout 0.3→0.2, pos_weight=6) osiągała tylko 27–30% czułości (wiele cienkich naczyń nie zostało wykrytych, dużo niebieskich FN na overlayu).

Konfiguracja 10/11/12 (dropout 0.2→0.1, pos_weight=5) poprawiła czułość do 45–52%, widać znacznie mniej niebieskich fragmentów (więcej zielonych TP), czyli sieć wyciąpuje więcej naczyń.

W pierwszym teście czułość była bardzo wysoka: 97–99%, niemal brak fałszywych alarmów. W drugim spadła do 93–97%, pojawia się więcej fałszywych wykryć (różowe piksele na tle).

Dropout i pos_weight regulują kompromis, wyższy dropout (0.3→0.2) i większy ciężar klasy naczyń (pos_weight=6) dają dużą ostrożność, mało FP, ale niska czułość (sieć gubi cienkie naczynia).

Niższy dropout (0.2→0.1) i mniejszy ciężar (pos_weight=5) pozwalają wychwycić więcej naczyń (wyższa sens), kosztem wzrostu fałszywych alarmów (lekki spadek spec).

Wybór parametrów zależy od priorytetów, jeśli kluczowe jest minimalizowanie fałszywych detekcji (oszczędność czasu specjalisty), lepsza jest pierwsza konfiguracja (wysoka spec).

Gdy priorytetem jest wykrycie nawet bardzo cienkich naczyń (nawet kosztem kilku FP), korzystniejszy jest drugi zestaw.

7.8 Trudności, testowane podejścia, wnioski

Obniżenie dropout i nieco słabsze karanie błędów negatywnych (mniejszy pos_weight) znacząco podniosło czułość sieci, kosztem umiarkowanego wzrostu fałszywych alarmów. Przełożyło się to na wyraźną poprawę zbalansowanego wyniku, co jest pożądane, gdy priorytetem jest wychwycenie wszystkich struktur naczyń.

Przy doborze parametrów głównym pytaniem było: czy bardziej istotna jest prawidłowa segmentacja tła, kosztem niskiej wykrywalności naczyń, czy kosztem błędного zakwalifikowania tła zwiększyć wykrywalność prawdziwych naczyń. Dobór najlepszych parametrów odbył się na drodze wielu testów i nierównej walki z wysoce niezrównoważonymi klasami. Głównie, testy polegały na zmianie wartości delty (0.00-0.15), Dropout (0.1-0.3) oraz pos_weight (5-10). Wysoka dokładność, chociaż jest własnością pożądaną nie jest zaskakującą (wiekszość obszaru to tło), kluczowy jest natomiast balans pomiędzy swoistością a czułością. W przypadku zaimplementowanej sieci, nie udało się doprowadzić do lepszego odróżnienia tła od naczyń. W post-processingu możnaby spróbować oczyścić maskę predykcji z bardzo małych elementów, które są fałszywie wykrywane jako naczynia (poza dopracowaniem samej sieci).