Raport z projektu: Wykrywanie naczyń dna siatkówki oka	
Imię i nazwisko: **Mateusz Matkowski 145432** **Damian Mielczarek 145388**	Data wykonania (wysyłki) zadania: **11.05.2022**

A) Zastosowany język programowania oraz dodatkowe biblioteki:
Do napisania programu użyliśmy języka python w środowisku Jupyter Notebook. Dodatkowe biblioteki:

- Numpy
- Skimage
- Matplotlib
- Cv2
- Sklearn
- Time
- Imblearn
- Warnings
- Pickle
- Random

B) Metoda: Przetwarzanie obrazów

- Poszczególne kroki
  1. Wstępne przetwarzanie obrazu

```
13  #funkcja przeprowadza wstępne przetwarzanie obrazu
14  def initial_processing(input_bitmap):
15      sigma = 3.0
16
17      result = exposure.equalize_hist(input_bitmap)
18      result = gaussian(result, sigma=(sigma, sigma), truncate=3.5, multichannel=True)
19      result = unsharp_mask(result, radius=3, amount=1)
20      result = rgb2gray(result)
21
22      MIN = np.percentile(result, 0.0)
23      MAX = np.percentile(result, 100.0)
24      result = (result - MIN) / (MAX - MIN)
25
26      return result
```

Zdjęcie 1. Funkcja wstępnego przetwarzania wejściowego zdjęcia.

Na początku wyrównujemy histogram aby zwiększyć kontrast. Następnie stosujemy filtr Gaussa, żeby rozmazać obraz, po czym usunąć ostre krawędzie. Na samym końcu funkcji przenosimy obraz do odcieni szarości i normalizujemy go.

2. Zaawansowane przetwarzanie obrazu

```python
#funkcja przeprowadza główną część przetwarzania obrazu
def advanced_processing(initial_result, input_bitmap):
    white_eye = rgb2gray(input_bitmap.copy())
    white_eye[white_eye > 0.02] = 1.0
    white_eye = sobel(white_eye)
    white_eye = mp.dilation(white_eye, mp.disk(5))
    white_eye = mp.dilation(white_eye, mp.disk(5))

    result = sobel(initial_result)
    result[result[:,:] < 0.02] = 0
    result[result[:,:] > 0] = 1

    result = mp.dilation(result, mp.disk(3))

    for y in range(len(result)):
        for x in range(len(result[y])):
            if(white_eye[y][x] > 0.5):
                result[y][x] = 0.0

    return result
```

Zdjęcie 2. Funkcja zaawansowanego przetwarzania zdjęcia.

Początkowy etap przetwarzania w tej funkcji opiera się na przygotowaniu maski służacej do usunięcia okręgu, który powstaje na przejściu oko – tło. Po usunięciu okręgu przechodzimy do wykrywania konturów filrem Sobela oraz odfiltrowujemy powstały szum. Końcowy etap to poszerzenie krawędzi oraz usunięcie okręgu (wspomnianego wyżej) przy pomocy maski.

3. Finalne przetwarzanie obrazu

```python
#funkcja przeprowadza końcowe poprawki i zwraca binarną maskę
def final_processing(advanced_result):
    result = mp.dilation(advanced_result, mp.disk(5))

    result = mp.erosion(result, mp.disk(5))

    return result
```

Zdjęcie 3. Funkcja finalnego przetwarzania zdjęcia.

Ostatni etap to poszerzenie krawędzi jeszcze raz oraz ich “skurczenie”. Jest to swego rodzaju “trick”, który został nam pokazany na zajęciach z KCK.

- Uzasadnienie wybranego rozwiązania

Wybraliśmy takie rozwiązanie kierując się doświadczeniami pozyskanymi na przedmiocie KCK- na zajęciach opartych na przetwarzaniu obrazów.

C) Metoda: Uczenie Maszynowe

- Przygotowanie danych

```
25  def get_x(input_bitmap, cell_size, shift):
26      img_width = int((input_bitmap.shape[0]) / cell_size) * cell_size
27      img_height = int((input_bitmap.shape[1])/ cell_size) * cell_size
28
29      if(shift==1):
30          img_width = input_bitmap.shape[0]
31          img_height = input_bitmap.shape[1]
32          top, bottom, left, right = [int(cell_size/2)]*4
33          input_bitmap = cv2.copyMakeBorder(input_bitmap, top, bottom,
34                                            left, right, cv2.BORDER_CONSTANT)
35
36      parameters = []
37      for x in range(0, img_width, shift):
38          for y in range(0, img_height, shift):
39
40              cell = input_bitmap[x : x+cell_size, y : y+cell_size]
41
42              green_var = np.var(cell[:,:,1])
43
44              cell_gray = rgb2gray(cell)
45              moments = cv2.moments(cell_gray)
46              moments_list = list(moments.values())
47              hu_moments = cv2.HuMoments(moments)
48              hu_moments_2 =[]
49              for i in range(len(hu_moments)):
50                  hu_moments_2.append(hu_moments[i][0])
51              parameters.append([green_var, *moments_list, *hu_moments_2])
52
53      return parameters
54
```

Zdjęcie 4. Funkcja służąca do stworzenia wycinków obrazu I ekstracji wybranych statystyk.

Parametry służace nam do nauczenia klasyfikatora to:

- Kanał zielony (wpływa on najbardziej na decyzję – sprawdzone empirycznie)
- Momenty centralne
- Momenty Hu

- Wstępne przetwarzanie zbioru uczącego

```
3  def RescaleImage(frame, scale = 0.20):
4      width = int(frame.shape[1] * scale)
5      height = int(frame.shape[0] * scale)
6      dimensions = (width, height)
7      return cv2.resize(frame, dimensions, interpolation=cv2.INTER_AREA)
8
```

Zdjęcie 5. Funkcja RescaleImage, służaca do zmiany wielkości przetwarzanego zdjęcia – wymagana, ponieważ nasz klasyfikator nie byłby zdolny do predykcji w skończonym czasie.

```
117         y_test[y_test<=100] = 0
118         y_test[y_test>0] = 1
```

Zdjęcie 6. Element poprawiający maskę pierwotną, użytą do testowania, po przetworzeniu jej funkcją RescaleImage.

```
    for x in range(begin_from, img_width-begin_from, shift):
        for y in range(begin_from, img_height-begin_from, shift):
            if(expert_mask[x][y]>100):
                pixels.append(1)
            else:
                pixels.append(0)

    return pixels
```

Zdjęcie 7. Element poprawiający maskę pierwotną, użytą do trenowania, po przetworzeniu jej funkcją RescaleImage.

- Zastosowane metody uczenia maszynowego I wybrane parametry

```
    sampler = RandomUnderSampler(sampling_strategy=1, random_state=200)
    x_train, y_train = sampler.fit_resample(x_train, y_train)
```

Zdjęcie 8. Sampler służacy do odsiania próbek uczących w zbiorze treningowym.

```
def KNN_learn(x_train, y_train):
    knn_model = KNeighborsClassifier(n_neighbors=13, n_jobs=-1)
    knn_model.fit(x_train, y_train)
    knnPickle = open('knnpickle_file', 'wb')
    pickle.dump(knn_model, knnPickle)


def KNN_predict(x_test, y_test, img_width, img_height):
    knn_model = pickle.load(open('knnpickle_file', 'rb'))
    x_predict=knn_model.predict(x_test)

    x_predict = np.array(x_predict)
    shape = (img_width, img_height)
    mask = x_predict.reshape(shape)
    plt.figure()
    io.imshow(mask, cmap='gray')

    statistical_analysis(mask, y_test)
```

Zdjęcie 9. Zastosowany klasyfikator, metody uczenia oraz zapis modelu do pliku.

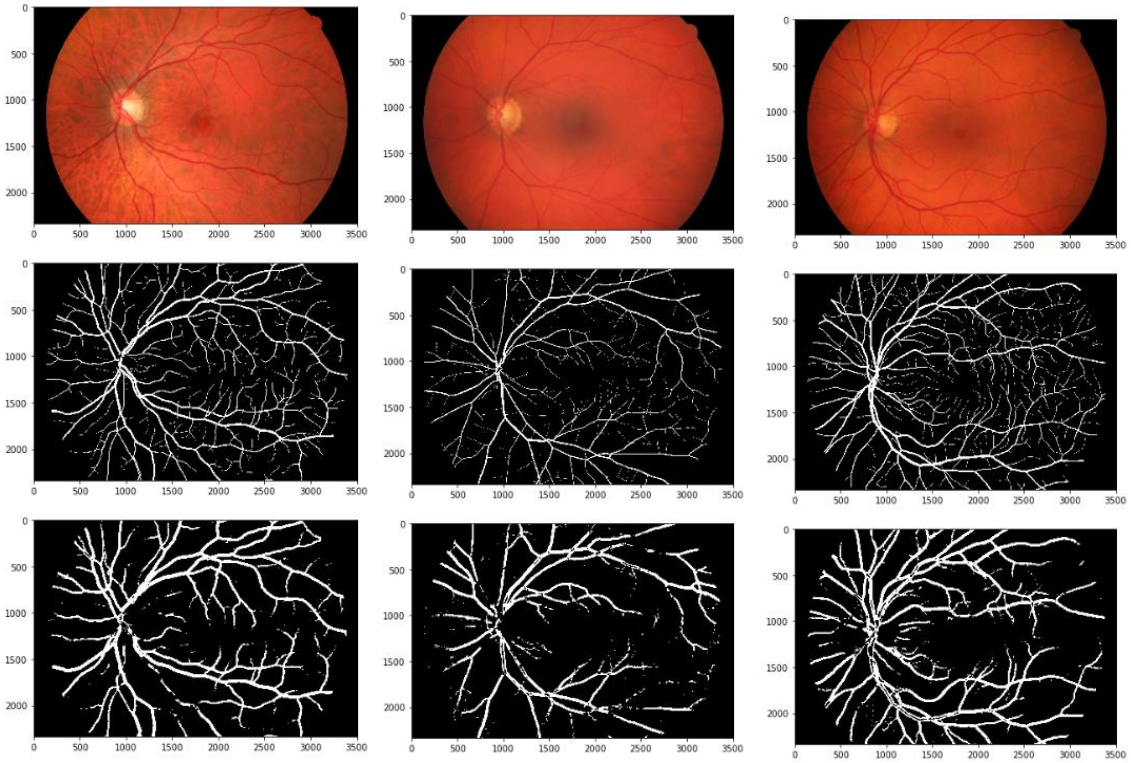
Zastosowaliśmy parameter liczby sąsiadów równy 13 (wybrany empirycznie). Dodatkowo parameter n_jobs = -1 pozwala na użycie wielowątkowości procesora co przyspiesza uzyskanie wyniku. Przy użyciu modułu “pickle” byliśmy w stanie zapisać nasz model do pliku, co pozwoli na poźniejsze jego wykorzystanie, jak I podział kodu na osobne moduły.

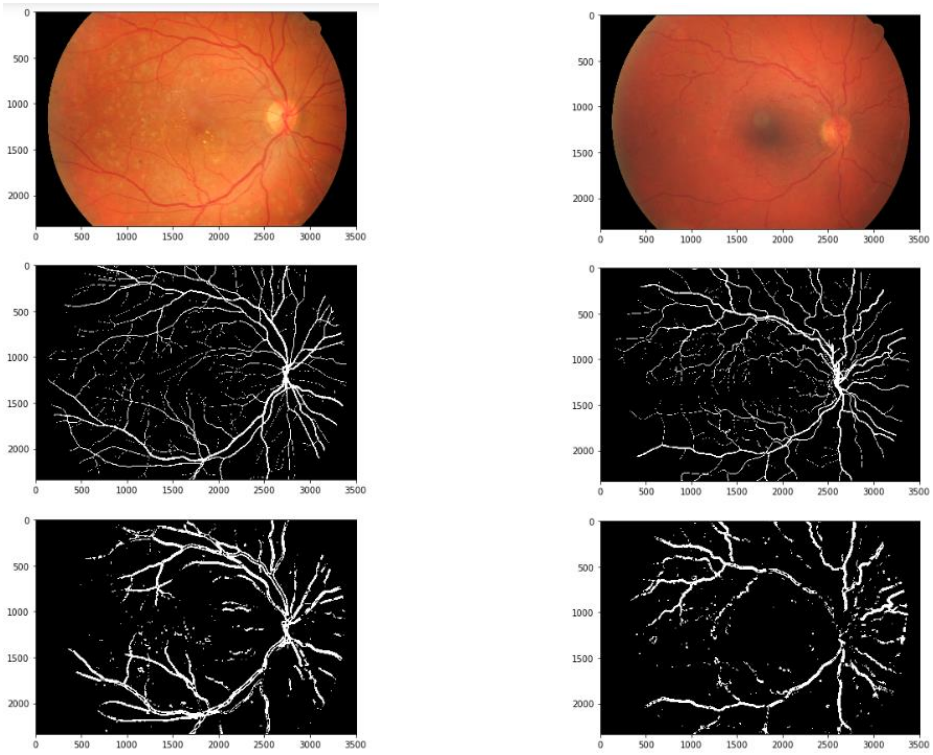
- Uzasadnienie wybranego rozwiazania

Wybraliśmy ten klasyfikator, ponieważ podczas studiowania jego działania wydał nam się najciekawszy oraz dość łatwy w implementacji. Jednak, nie nadaje się on gdy zdjęcia są bardzo duże (3504x2336) ze względu na czas predykcji – nie jest w stanie zrobić zaledwie 5 zdjęć w czasie <= 10h.

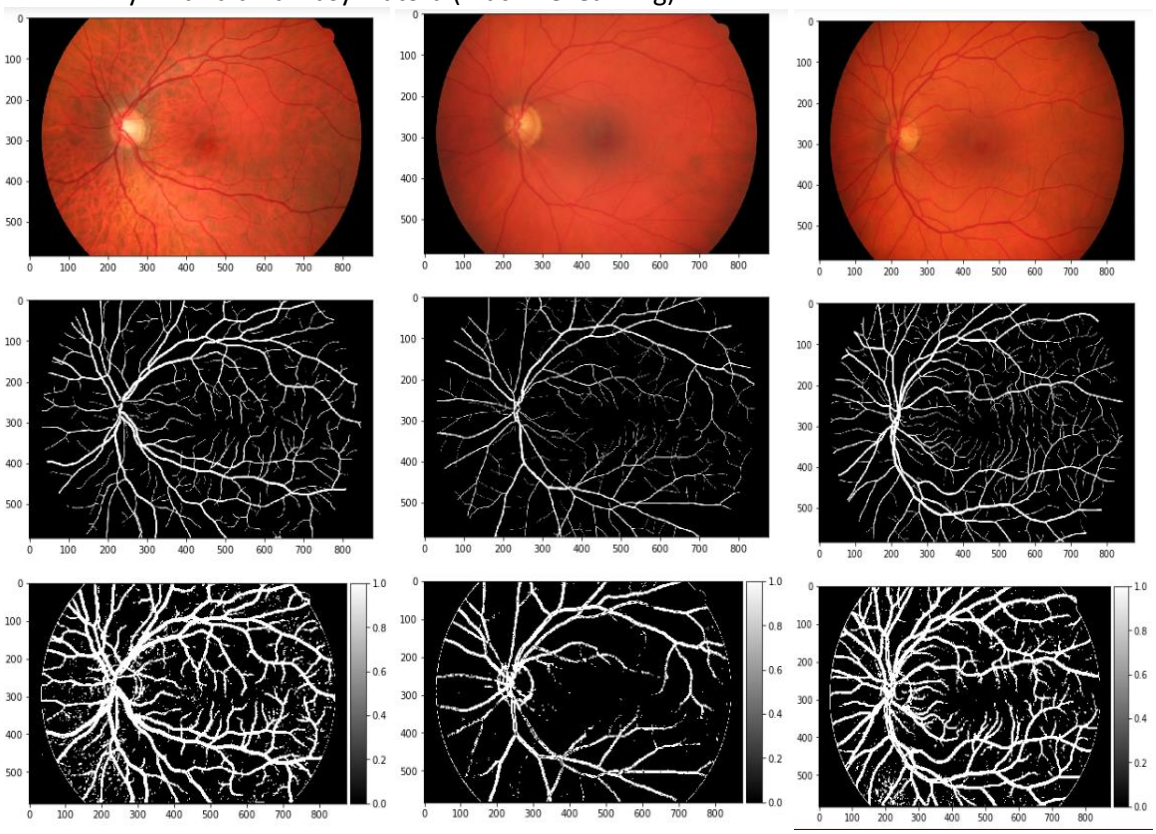
D) Wizualizacja wyników działania program

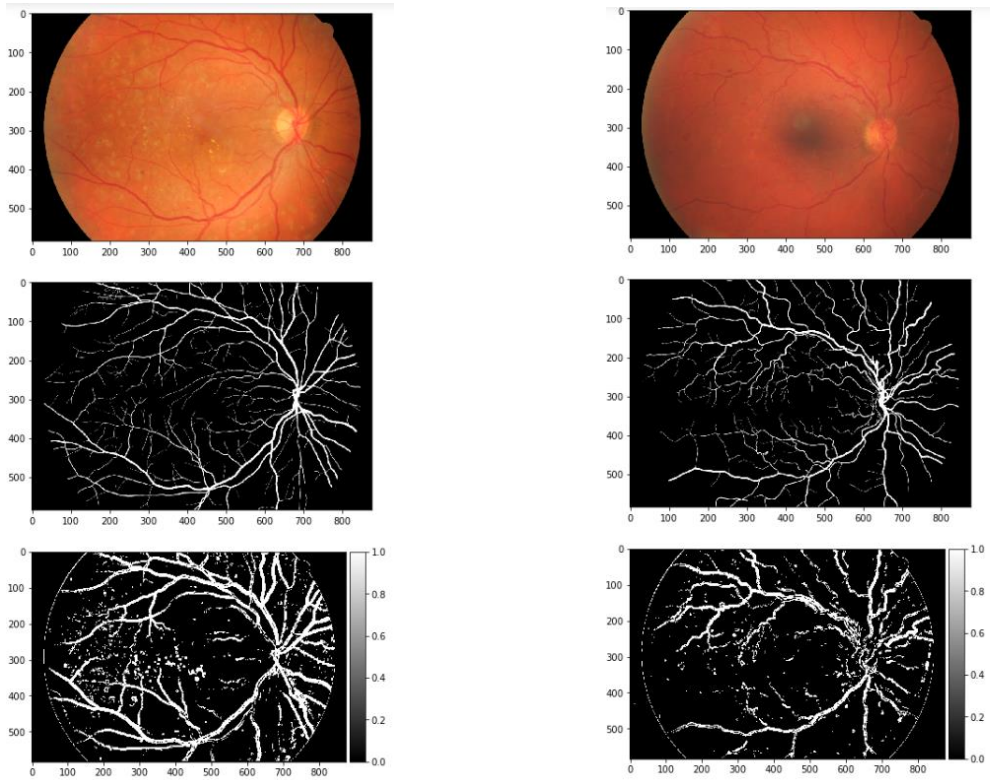
- Wynik działania przetwarzania obrazu





- Wynik działania klasyfikatora (machine learning)





E) Analiza wyników działania program

- Tabela porównania miar oceny uzyskanych wyników

	Accuracy	Sensivity	Specyfity	Geometric Mean
Zdjęcie 1 - IP	0.92	0.76	0.94	0.84
Zdjęcie 1 - ML	0.83	0.96	0.82	0.89
Zdjęcie 2 - IP	0.92	0.63	0.94	0.77
Zdjęcie 2 - ML	0.91	0.82	0.91	0.87
Zdjęcie 3 - IP	0.90	0.71	0.92	0.81
Zdjęcie 3 - ML	0.83	0.96	0.82	0.89
Zdjęcie 4 - IP	0.91	0.49	0.94	0.68
Zdjęcie 4 - ML	0.87	0.78	0.87	0.82
Zdjęcie 5 - IP	0.96	0.44	0.96	0.64
Zdjęcie 5 - ML	0.91	0.60	0.92	0.74

Tabela 1. Porównanie miar oceny uzyskanych wyników.

Legenda:
Zielony – IP – wynik przetwarzania obrazu
Czerwony – ML – wynik machine learningu

Z powyższej tabeli na podstawie miary Accuracy można wywnioskować, iż sposób oparty na przetwarzaniu obrazów radzi sobie lepiej – należy jednak pamiętać (co można zauważyć na wynikach w poprzednim podpunkcie), że na miarę tą duży wpływ ma okrąg (o którym mowa w punkcie B.2). W prostym przetwarzaniu obrazu jesteśmy w stanie pozbyć się go w miarę mało inwazyjny sposób, natomiast w przypadku machine learningu nie mamy wpływu na jego powstanie bądź usunięcie. Natomiast wizualnie wynik po machine learningu wygląda lepiej. Jesteśmy w stanie zauważyć nawet mniejsze naczynia krwionośne, które są usuwane w przypadku prostego przetwarzania.
Biorąc pod uwagę miarę średniej geometrycznej (na którą składają się miary Sensivity oraz Specyfity) również możemy dostrzec, że machine learning lepiej radzi sobie z detekcją naczyń krwionośnych. Należy jednak zdawać sobie sprawę z tego, że zdjęcia przetwarzane przy uzyciu machine learningu zostały sztucznie zmniejszone w celu osiągniecia rezultatów w skończonym czasie – jednak gdyby posiadać odpowiednio dużo czasu, miary te na oryginalnym zdjęciu z pewnością osiągnęłyby lepsze wyniki.