Projekt 2 – Wykrywanie naczyń dna siatkówki oka

1. Skład grupy:
    1. Jakub Górny 136711
    2. Jakub Eichner 136701
2. Projekt został wykonany przy użyciu Python Jupyter Notebook wspomaganego dodatkowymi bibliotekami:
    - numpy
    - matplotlib
    - skimage – biblioteka do przetwarzania obrazów
    - sklearn – biblioteka do uczenia maszynowego

3. Opis zastosowanych metod:
    a. Przetwarzanie obrazów:

    Podajemy na wejście obraz siatkówki oka. Dany obraz jest wczytywany i konwertowany na obraz czarno biały. Następnie następuje wstępne przetwarzanie obrazu.

```
def preprocess(img): #Preprocess input image

    img = filters.gaussian(img, sigma=0.4)
    img = exposure.equalize_adapthist(img, clip_limit=0.03)

    return img
```

    Przez obraz przepuszczamy filtr gaussowski w celu redukcji szumu, a następnie zastosowujemy normalizacje histogramu koloru w celu przyciemnienia jasny obszarów i rozjaśnienia ciemniejszych obszarów. Następnie z takiego obrazu tworzymy maske siatkówki oka.

```
def getMask(img): #Create mask for image
    mask = feature.canny(img) * 1.
    mask = convex_hull_image(mask)
    return mask
```

    Wykrywamy kontur oka i zamalowujemy wnętrze tego konturu. Utworzoną maskę wykorzystamy następnie w celu usunięcia znalezionego obramowania przy wykrywaniu naczyń krwionośnych, oraz przy porównaniu wyniku z maską ekspercką. Następnie następuje proces wykrywania naczyń krwionośnych.

```
def process(img): #Process image looking for blood vessels
    img = filters.frangi(img)
    minimum = np.min(img)
    maximum = np.max(img)
    img = (img - minimum) / (maximum - minimum)
    img = img * (img>np.percentile(img, 80))
    img = (img>0) * 1.0
    return img
```

    Do wykrywania wykorzystujemy filtr Frangi, który wykrywa ciągłe krawędzie. Wynik działania filtru następnie normalizujemy do wartości z przedziału [0:1].
    Następnie pozostawiamy tylko fragmenty o jasności większej od 80 percentyla obrazka. Dzięki tej operacji usuwamy szum powstały przez wykrycie zbyt wielu

krawędzi. Na koniec tworzymy z tego maskę binarną i przechodzimy do procesu poprawienia jakości otrzymanego wyniku.

```
def postprocess(img, mask): #Postprocess image

    img = dilation(img)
    img = erosion(img)

    img = img*mask

    return img
```

W ostatniej fazie przetwarzania obrazu staramy się poprawić jakość krawędzi poprzez zastosowania dwóch funkcji. Najpierw korzystamy z dylatacji, która powiększy zarys naszych krawędzie, a następnie wykorzystując erozje zmniejszamy tę krawędzie. Może to nam pozwolić na połączenie przerwanych krawędzi i polepszyć nasz wynik. Na koniec łączymy otrzymany wynik z wcześniej stworzoną maską, dzięki czemu usuniemy niepotrzebnie znalezioną krawędź oka i zostawić jedynie wnętrze oka. Utworzona maska nie jest niestety idealna i czasami mogą pozostać pewne fragmenty konturu. Uzyskany wynik łączymy z wejściowym zdjęciem zakolorowując na biało znalezione naczynia krwionośne.

```
def combine(img, res): #Combine result with input image
    init = img.copy()
    init[res==1, :] = 255
    return init
```

Otrzymany wynik porównujemy z maską ekspercką wyliczając jakość, czułość, swoistość i średnią arytmetyczną czułości i swoistości. Dzięku utworzeniu maski oka, możemy wyliczyć dane jedynie we wnętrzu oka, a nie na całym obrazku.

```
def stats(res, exp, mask): #Calculate statistics for received mask, comparing with expected result
    TP = 0 #TruePositive
    TN = 0 #TrueNegative
    FP = 0 #FalsePositive
    FN = 0 #FalseNegative
    for re, ex, mas in zip(res, exp, mask):
        for i, j, ma in zip(re, ex, mas):
            if ma == 1:
                if(i==j==1):
                    TP += 1
                elif(i==j==0):
                    TN += 1
                elif i==1 and j==0:
                    FP += 1
                elif i==0 and j==1:
                    FN += 1
    accuracy = (TP + TN)/(TP + TN + FP + FN)
    sensitivity = TP / (TP + FN)
    specificity = TN / (TN + FP)
    mean_sens_spec = (sensitivity + specificity) / 2
    print("Accuracy: "+str(accuracy))
    print("Sensitivity: "+str(sensitivity))
    print("Specificity: "+str(specificity))
    print("Mean of sensitivity and specificity: "+str(mean_sens_spec))
```

b. Uczenie maszynowe:

Realizacje uczenia maszynowego rozpoczynamy od utworzenia zbioru danych. W tym celu pobieramy z wybranych obrazków po 150 losowych fragmentów obrazka(po 75 pozytywnych i negatywnych) o rozmiarze 5x5. Z każdego fragmentu pobieramy wartość środkowego piksela – czy jest naczyniem krwionośnym czy nie, sprawdzając z maską ekspercką, oraz wyliczamy momenty Hu na podstawie których będziemy uczyć nasz model. Obrazek na początku jest przekształcany na obraz czarno biały i wstępnie przetwarzany tak jak przy przetwarzaniu obrazu. Tworzymy maskę i pobieramy maskę ekspercką. Wszystkie znalezione dane zapisujemy do dwóch list, które na koniec łączymy.

```
def partImg(size=5, amount=500):#Dividing images into size x size parts, collecting hu moment of parts for machine
    half = size//2
    images=['Image_08R.jpg', 'Image_09L.jpg', 'Image_07L.jpg', 'Image_13R.jpg', 'Image_11L.jpg', 'Image_14R.jpg']
    momentsPos = []
    momentsNeg = []
    for img in images:
        initial = io.imread("images/" + img)
        img_gray = img_as_float(rgb2gray(initial))
        img_pre = preprocess(img_gray)
        mask = getMask(img_pre)
        expected = io.imread("results/" + img[:-4] + "_1stHO.png")
        expected = img_as_float(rgb2gray(expected))

        momentPos = []
        momentNeg = []
        while len(momentPos) < amount/2 or len(momentNeg) < amount/2:
            x = random.randint(half, len(img_pre)-size+half)
            y = random.randint(half, len(img_pre[0])-size+half)
            if mask[x][y] == 1:
                if expected[x][y] == 1:
                    if len(momentPos) < amount/2:
                        mu = moments_central(img_pre[x-half:x+size-half, y-half:y+size-half])
                        nu = moments_normalized(mu)
                        hu = moments_hu(nu)
                        momentPos.append(hu)
                else:
                    if len(momentNeg) < amount/2:
                        mu = moments_central(img_pre[x-half:x+size-half, y-half:y+size-half])
                        nu = moments_normalized(mu)
                        hu = moments_hu(nu)
                        momentNeg.append(hu)
        momentsPos += momentPos
        momentsNeg += momentNeg
    info = np.ones(len(momentsPos) + len(momentsNeg))
    info[len(momentsPos):] = 0
    allMom = momentsPos + momentsNeg
    return info, allMom
```

Tak utworzone dane wykorzystujemy następnie przy tworzeniu modelu. W naszym projekcie postanowiliśmy utworzyć dwa modele. Pierwszy model wykorzystywał wbudowany klasyfikator odległościowy z biblioteki sklearn – KneighborsClassifier z ustawieniem parametru n_neighbors na wartość 15(wyznaczona przez porównywanie wyników jakościowych modelu z różnymi wartościami tego parametru). Utworzone wcześniej dane dzielimy na zbiory treningowe i testowe, wykorzystując funkcje train_test_split. Ustawiamy wielkość zbioru testowego na 20%, oraz parametr shuffle na True w celu przemieszania otrzymanych zbioru. Następnie uczymy nasz model na utworzonym zbiorze treningowym i sprawdzamy jego jakość na zbiorze testowym.

```python
def model4(X, y): #Create model using KNeighborsClassifier
    X_train, X_test, y_train, y_test = train_test_split(
        X,
        y,
        test_size=0.2,
        shuffle=True,
        random_state=42,
    )
    model = KNeighborsClassifier(n_neighbors = 15)
    model.fit(X_train, y_train)
    acc = model.score(X_test, y_test)
    print("[INFO] histogram accuracy: {:.2f}%".format(acc * 100))
    return model
```

Do utworzenia drugiego modelu wykorzystaliśmy bardziej zaawansowany klasyfikator – las. Ponownie zastosujemy biblioteke sklearn, tym razem klasyfikator. Zastosujemy tutaj również k krotną walidację skrośną – w dalszych przykładach wartość k wynosi 10. Dzielimy otrzymany zbiór danych na zbiór treningowy i testowy. Następnie zbiór treningowy dzielimy na 10 różnych zbiorów treningowych i walidujących, uczymy model na każdym podzbiorze zbioru testowego i sprawdzamy go z danym zbiorem walidującym. Otrzymane wyniki zapisujemy w tablicy, a na koniec wyciągamy z niej średnią, przedstawiając wynik działania modelu na zbiorach walidujących. Na koniec wykorzystujemy zbiór testowy do określenia wyniku działania modelu.

```python
def model5(X, y, k=5): #Create model using RandomForestClassifier with k-fold cross validation
    bestScore = 0.
    X_train, X_test, y_train, y_test = train_test_split(
        X,
        y,
        test_size=0.2,
        shuffle=True,
        random_state=42,
    )
    kf = KFold(n_splits=k, shuffle = True, random_state = 42)
    model = RandomForestClassifier()
    val = []

    for train_index, test_index in kf.split(X_train):
        X_train_n, X_test_n = np.array(X_train)[train_index], np.array(X_train)[test_index]
        y_train_n, y_test_n = np.array(y_train)[train_index], np.array(y_train)[test_index]

        model.fit(X_train_n, y_train_n)
        acc = model.score(X_test_n, y_test_n)
        val.append(acc)
    acc = np.mean(val)
    print("[INFO] validation mean accuracy: {:.2f}%".format(acc * 100))
    acc = model.score(X_test, y_test)
    print("[INFO] histogram accuracy: {:.2f}%".format(acc * 100))
    return model
```

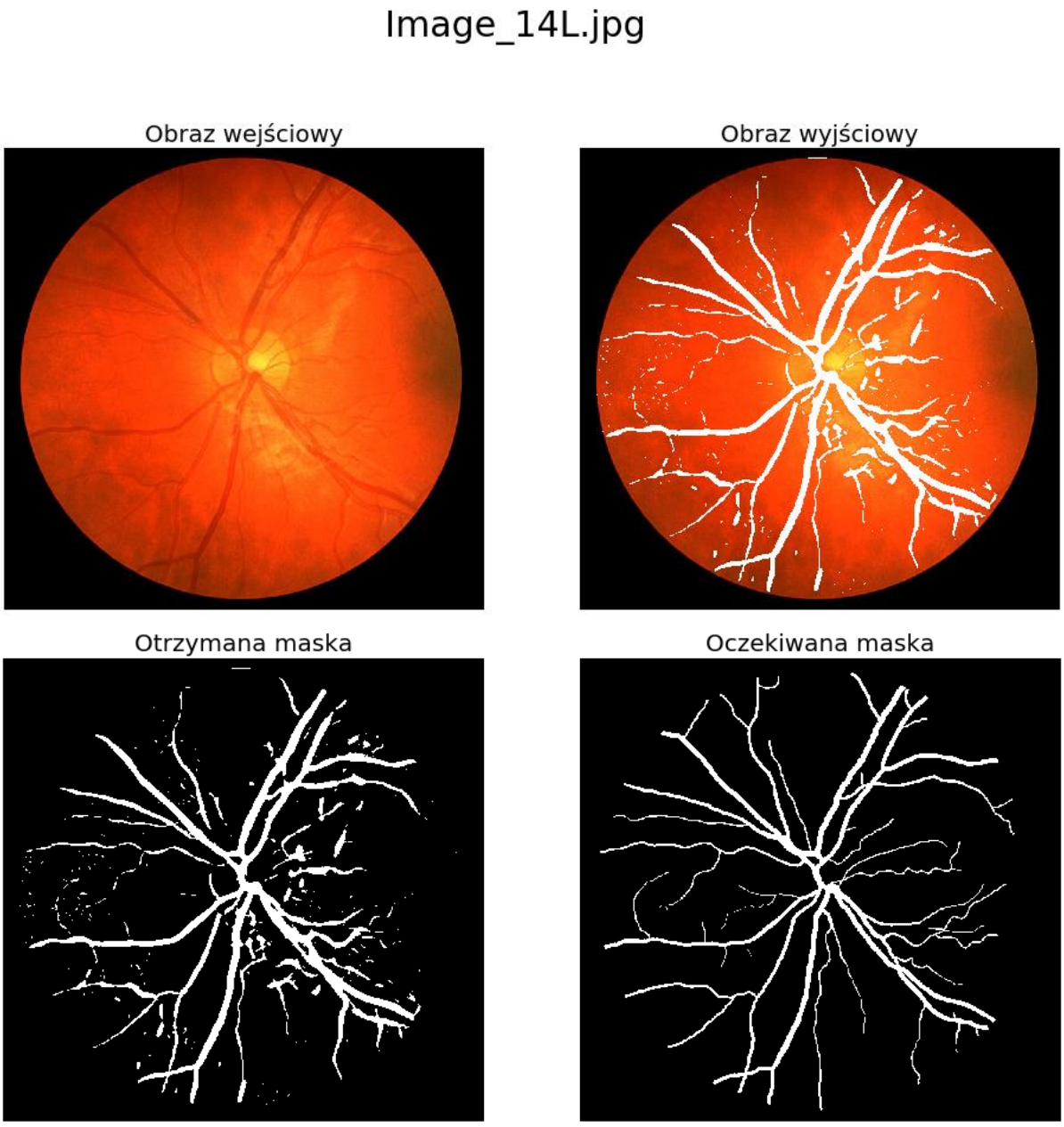
Otrzymany model następnie wykorzystujemy do wykrycia naczyń krwionośnych na całym obrazku. W tym celu wykorzystujemy podobny mechanizm co do tworzenia zbioru danych, tym razem jednak pobieramy wszystkie fragmenty 5 x 5 z wnętrza oka. Z każdego takiego fragmentu wyznaczamy momenty Hu i przekazujemy do modelu, aby ocenił czy w tym miejscu(środkowy piksel fragmentu) będzie naczynie czy nie. Operacja predykcji trwa jednak bardzo długo, dlatego najpierw zczytujemy wszystkie momenty Hu z jednego wiersza, a następnie przekazujemy je do modelu i rysujemy na odpowiednim miejscu tworzonej maski. Proces trwa bardzo długo z uwagi na konieczność sprawdzenia każdego piksela w oku, a przy dużych rozdzielczość jest ich bardzo dużo. Na koniec, podobnie jak przy przetwarzaniu

obrazu, otrzymana maska jest łączona z wejściowym obrazkiem, porównywana z maską ekspercką i wyświetlana na ekranie.

```
def imageModel(model, img, size=5): #Create mask for given image using given model
    half = size//2
    initial = io.imread("images/" + img)
    img_gray = img_as_float(rgb2gray(initial))
    img_pre = preprocess(img_gray)
    mask = getMask(img_pre)
    expected = io.imread("results/" + img[:-4] + "_1stHO.png")
    expected = img_as_float(rgb2gray(expected))

    predicted = np.zeros((len(img_pre), len(img_pre[0])))

    for x in range(half, len(img_pre) - size + half):
        tmp = []
        for y in range(half, len(img_pre[0]) - size + half):
            if mask[x][y] == 1:
                mu = moments_central(img_pre[x-half:x+size-half, y-half:y+size-half])
                nu = moments_normalized(mu)
                hu = moments_hu(nu)
                tmp.append(hu)
        if len(tmp) != 0:
            pred = model.predict(tmp)
            licz = 0
            for y in range(half, len(img_pre[0]) - size + half):
                if mask[x][y] == 1:
                    predicted[x][y] = pred[licz]
                    licz += 1

#     predicted = postprocess(predicted, mask)

    fin = combine(initial, predicted)

    showImg(img, fin, predicted, expected)
    print("Statistics for " + img)
    stats(predicted, expected, mask)
    print("")
    plt.show()
```

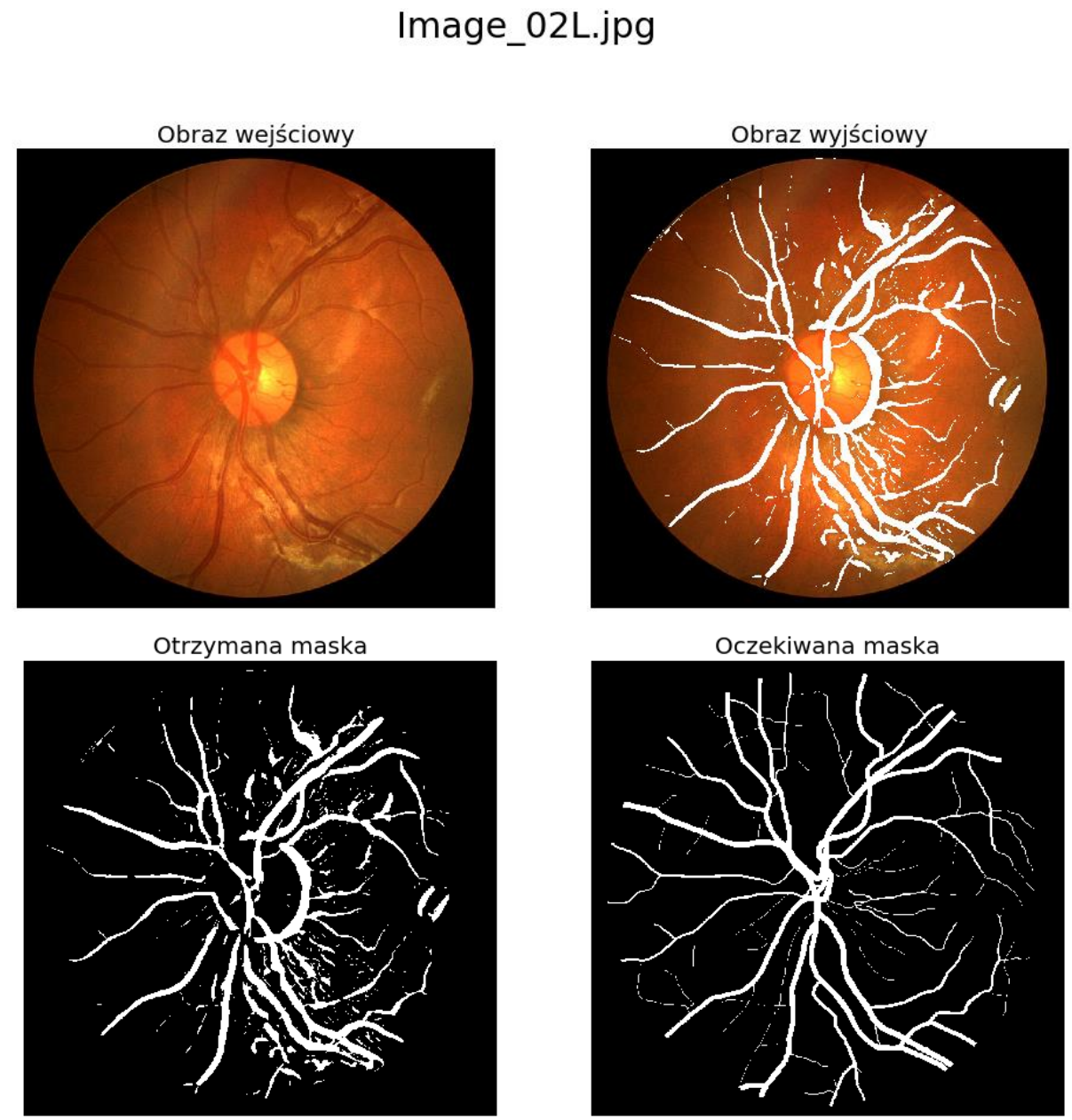
4. Wizualizacja wyników działania programu dla wybranych obrazów
    a. Przetwarzanie obrazów:

```
Statistics for Image_14L.jpg
Accuracy: 0.9312996679124351
Sensitivity: 0.7829018134440286
Specificity: 0.9477135244734152
Mean of sensitivity and specificity: 0.8653076689587219
```



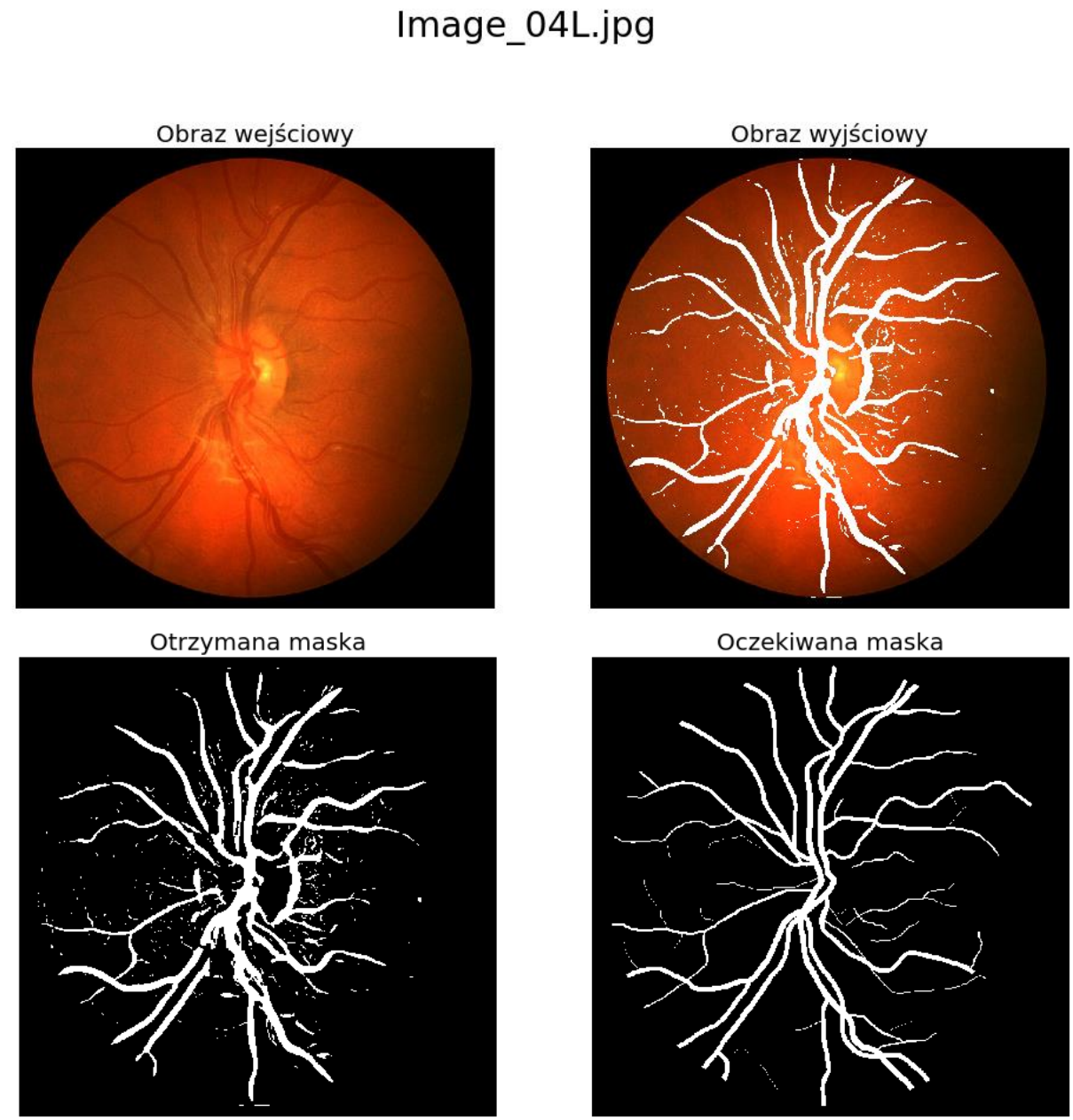
Na powyższym obrazie zostały wykryte większe żyły, problem występował przy cienkich żyłach, w pobliżu których powstawał lekki szum.

```
Statistics for Image_02L.jpg
Accuracy: 0.9027638736445983
Sensitivity: 0.6901118153243675
Specificity: 0.930951226342283
Mean of sensitivity and specificity: 0.8105315208333252
```



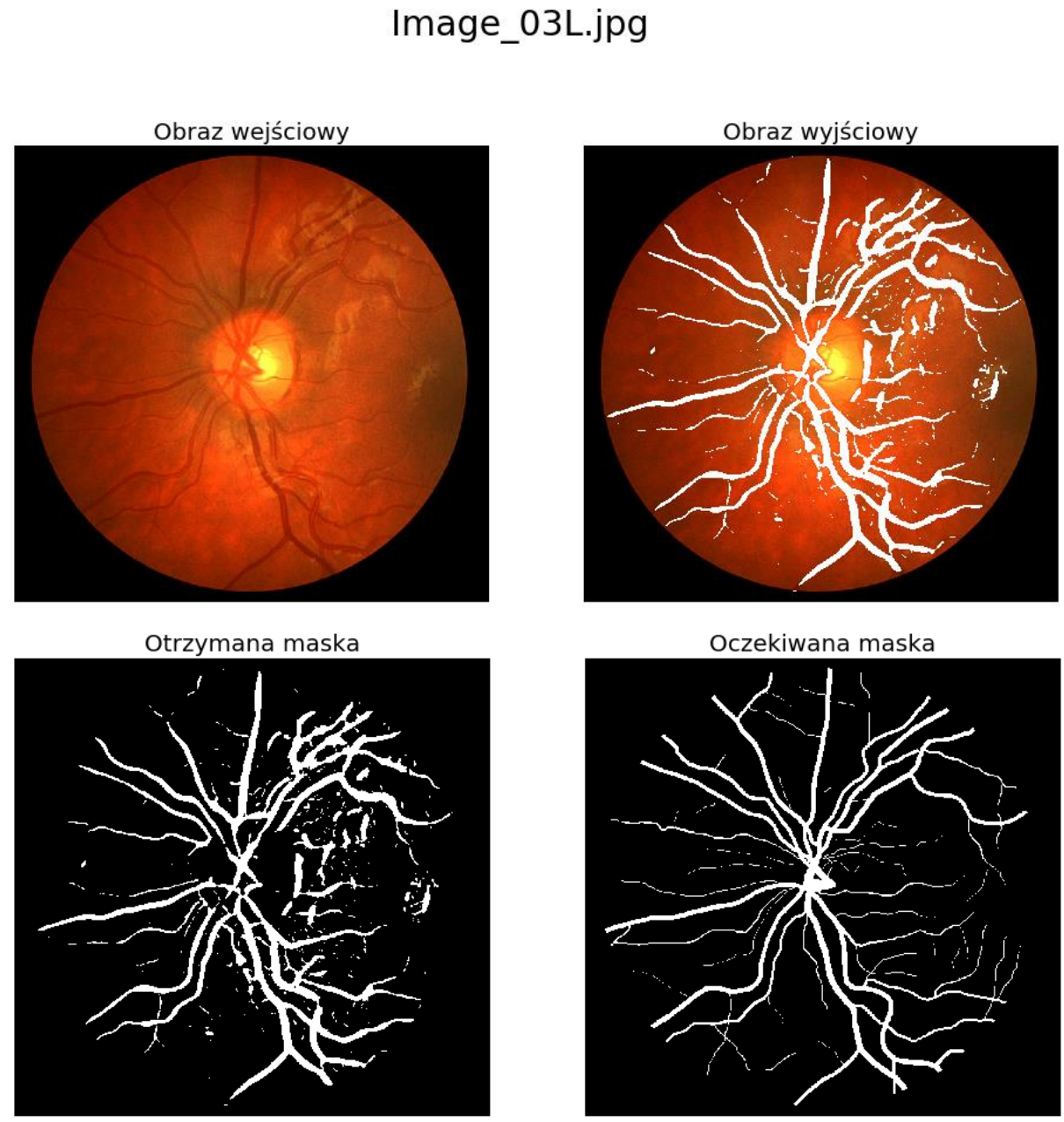
Podobnie jak przy poprzednim program znalazł wiekszę żyły, niestety błędnie wyznaczył częśc konturu tarczy nerwy wzrokowego. Widoczne są też problemy przy konturach oka.

```
Statistics for Image_04L.jpg
Accuracy: 0.9249604756537592
Sensitivity: 0.7713779412156581
Specificity: 0.9446334124666557
Mean of sensitivity and specificity: 0.8580056768411568
```



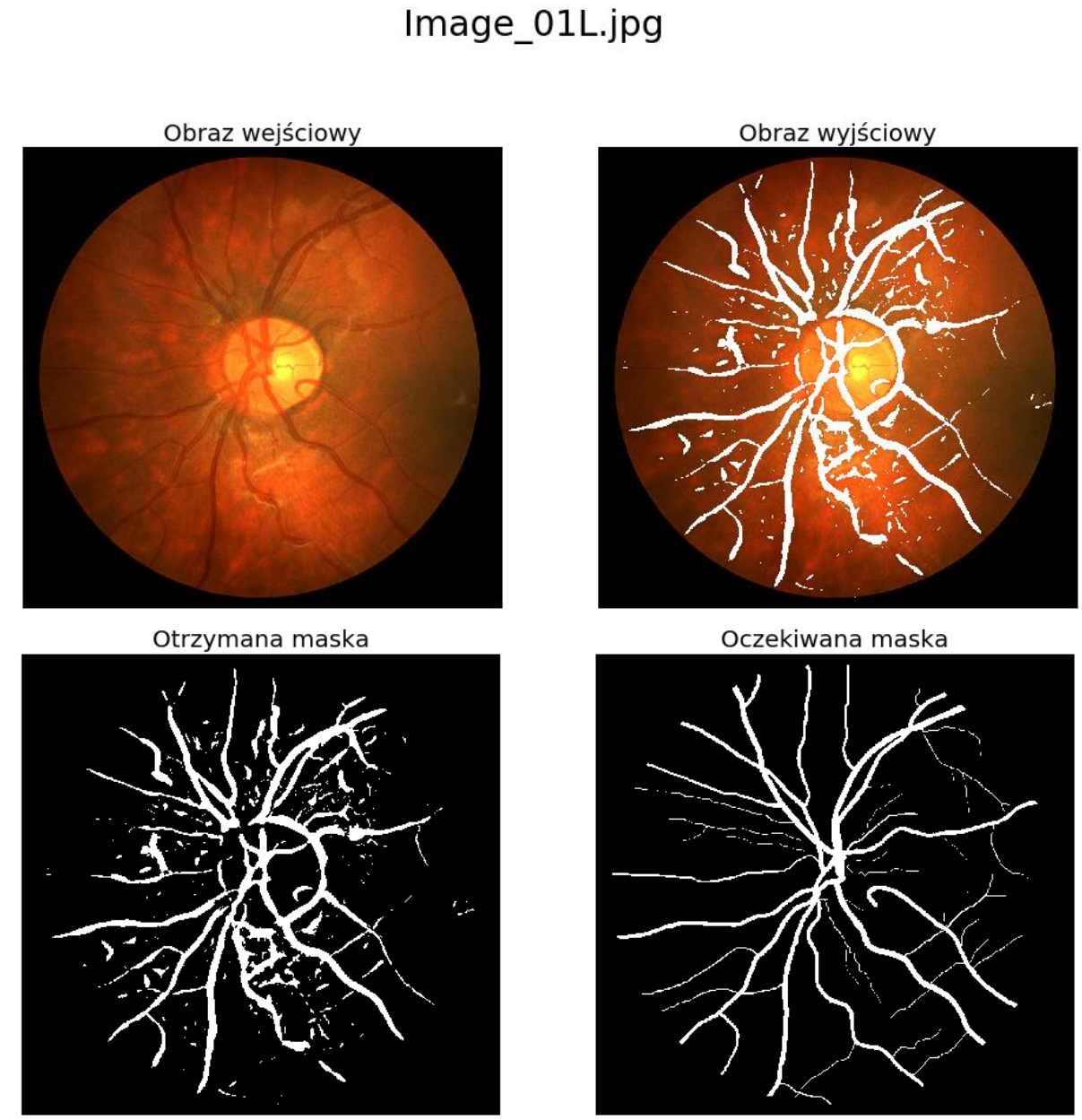
Ponownie występuje problem z konturem tarczy nerwu, lekkie problemy z cienkimi naczyniami.

```
Statistics for Image_03L.jpg
Accuracy: 0.9131644158938838
Sensitivity: 0.75489725753578
Specificity: 0.933391237705965
Mean of sensitivity and specificity: 0.8441442476208725
```



Widoczny jest duży szum w miejscu występowania cienkich naczyń, oraz brak wykrycia ich w tarczy nerwu wzrokowego.

```
Statistics for Image_01L.jpg
Accuracy: 0.9078408055101141
Sensitivity: 0.7404439518303277
Specificity: 0.9265966516298803
Mean of sensitivity and specificity: 0.833520301730104
```



Ponownie widzimy problem z konturem tarczy nerwu wzrokowego, tutaj już bardziej widoczny niż wcześniej. Wykrywanie większych naczyń nie sprawia, aż tak dużych problemów.

Porównanie wyników przetwarzania obrazu:

Zbiorcze wyniki(średnie):
Accuracy = 0.91
Sensitivity = 0.75
Specificity = 0.94
Mean of sensitivity and specificity = 0.84

Porównując statystyki wszystkich otrzymanych obrazów możemy zauważyć, że osiągnięcie wysokiej jakości i swoistości wyniku działania naszego programu nie stanowi dużego problemu, osiągając co najmniej 90% w przypadku jakości i co najmniej 85% w przypadku swoistości. Wysoki wynik swoistości oznacza, że nasz program odpowiednio oznaczał punkty, które nie były naczyniami. Gorzej już wygląda wynik czułości, która w najlepszym przypadku osiągneła około 78%, a w najgorszym około 69%. Odpowiada ona za oznaczanie punktów, które są naczyniami krwionośnymi. Możemy zauważyć, że jest to widoczne przy wykrywaniu cienkich naczyń, gdzie tworzył się duży szum. Spowodowane to może być niewystarczającym przetwarzaniem wstępnym zdjęcia, co mogło utrudniać ich wykrywanie. Średnie czułości i swoistości pokazują bardziej prawdopodobną jakość rezultatu, która wynosiła co najmniej 80%. Biorąc pod uwagę średnie wyniki wszystkich statystyk możemy być zadowoleni z działania przetwarzania.

b. Uczenie maszynowe

Klasyfikator KNeighborsClassifier

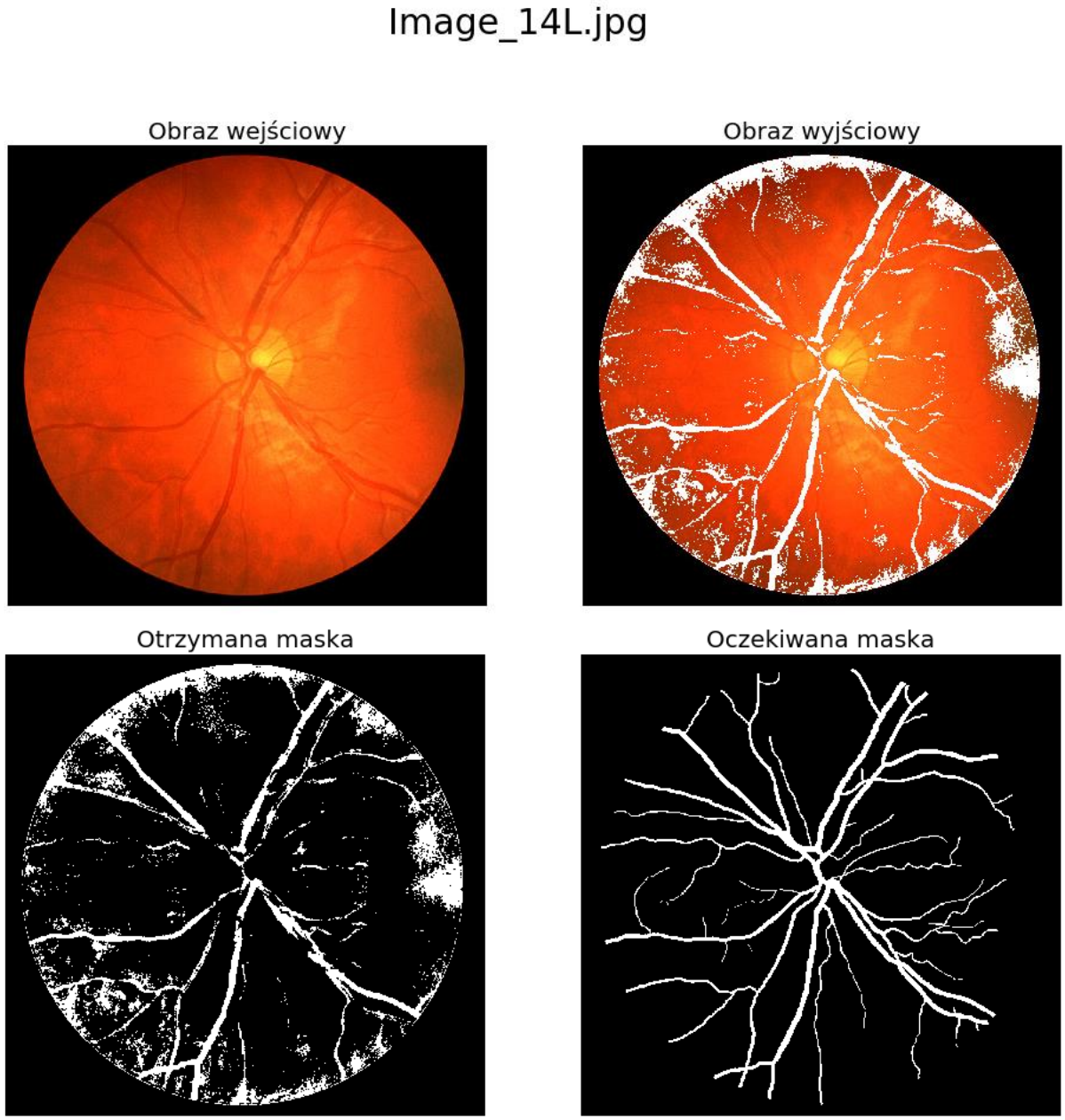
Parametry oraz jakość na zbiorze testowym:

```
grade = 4 # [5, 4] - choose model: 4-kNN 5-Forest
size = 5 # divided image shape (size x size)
k = 10 # k Fold cross validation
amount =  150 # size of collected data from one image
```

```
model = getModel(grade, size, k, amount)
```

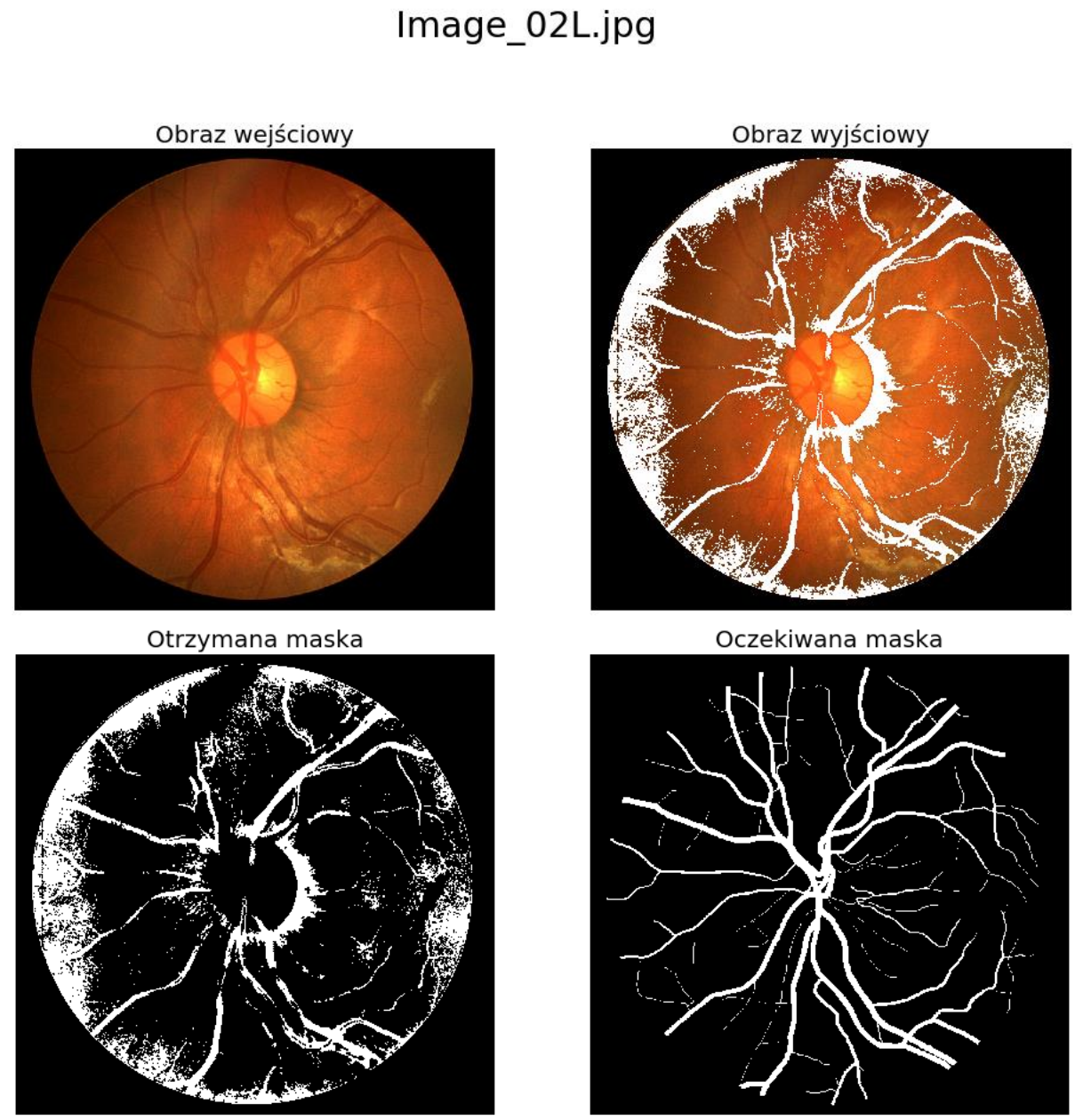
```
[INFO] histogram accuracy: 73.89%
```

```
Statistics for Image_14L.jpg
Accuracy: 0.8574528084106721
Sensitivity: 0.6031633008620297
Specificity: 0.8855790337986159
Mean of sensitivity and specificity: 0.7443711673303228
```



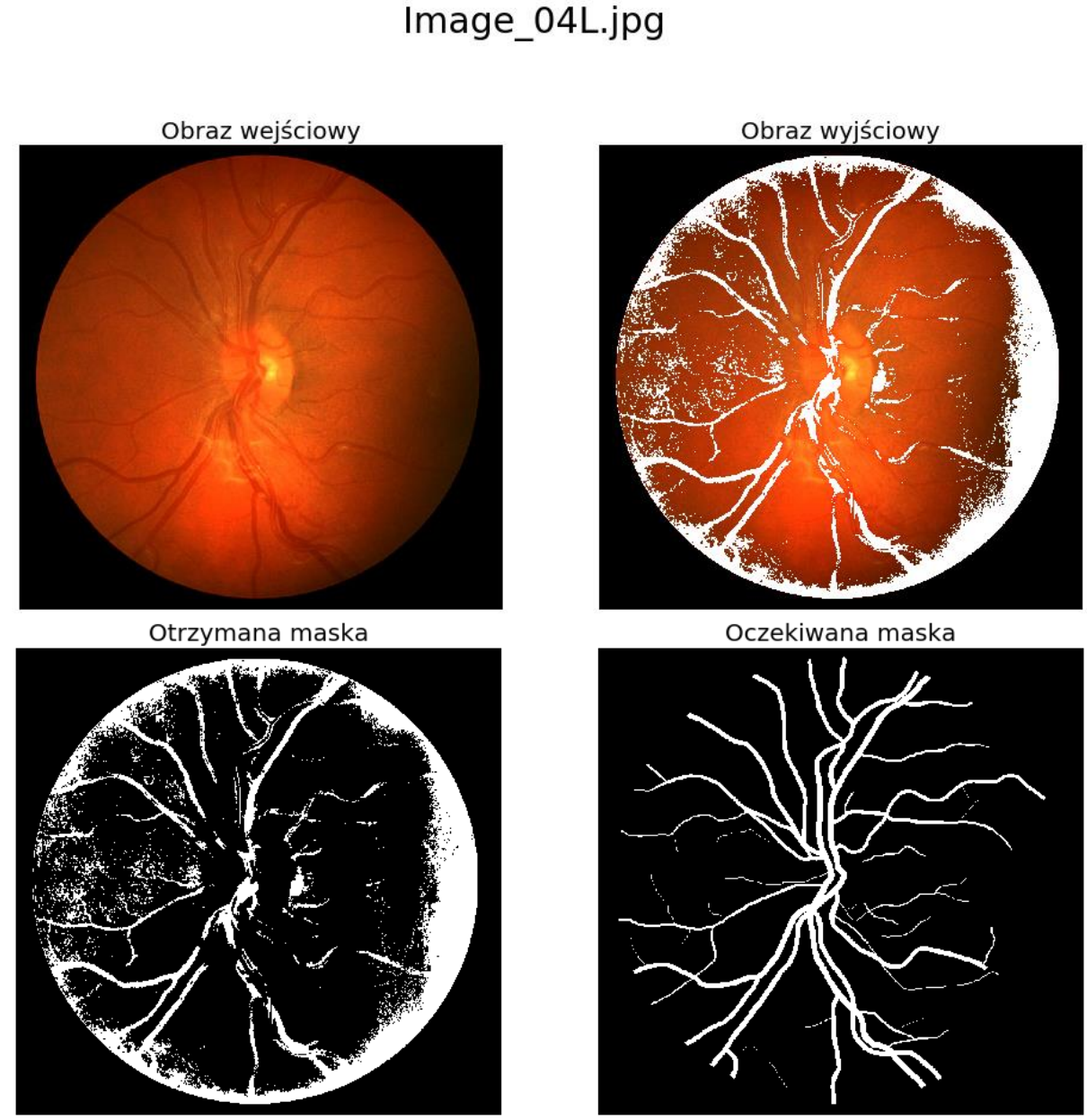
Możemy zauważyć duże problemy w okolicach konturu oka, gdzie jest wykrywane dużo punktów , które nie powinno być. Wyraźne za to są grubsze naczynia.

```
Statistics for Image_02L.jpg
Accuracy: 0.792115451880978
Sensitivity: 0.5913974339155975
Specificity: 0.8187209294383618
Mean of sensitivity and specificity: 0.7050591816769797
```



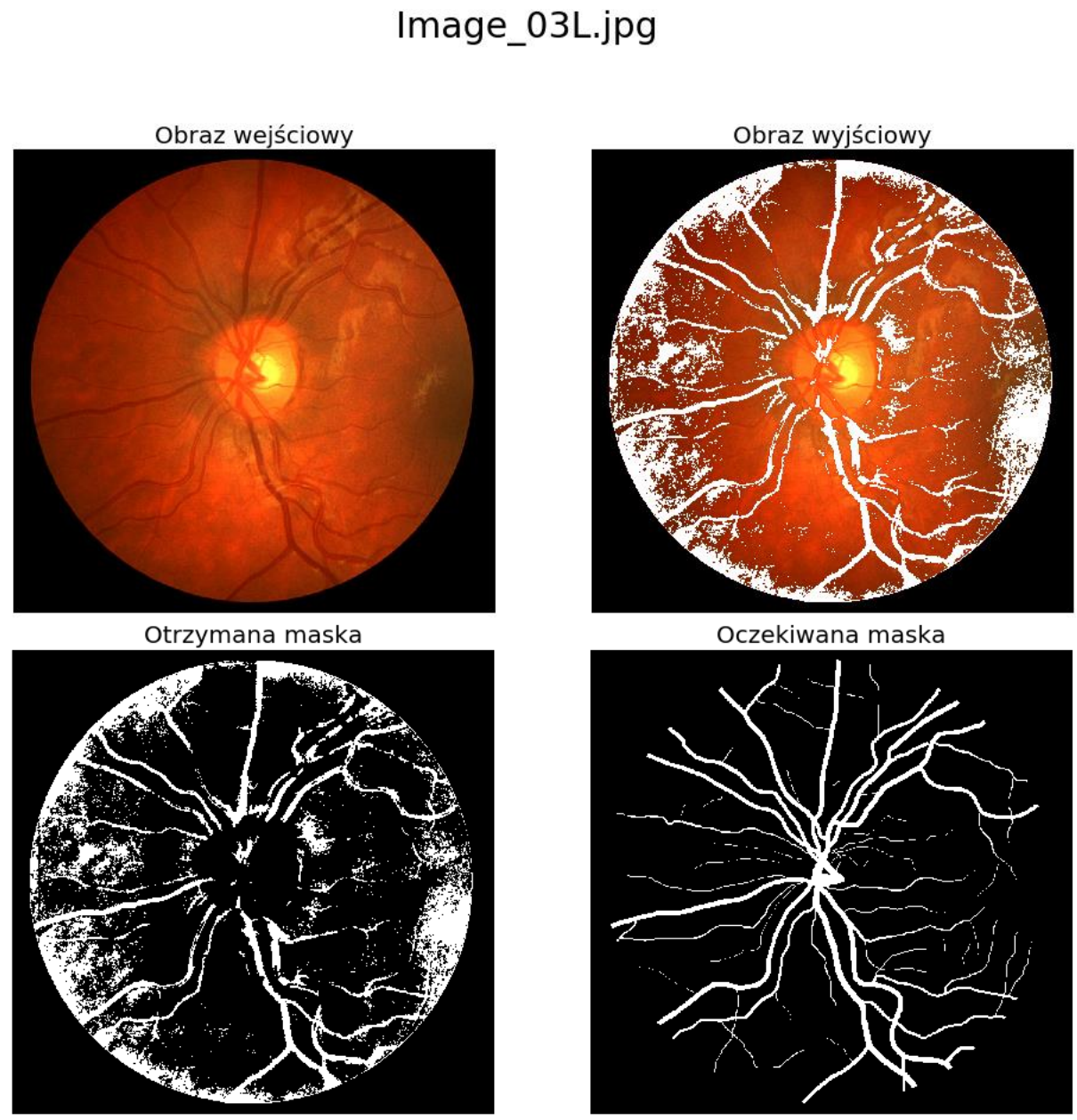
Ponownie problemy przy konturach, dodatkowo problem z konture tarczy nerwu wzrokowego.

```
Statistics for Image_04L.jpg
Accuracy: 0.7785184232622525
Sensitivity: 0.5819143549977349
Specificity: 0.8037021424024331
Mean of sensitivity and specificity: 0.692808248700084
```



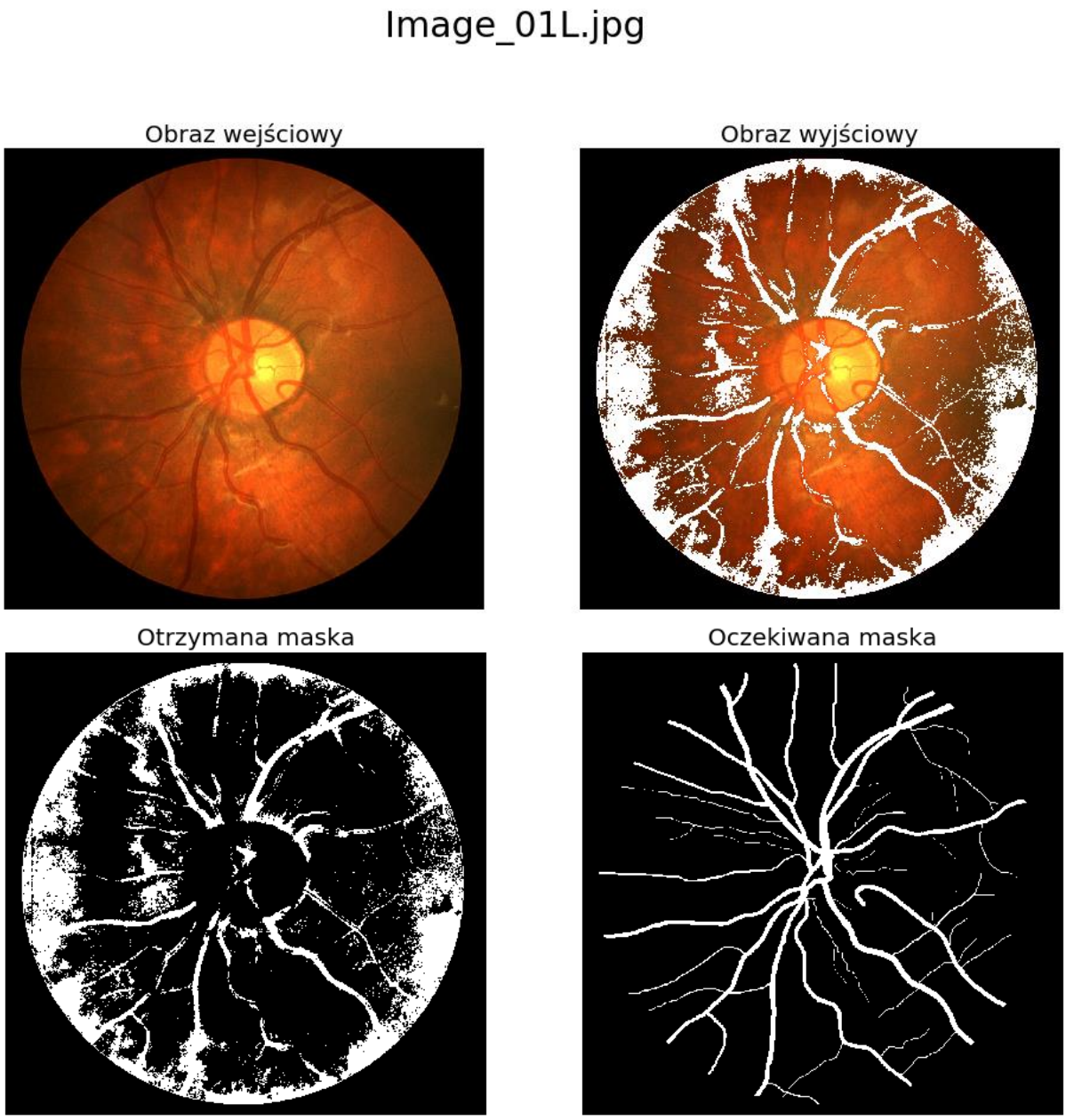
Po raz kolejny problem przy konturach, problem również w tarczy nerwu wzrokowego. Możemy zauważyć, że z prawej strony oka, gdzie obraz jest ciemniejszy, pojawił się ogromny szum.

```
Statistics for Image_03L.jpg
Accuracy: 0.8001138599582917
Sensitivity: 0.6766210921883745
Specificity: 0.8158964533571763
Mean of sensitivity and specificity: 0.7462587727727754
```



Podobne problemy co we wcześniejszych obrazkach. Możemy zauważyć, że wysoka jasność tarczy nerwu wzrokowego bardzo mocno wpływa na wynik.

```
Statistics for Image_01L.jpg
Accuracy: 0.7579787394587677
Sensitivity: 0.6168743634293931
Specificity: 0.773788666022233
Mean of sensitivity and specificity: 0.695331514725813
```



Podobnie jak wcześniej widoczny problemy z jasnym centrum oka. Widoczne dobrze grubsze naczynia krwionośne.

Porównanie wyników klasyfikatora odległościowego:

Zbiorcze wyniki(średnie):
Accuracy = 0.80
Sensitivity = 0.61
Specificity = 0.82
Mean of sensitivity and specificity = 0.72

Podobnie jak w przypadku przetwarzania obrazu lepiej wygląda oznaczanie braku naczynia od wyznaczenia czy jest naczynie. Model miał duże problemy, gdy była zbyt duża jasność i uznawał obszar jako negatywny, lub obszar był zbyt ciemny i oznaczał obraz jako pozytywny. Średbua jakość otrzymanych wyników wynosiła 80%. W przypadku czułości wynik jest przeciętny – 61%. Zadowalająco wygląda to w przypadku swoistości, gdzie otrzymaliśmy średnią równą 82%. Średnie czułości i swoistości pokazują realną jakość działania, która wynosiła średnio 72%.

Klasyfikator RandomForestClassifier

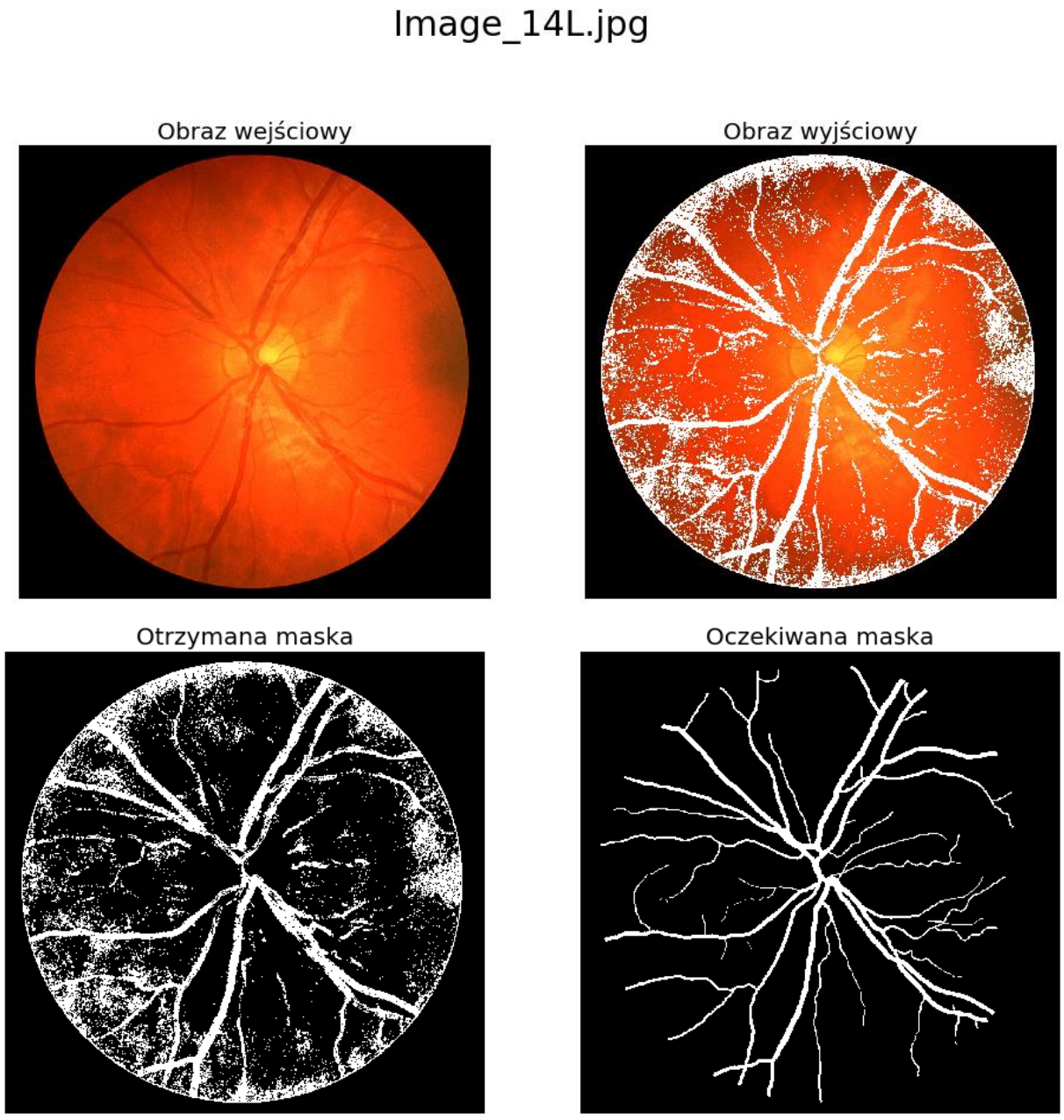
Parametry oraz jakość na zbiorze testowym:

```
grade = 5 # [5, 4] - choose model: 4-kNN 5-Forest
size = 5 # divided image shape (size x size)
k = 10 # k Fold cross validation
amount =  150 # size of collected data from one image
```

```
model = getModel(grade, size, k, amount)
```

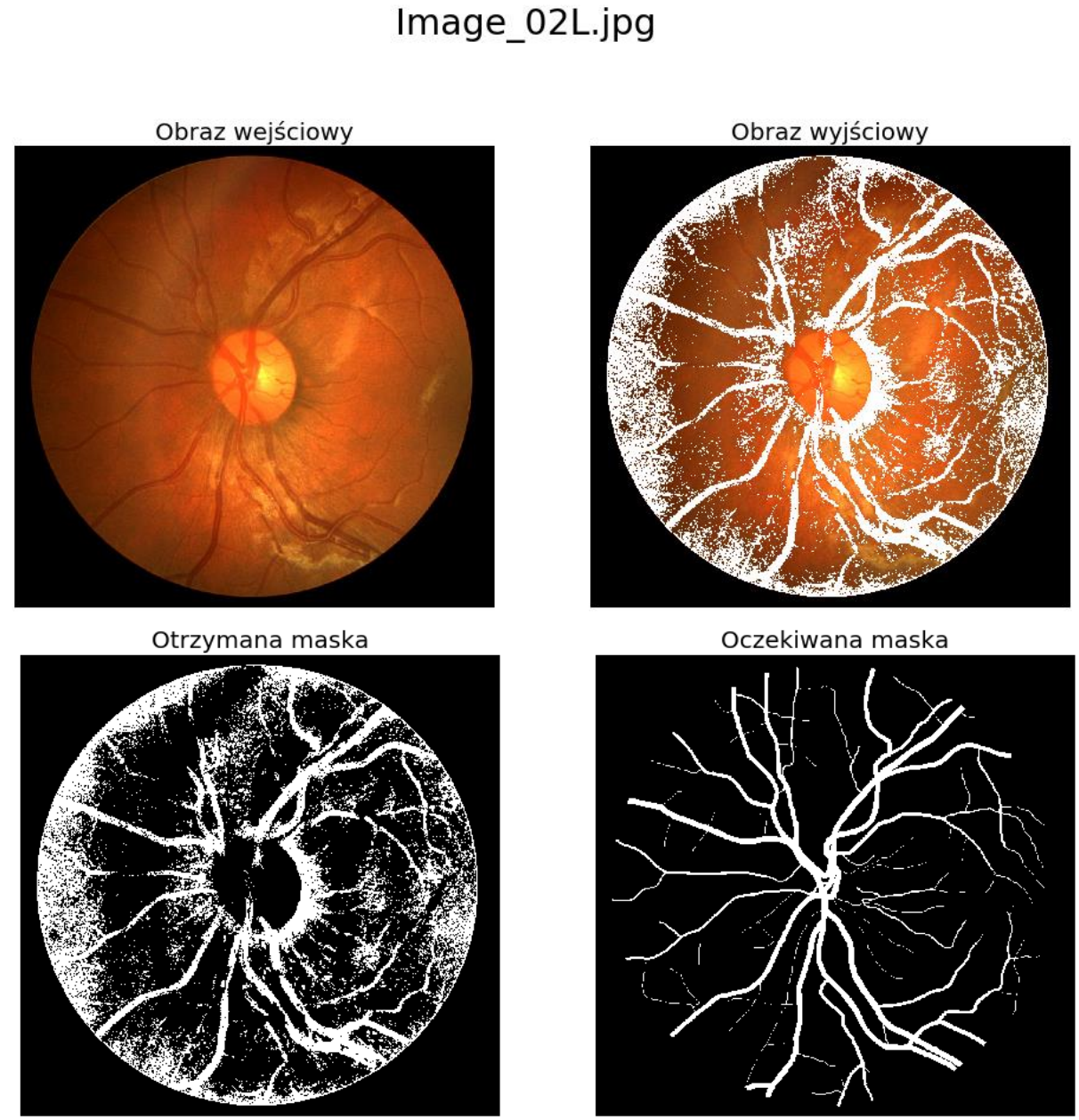
```
[INFO] validation mean accuracy: 75.69%
[INFO] histogram accuracy: 75.56%
```

```
Statistics for Image_14L.jpg
Accuracy: 0.7971384040182444
Sensitivity: 0.776326753223143
Specificity: 0.7994403203914406
Mean of sensitivity and specificity: 0.7878835368072918
```



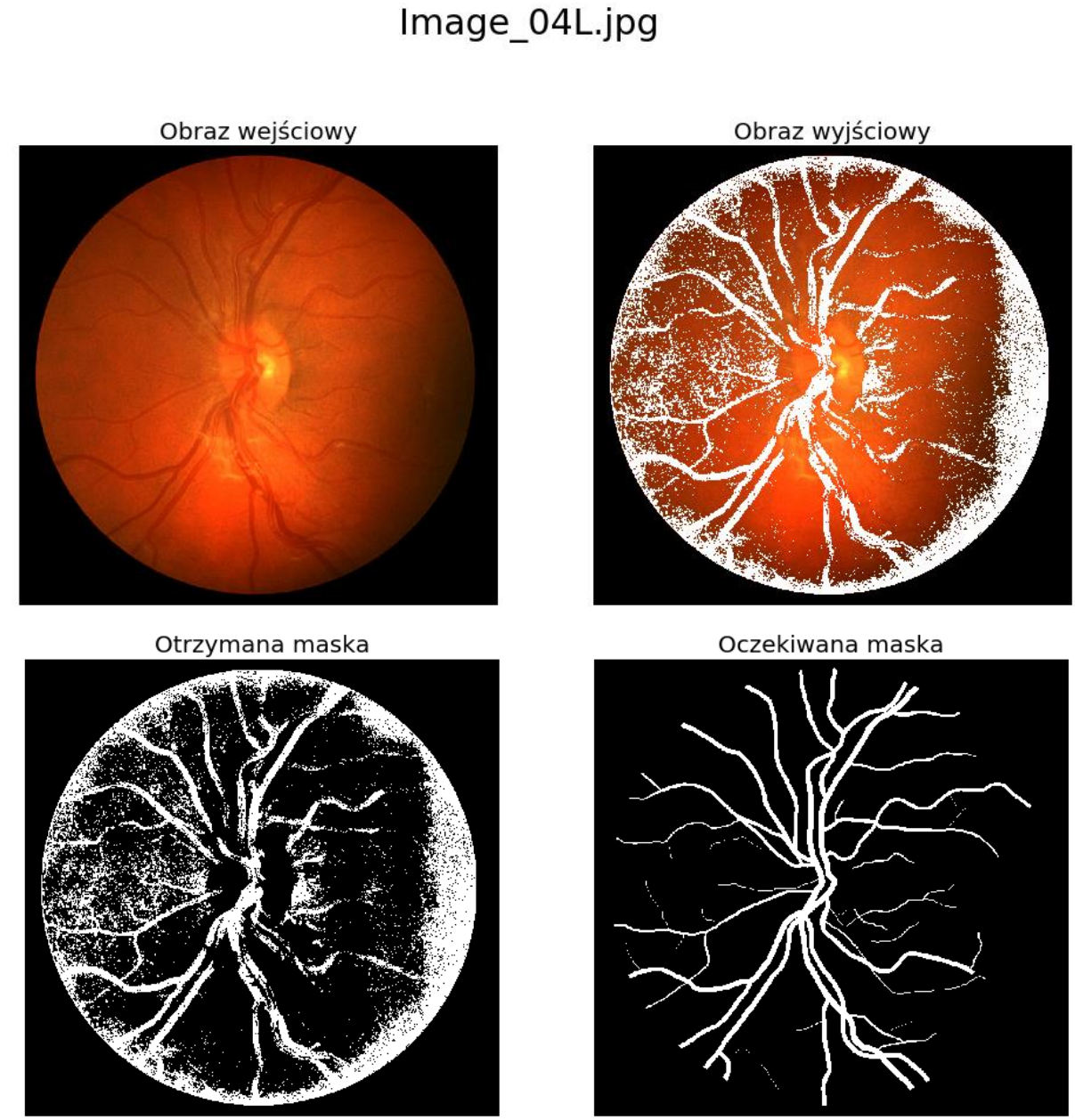
Możemy zauważyć dobre wyznaczanie grubszych naczyń, lekkie problemy w jasny centrum oka i większe w ciemniejszych cześciach w okolicach kontura

```
Statistics for Image_02L.jpg
Accuracy: 0.7302584761606098
Sensitivity: 0.752872674808059
Specificity: 0.7272609298481671
Mean of sensitivity and specificity: 0.7400668023281131
```



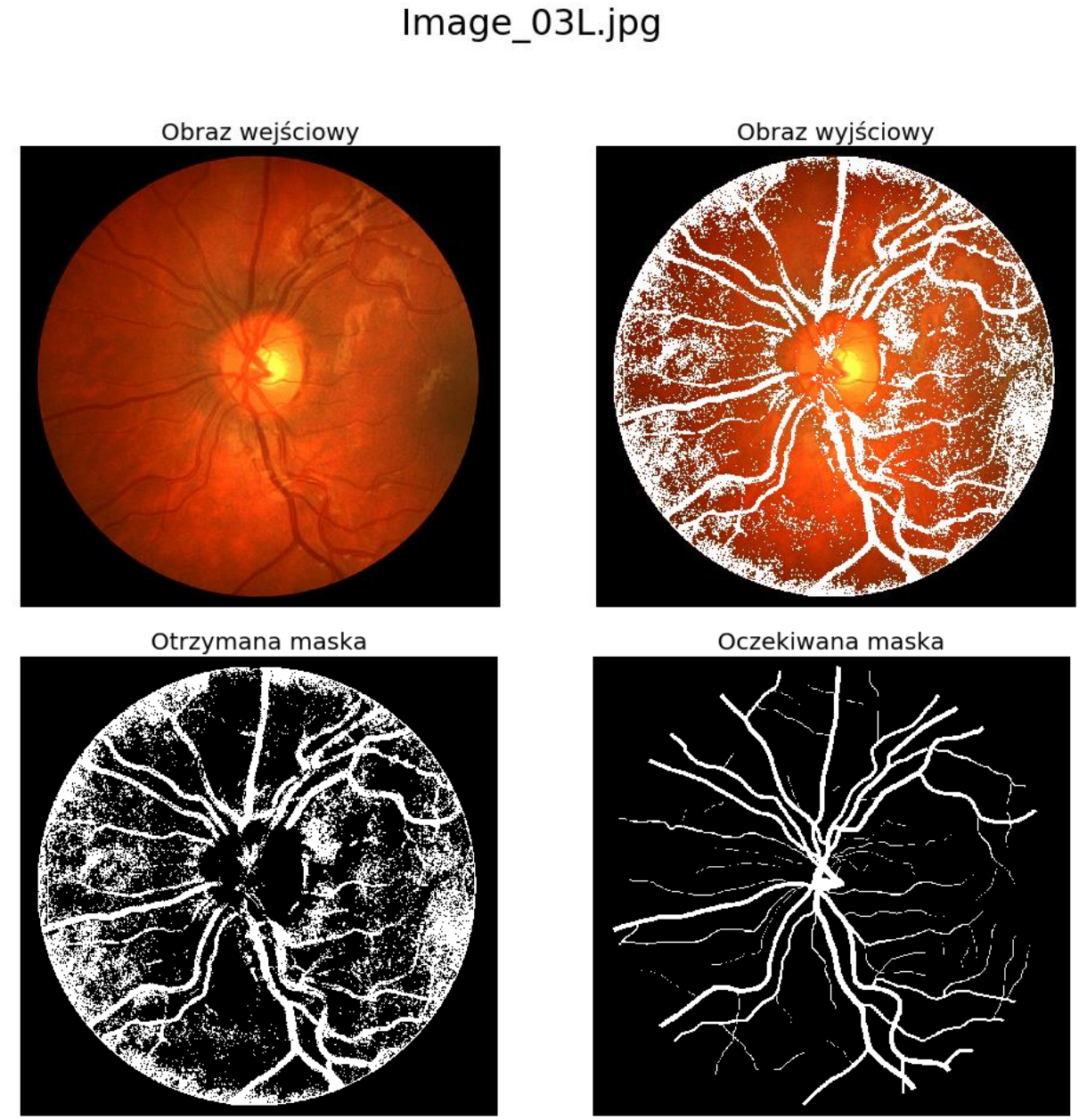
W tym przypadku mamy problem błędnego wyznaczania konturów tarczy oka jako naczynie. Widzimy że w ciemniejszych częściach występuje duży szum. Tym razem jasne centrum oka sprawia większe problemy

```
Statistics for Image_04L.jpg
Accuracy: 0.7456190382535156
Sensitivity: 0.7341114397633703
Specificity: 0.7470930877526535
Mean of sensitivity and specificity: 0.7406022637580119
```



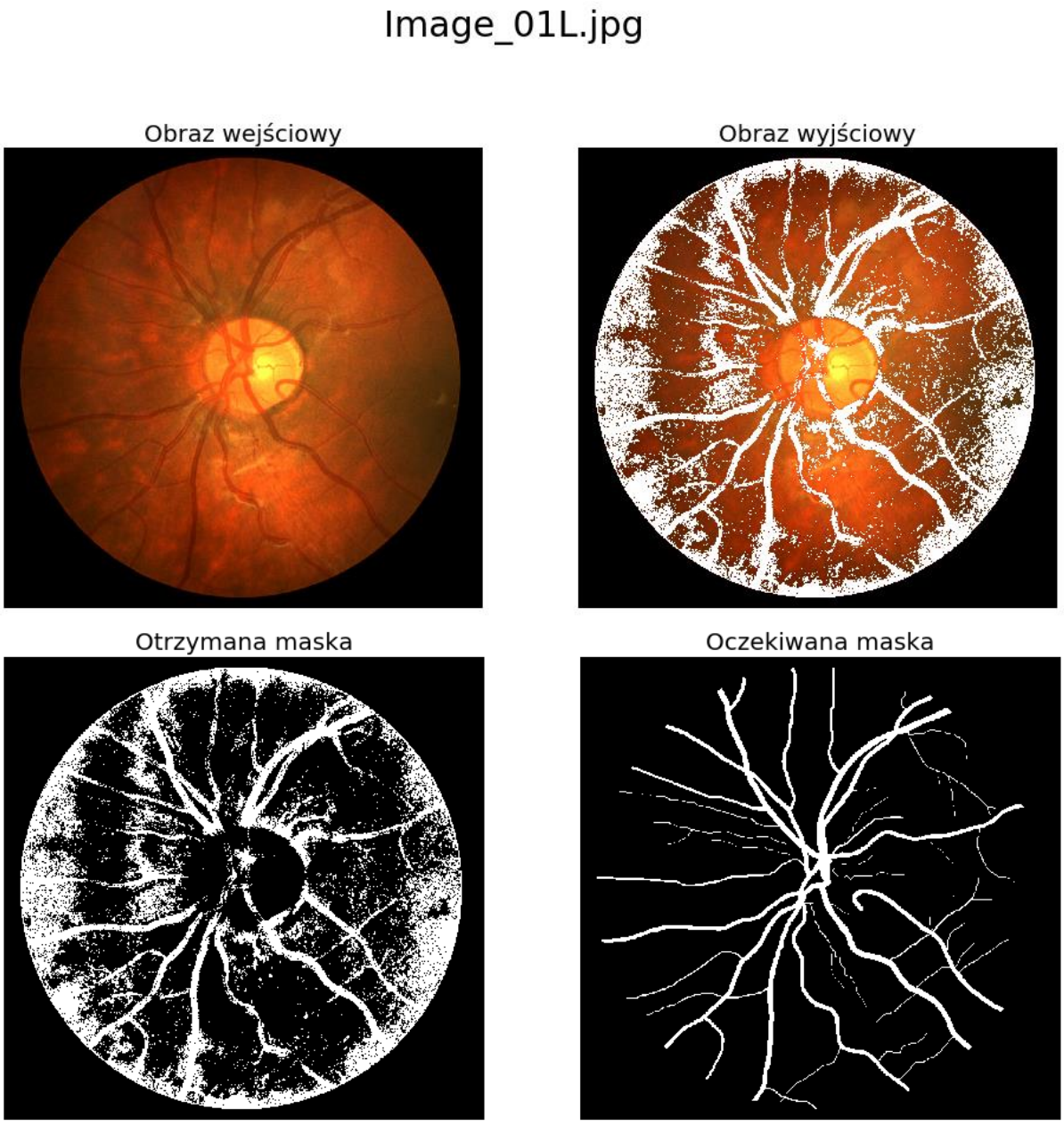
Ponownie widzimy problemy przy ciemniejszych częściach obrazka, głównie z prawej strony. Dobre wykrywanie grubszych naczyń.

```
Statistics for Image_03L.jpg
Accuracy: 0.7269460010842368
Sensitivity: 0.8103861837371072
Specificity: 0.7162821986630902
Mean of sensitivity and specificity: 0.7633341912000987
```



Tutaj możemy zauważyć ponowny problem w ciemnych obszarach, gdzie powstał szum, oraz lekkie problemy przy jasnym centrum oka.

```
Statistics for Image_01L.jpg
Accuracy: 0.7202672357070851
Sensitivity: 0.7655922353364089
Specificity: 0.7151888325009398
Mean of sensitivity and specificity: 0.7403905339186743
```



Ponownie widzimy szum w ciemnych miejscach i lekkie problemy przy jasnych obszarach.

Porównanie wyników działania klasyfikatora lasu:

Zbiorcze wyniki(średnie):
Accuracy = 0.75
Sensitivity = 0.77
Specificity = 0.74
Mean of sensitivity and specificity = 0.75

W przypadku klasyfikatora lasu uzyskana jakość obrazków jest gorsza niż w przypadku klasyfikatora odległościowego. Widzimy jednak usprawnienie w

przypadku czułości, ale za to lekkie pogorszenie w swoistości. Wyniki tych miar są jednak bardziej zbliżone do siebie, nie widzimy dużych różnic w pojedyńczych wynikach czułości i swoistości w porównaniu do ich średniej. Efekt nie był w pełni zadowalający, głównie przez duży szum w ciemnych obszarach, jednak cieszy polepszenie wyników w przypadku jasnych obszarów. Średnia statystyk ze wszystkich obrazków pokazują, że wyniki działania algorytmu są zadowalające – średnia każdej statystyki jest zbliżona do 75%.

5. Porównanie wyników działania algorytmów

   Możemy zauważyć, że nasze rozwiązanie spisuje się lepiej dla przetwarzania obrazu niż dla uczenia maszynowego. Otrzymane wyniki przetwarzania obrazów są o co najmniej 10% lepsze od uczenia maszynowego. Największym plusem działania przetwarzania obrazu w porównaniu z uczeniem maszynowym to brak powstawania szumu w obszarach krawędzi oka. Z drugiej jednak strony powstałe rozwiązania miały duże problemy ze znajdowaniem cieńkich naczyń, co stanowiło mniejszy problem dla uczenia maszynowego. Dużym problemem dla obu metod był kontur tarczy nerwu wzrokowego, w przypadku gdy ten obszar był zbyt ciemny.

   Działanie przetwarzania obrazu można by polepszyć, poprzez zastosowania lepszego przetwarzania wstępnego obrazu, który by lepiej eliminował zbyt ciemne i zbyt jasne obszary.

   Poprawienie tego wpłynełoby też na działanie uczenia maszynowego, jednak w tym przypadku lepszym rozwiązaniem mogłoby być dobranie lepszego zbioru treningowego, lub próba wykrywania fragmentów obrazka tak, aby w pełni znajdował się we wnętrzu oka. Spowodowałoby to jednak ograniczenie znajdowania naczyń przy konturach oka.