

Sprawozdanie z Informatyki w Medycynie

Temat: Oko (Human Retina)

Zastosowany język programowania oraz dodatkowe biblioteki

Python;

```
import skimage as ski
from skimage import color, filters
import matplotlib.pyplot as plt
import numpy as np
import sklearn.metrics as metrics
from PIL import Image
from scipy import ndimage as ndi
import cv2
from sklearn.model_selection import train_test_split
from scipy.stats import gmean
from typing import Literal
import copy
from scipy import stats
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from skimage import filters
from sklearn.metrics import accuracy_score
import joblib
from random import sample
import time
```

Opis zastosowanych metod

Przetwarzanie obrazów:

Ładowanie obrazu:

```
def loader_image(filename):  
    image = ski.io.imread(filename)  
    image = image.astype(np.float64)  
    image /= np.max(image)  
    return image
```

```
def tif_loader(mask_filename):  
    im = Image.open(mask_filename)  
    imarray = np.array(im)  
    return imarray
```

Contrast:

```
image = ski.exposure.rescale_intensity(image, in_range=(0.009, 0.015))
```

Filtracja Sato:

```
eye = color.rgb2gray(retina_source)  
x, y = filters.threshold_multiotsu(eye, classes=3)  
mask = (eye > x)  
vessels = filters.sato(eye, sigmas=range(1, 15)) * mask
```

Progowe znaczenia:

```
thresholded = filters.apply_hysteresis_threshold(vessels, 0.01, 0.03)  
labeled = ndi.label(thresholded)[0]
```

Wyświetlanie obrazu po przetwarzeniu:

```
plt.figure()  
plt.imshow(stan_image)  
plt.axis('off')  
plt.title('Human eye')  
plt.show()
```

Obliczanie parametrów jakości przetwarzania:

```
true_positive, false_positive, false_negative, true_negative = 0, 0, 0, 0
for x in range(width):
    for y in range(height):
        if binary_predictions[x, y]:
            if binary_ground_truth[x, y]:
                true_positive += 1
            else:
                false_positive += 1
        else:
            if binary_ground_truth[x, y]:
                false_negative += 1
            else:
                true_negative += 1

evaluation = { }
evaluation['accuracy'] = (true_positive + true_negative) / (true_negative +
false_negative + true_positive + false_positive)
evaluation['sensitivity'] = true_positive / (true_positive + false_negative)
evaluation['specificity'] = true_negative / (false_positive + true_negative)
evaluation['precision'] = true_positive / (true_positive + false_positive)
evaluation['g_mean'] = np.sqrt(evaluation['sensitivity'] * evaluation['specificity'])
evaluation['w_average'] = (evaluation['sensitivity'] + evaluation['specificity']) / 2
evaluation['confusion_matrix'] =
metrics.confusion_matrix(ground_truth_mask.flatten(), y_pred.flatten()).flatten()
```

To są metody dla ręcznej filtracji bez AI. Pozwalają na uzyskanie rysunków dobrej rozdzielczości;

Algorytm opis teoretyczny (bez niepotrzebnych rzeczy)

1. Na początku korzystamy z filtru Sato który jest pobierany z biblioteki
2. Regulujemy intensywność kolorów, żeby zrobić bardziej czytelne i przyjazne nasze oko (można pobawić się cyferkami wewnątrz bo mają wpływ na jakość)
3. Dodatkowo robimy filtrację która pozwala podzielić bardziej jasne pikseli i bardziej ciemne co ponownie zwiększy czytelność jakby nakłada markery na te pikseli co działa niezależnie od filtracji w kroku 1 i 2, to oznacza, że osobno dodatkowo zwiększy kontrast zdjęcia oka

Uczenie maszynowe

1. Podział obrazu na wycinki:

```
def get_fragments(input_image, patch_size=5):
    height, width = input_image.shape[:2]
    patches = []
    for i in range(0, height - patch_size + 1, patch_size):
        for j in range(0, width - patch_size + 1, patch_size):
            patch = input_image[i:i + patch_size, j:j + patch_size].copy()
            patches.append(patch)
    return patches

def get_mid_px(input_image, patch_size=5):
    height, width = input_image.shape[:2]
    piece = patch_size // 2
    pixels = []
    for i in range(piece, height - piece, patch_size):
        for j in range(piece, width - piece, patch_size):
            pixel = int(input_image[i, j])
            if pixel < 128:
                pixel = 0
            else:
                pixel = 255
            pixels.append(pixel)

    return pixels
```

2. Ekstrakcja cech wycinka:

```
def extract_image_features(flat_patch):
    features = []
    moments = [np.mean(flat_patch), np.var(flat_patch)]
    histogram = np.histogram(flat_patch, bins=10)[0]
    features.extend(moments)
    features.extend(histogram)

    return features
```

Krok po kroku AI

1. Wczytujemy obraz do uczenia:

```
image = cv2.imread(f'images/0' + str(i) + '_h.jpg')
mask = cv2.imread(f'images/0' + str(i) + '_h_mask.tif')
ex_mask = cv2.imread(f'images/0' + str(i) + '_h.tif')
```

2. Nasztywno ustawiamy rozmiar obrazu:

```
resized_image = cv2.resize(copy.deepcopy(image), (size_width, size_height))
resized_mask = cv2.resize(mask, (size_width, size_height))
res_ex_mask = cv2.resize(ex_mask, (size_width, size_height))
```

3. Odbieramy nasze odcinki dla uczenia korzystając z metod w punkcie 1, każdy z tych arrayów będzie miał specyficzną informację która jest potrzebna dla uczenia się naszej modeli:

```
spots = get_fragments(contrasted_image)
result_patches_pixels = get_mid_px(expected_result)
```

```
for patch in spots:
    flat_patch = patch.flatten().astype(float)
    features = extract_image_features(flat_patch)
    image_set.append(list(flat_patch) + features)
```

```
dataset.extend(image_set)
pixels_of_mask.extend(result_patches_pixels)
```

4. Dzielimy dane na uczące i testowe:

```
ax_tr, ax_t, ay_tr, ay_t = train_test_split(dataset, pixels_of_mask, test_size=0.3)
```

```
scaler = StandardScaler()
ax_tr_transformed = scaler.fit_transform(ax_tr)
ax_t_transformed = scaler.transform(ax_t)
```

5. Model Regresji Logistycznej:

Algorytm optymalizacji: "SAGA";
Ilość iteracji do uczenia się modelu: 5000;

```
model = LogisticRegression(solver='saga', max_iter=5000)
model.fit(ax_tr_transformed, ay_tr)
y_predicted = model.predict(ax_t_transformed)
```

6. Wyniki wstępnej oceny zbudowanego klasyfikatora:

```
true_positive, false_positive, false_negative, true_negative = 0, 0, 0, 0
for x in range(width):
    for y in range(height):
        if binary_predictions[x, y]:
            if binary_ground_truth[x, y]:
                true_positive += 1
            else:
                false_positive += 1
        else:
            if binary_ground_truth[x, y]:
                false_negative += 1
            else:
                true_negative += 1

evaluation = {}
evaluation['accuracy'] = (true_positive + true_negative) / (true_negative + false_negative +
true_positive + false_positive)
evaluation['sensitivity'] = true_positive / (true_positive + false_negative)
evaluation['specificity'] = true_negative / (false_positive + true_negative)
evaluation['precision'] = true_positive / (true_positive + false_positive)
evaluation['g_mean'] = np.sqrt(evaluation['sensitivity'] * evaluation['specificity'])
evaluation['w_average'] = (evaluation['sensitivity'] + evaluation['specificity']) / 2
evaluation['confusion_matrix'] = metrics.confusion_matrix(ground_truth_mask.flatten(),
y_pred.flatten()).flatten()
```

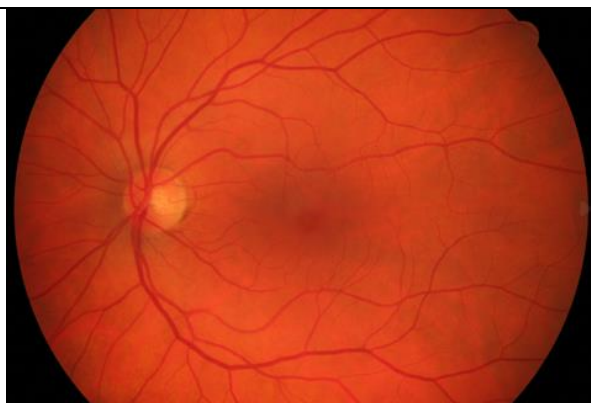
7. Uzasadnienie:

Prosta implementacja, szybko działa i nie wymaga dużej mocy komputera, jest efektywny do problemów w stylu – czy mamy naczynię krwionosną albo nie mamy – takie binarne problemy i nie tylko. Też jest używany w medycynie do przesiewowego badania dużej populacji.

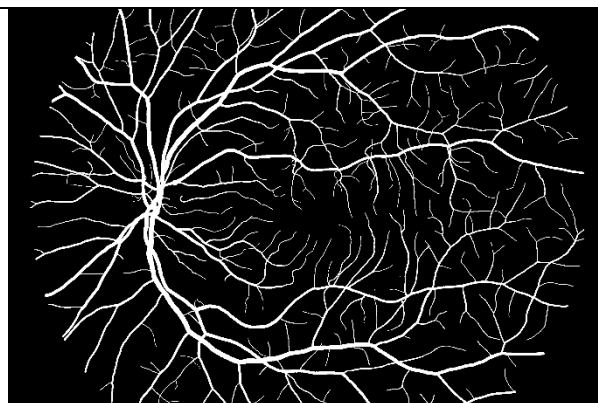
Wizualizacja wyników

UWAGA !

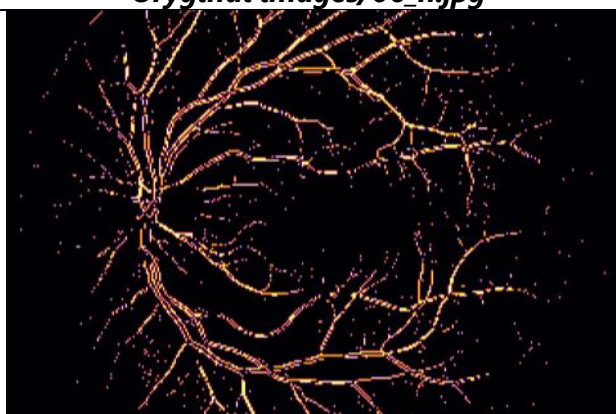
Nie znalazłem konkretnej informacji o tym, jakie obrazy muszą być umieszczone po badaniach, dlatego wybrałem: Oryginał oka i maski + obraz po przetwarzaniu maszynowym i po przetwarzaniu algorytmicznym. W razie wątpliwości albo problemów, że coś jest nie tak, proszę o kontakt przed wystawieniem oceny.



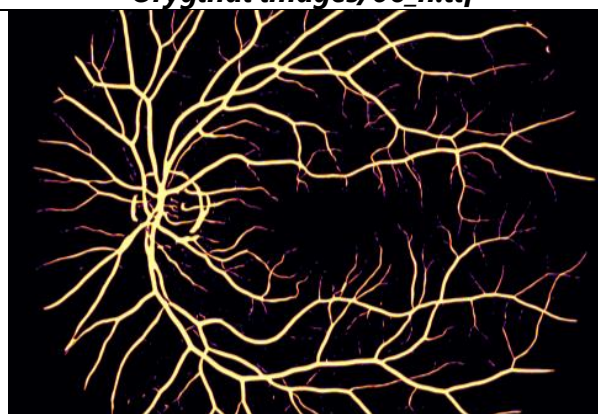
Oryginał images/06_h.jpg



Oryginał images/06_h.tif



Obraz po przetwarzaniu maszynowym 06



Obraz po przetwarzaniu 06


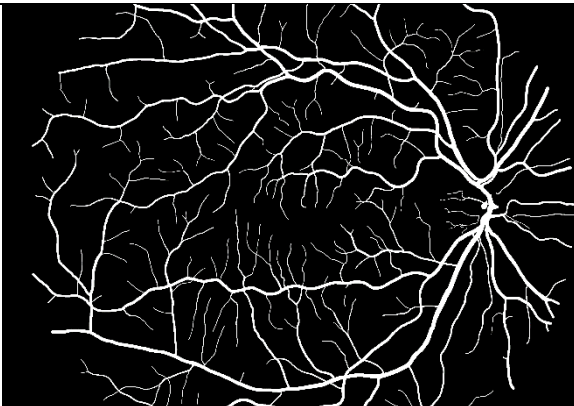
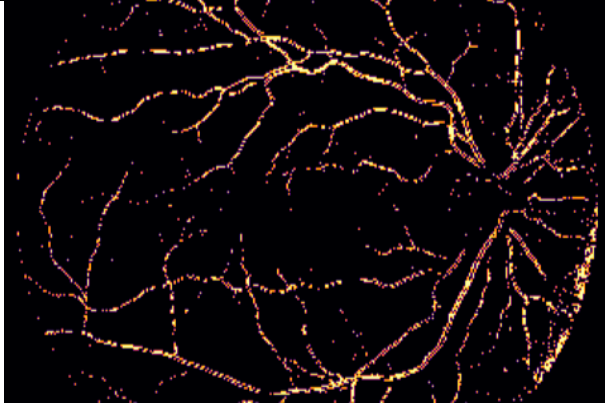
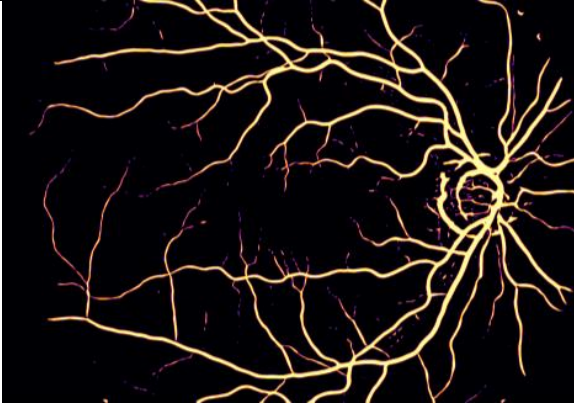
Charakterystyki przy użyciu ML:


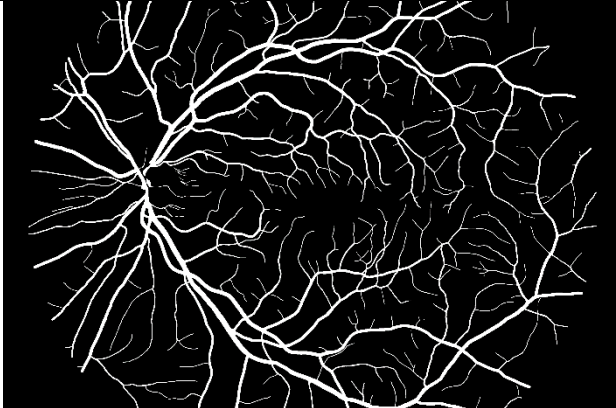
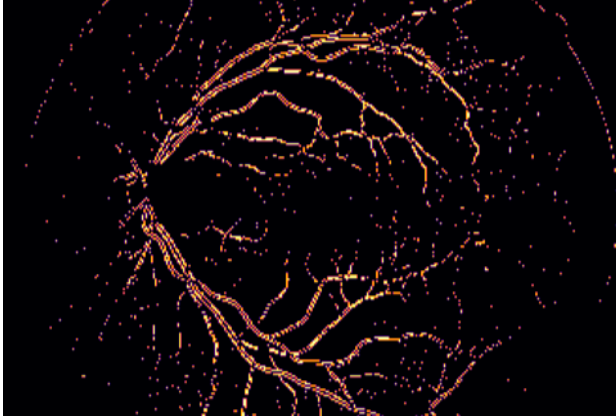
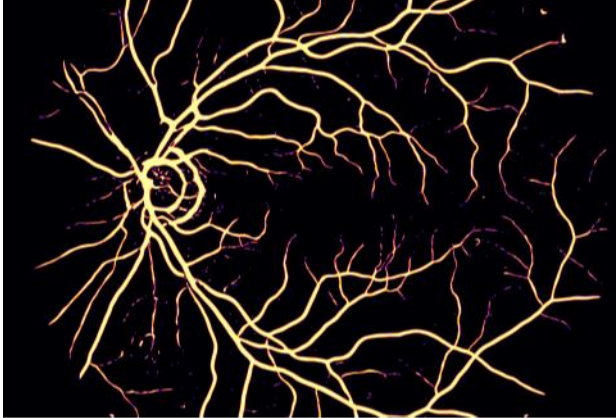
Accuracy 06: 91.09
Sensitivity 06: 37.42
Specificity 06: 97.99
Confusion 06:
[1623631 33357 133294 79718]


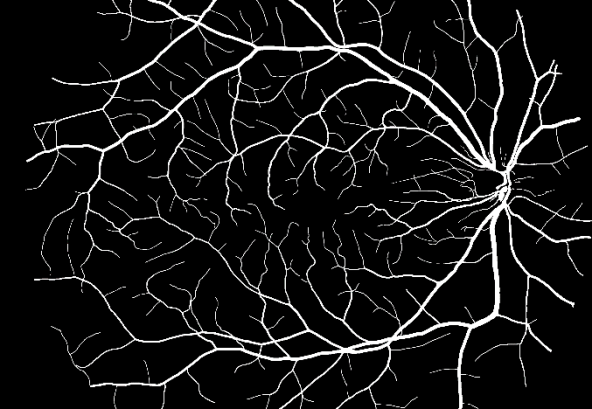
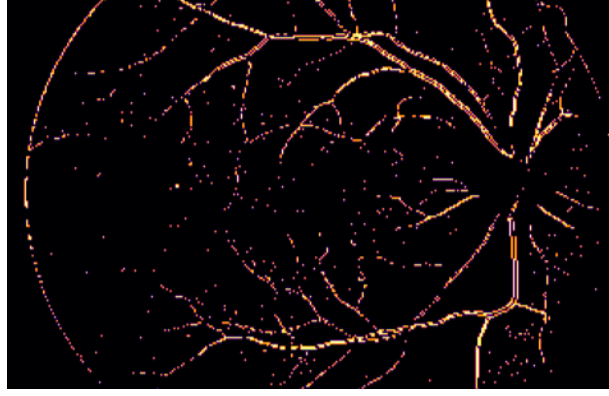
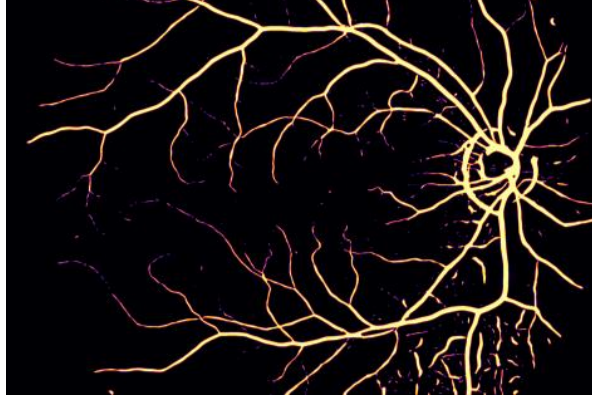
Charakterystyki przy zwykłej filtracji:

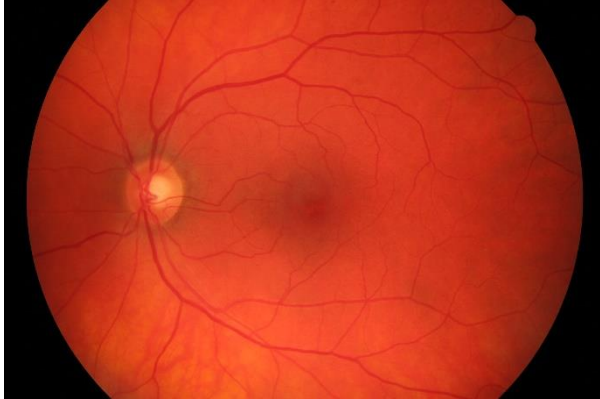
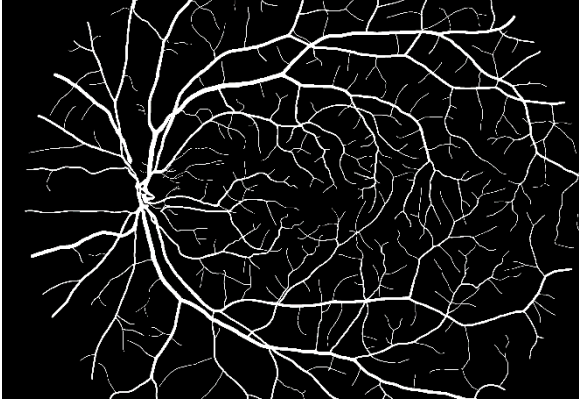
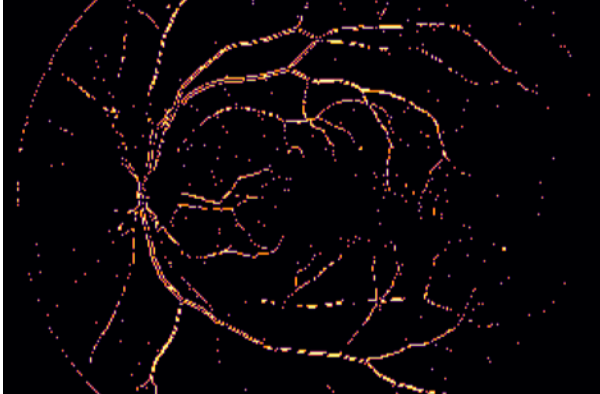
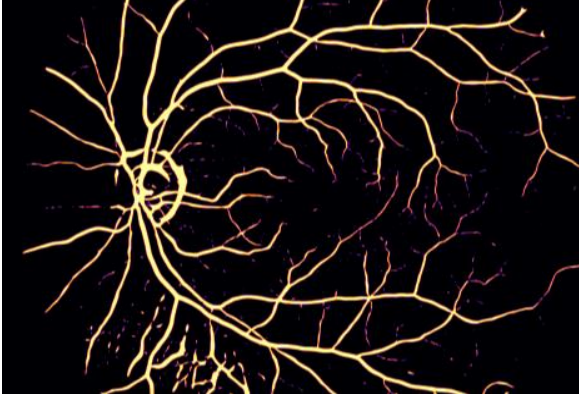
Accuracy score: 95.5
Sensitivity score: 81
Specificity score: 97.22
Confusion matrix: [7106580 248894
155180 674690]

Algorytm znacznie skuteczniej sklasyfikował prawdziwie pozytywne piksele obrazu, oprócz tego lepiej rozpoznał naczynie krwionośne natomiast ML działa gorzej z wyjątkiem tego, że lepiej wykrywa negatywne pikseli.

	
<p><i>Oryginał images/07_h.jpg</i></p>	<p><i>Oryginał images/07_h.tif</i></p>
	
<p><i>Obraz po przetwarzaniu maszynowym 07</i></p> <p>Charakterystyki przy użyciu ML: Accuracy 07: 92.41 Sensitivity 07: 39.06 Specificity 07: 98.30 Confusion matrix 07: [1655547 28604 113253 72596]</p>	<p><i>Obraz po przetwarzaniu 07</i></p> <p>Charakterystyki przy zwykłej filtracji: Accuracy score: 96.71 Sensitivity score: 74.32 Specificity score: 98 Confusion matrix: [7300057 156274 192283 536730]</p>
<p>Algorytm znacznie skuteczniej sklasyfikował prawdziwie pozytywne piksele, oprócz tego lepiej rozpoznał naczynie krwionośne natomiast ML działa gorzej, jednak łatwiej przechodzi test specificity, wykrywając negatywne pikseli.</p>	

	
<p><i>Oryginal images/08_h.jpg</i></p>	<p><i>Oryginal images/08_h.tif</i></p>
	
<p><i>Obraz po przetwarzaniu maszynowym O8</i></p>	<p><i>Obraz po przetwarzaniu O8</i></p>
<p>Charakterystyki przy użyciu ML: Accuracy O8: 90.81 Sensitivity O8: 29.83 Specificity O8: 98.50 Confusion O8: [1635737 24935 146888 62440]</p>	<p>Charakterystyki przy zwykłej filtracji: Accuracy score: 96.7 Sensitivity score: 77.14 Specificity score: 98 Confusion matrix: [7197839 165878 191368 630259]</p>
<p>Algorytm znacznie skuteczniej sklasyfikował prawdziwie pozytywne piksele, oprócz tego lepiej rozpoznał naczynia krwionośne natomiast ML działa gorzej, oprócz tego, że dobrze wykrywa piksele negatywne.</p>	

	
<p><i>Oryginał images/09_h.jpg</i></p>	<p><i>Oryginał images/09_h.tif</i></p>
	
<p><i>Obraz po przetwarzaniu maszynowym 09</i></p>	<p><i>Obraz po przetwarzaniu 09</i></p>
<p>Charakterystyki przy użyciu ML: Accuracy 09: 92.72 Sensitivity 09: 30.82 Specificity 09: 98.72 Confusion 09: [1683089 21866 114186 50859]</p>	<p>Charakterystyki przy zwykłej filtracji: Accuracy score: 96.93 Sensitivity score: 72.87 Specificity score: 98.6 Confusion matrix: [7363771 185105 180820 455648]</p>
<p>Algorytm znacznie skuteczniej sklasyfikował prawdziwie pozytywne piksele obrazu, oprócz tego lepiej rozpoznał naczynie krwionośne natomiast ML nieznacznie lepiej rozpoznał negatywne piksele.</p>	

	
<p><i>Oryginał images/10_h.jpg</i></p>	<p><i>Oryginał images/10_h.tif</i></p>
	
<p><i>Obraz po przetwarzaniu maszynowym 10</i></p>	<p><i>Obraz po przetwarzaniu 10</i></p>
<p>Charakterystyki przy użyciu ML: Accuracy 10: 91.85 Sensitivity 10: 24.60 Specificity 10: 99.17 Confusion 10: [1672560 14029 138290 45121]</p>	<p>Charakterystyki przy zwykłej filtracji: Accuracy score: 95.09 Sensitivity score: 73.21 Specificity score: 97.48 Confusion matrix: [7264593 215313 191378 514060]</p>
<p>Algorytm znacznie skuteczniej sklasyfikował prawdziwie pozytywne piksele, oprócz tego lepiej rozpoznał naczynie krwionośne natomiast ML działa lepiej do wykrywania negatywnych pikseli</p>	

Wnioski

Accuracy (dokładność): Oznacza odsetek poprawnie sklasyfikowanych przypadków ze wszystkich przypadków.

Sensitivity (czułość): Czuość to wskaźnik, który mierzy zdolność modelu do poprawnego wykrywania pozytywnych przypadków (naczyń krwionośnych) spośród wszystkich rzeczywiście pozytywnych przypadków.

Specificity (swoistość): Swoistość to wskaźnik, który mierzy zdolność modelu do poprawnego wykrywania negatywnych przypadków (brak naczyń krwionośnych) spośród wszystkich rzeczywiście negatywnych przypadków.

Algorytm tradycyjny wykazuje wyższą skuteczność w porównaniu do uczenia maszynowego (ML) w przypadku analizowanego problemu. Średni wynik dla algorytmu tradycyjnego wynosi około 75%, podczas gdy dla ML spada do około 35%. Wyniki te sugerują, że ML może mieć trudności lub wymaga dodatkowego dostosowania w celu skutecznego rozwiązania tego problemu.

Niemniej jednak, istnieją pewne przypadki, w których ML osiąga wyniki zbliżone do algorytmu tradycyjnego przy pozytywnych piskelach (czułość). To sugeruje, że ML może być skuteczniejszy w tych konkretnych przypadkach, gdzie może wykazywać wyższą czułość lub swoistość lepiej, niż algorytm.