Raport - Wykrywanie naczyń dna siatkówki oka

1. **Skład grupy.**
   - Aleksandra Bamberska 156021
   - Adam Biernacki 155934

2. **Zastosowany język programowania oraz dodatkowe biblioteki.**
   - Język: Python 3.11.0
   - Dodatkowe biblioteki:
     - Streamlit – framework który wykorzystaliśmy do stworzenia interfejsu użytkownika
     - os – praca z plikami i ścieżkami w systemie operacyjnym
     - pandas - Tworzenie i wyświetlanie tabel z metrykami
     - numpy – operacje numeryczne
     - Pillow – Przetwarzanie obrazów
     - OpenCV-python - Zaawansowane przetwarzanie obrazów
     - Scikit-image (0.24.0) - Implementacja filtrów przetwarzania obrazu i ekstrakcja cech
     - Scikit-learn (1.6.1) - Klasyfikacja i ewaluacja modelu klasycznego (Random Forest)
     - Imbalanced-learn - Balansowanie zbioru uczącego
     - Joblib - Zapisywanie i wczytywanie modeli klasyfikacyjnych (Random Forest)
     - Matplotlib - wizualizacja wyników
     - Torch - Budowa i trenowanie modelu UNet
     - Segmentation_models_pytorch – gotowa implementacja architektury UNet

3. **Opis zastosowanych metod:**
   - **Przetwarzanie obrazów**

```python
def generate_fov_mask_by_brightness(image, threshold=41):  12 usages
    # 1. Skala szarości
    if image.ndim == 3:
        gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
    else:
        gray = image.copy()

    # 2. Proste progowanie jasności
    mask = (gray > threshold)

    return mask
```

Na podstawie obrazu wejściowego generujemy maskę FOV używający wartości jasności danego pixela, maska ta która określa nam obszar, który będzie poddany filtrowaniu Frangiego.

Następnie obraz wejściowy poddawany jest wstępnemu przetworzeniu.

```python
def preprocess_image(image):  11 usages
    green_channel = image[:, :, 1]

    clipped = np.clip(green_channel, a_min: 10, a_max: 245)

    # Rozmycie Gaussa (usunięcie szumu)
    blurred = cv2.GaussianBlur(clipped, ksize: (9, 9), sigmaX: 0)

    # Wyostrzenie (maskowanie nieostrości)
    sharpened = cv2.addWeighted(blurred, alpha: 1.5, blurred, -0.5, gamma: 0)

    # Normalizacja histogramu lokalna (CLAHE)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    enhanced = clahe.apply(sharpened)

    return enhanced
```

Podczas wstępnego przetwarzania wyciągamy ze zdjęcia tylko kanał zielony, ponieważ, w nim najlepiej są widoczne naczynia. Wykorzystujemy rozmycie Gaussa w celu usunięcia szumu. Następnie obraz poddany jest wyostrzeniu i na koniec wykonana jest normalizacja histogramu przy użyciu CLAHE. Parametry do każdego etapu zostały wybrane metodą prób i błędów, w celu wypracowania parametrów bliskich optymalnym.

```python
def apply_frangi_filter(image, fov_mask=None):  3 usages
    filtered = frangi(image, sigmas=np.arange(1, 5, 0.5), black_ridges=True)
    filtered = filtered / filtered.max()

    if fov_mask is not None:
        filtered = filtered * fov_mask

    binary_mask = np.where(filtered > 0.03, 1, 0)
    return binary_mask
```

W funkcji apply_frangi_filter wykonywana jest aplikacja filtru Frangi na wstępnie przetworzony obraz. Wykonujemy filtrację, następnie normalizujemy ją względem największej wartości. Jeśli funkcja zostaje wywołana z maską FOV, usuwamy wszystko co jest poza nią – ustawiamy to jako tło. Kolejnym krokiem jest wybranie odpowiednich wartości już z filtrowanego obrazu aby utworzyć maskę binarną. Wartość 0.03, również została wybrana tą samą metodą co parametry w poprzedniej funkcji, może być ona zła dla innych obrazów, które nie znajdują się w tej bazie danych i musiałby być dostosowana do obrazów wejściowych.

```python
def postprocess_mask(binary_mask):  4 usages
    if binary_mask.dtype != np.uint8:
        binary_mask = (binary_mask * 255).astype(np.uint8) if binary_mask.max() <= 1 else binary_mask.astype(np.uint8)

    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, ksize: (3, 3))
    opened = cv2.morphologyEx(binary_mask, cv2.MORPH_OPEN, kernel, iterations=1)

    return opened/255
```

Opcją w interfejsie jest poddanie wygenerowanej maski binarnej końcowemu przetworzeniu, zalecane jest włączenie tej funkcji, ponieważ w testowanych obrazach zwiększała wartość dokładności względem maski eksperckiej.

Funkcja ta jest bardzo prosta, ale też wystarczająca, usuwamy w niej tylko małe elementy szumu, który pozostał po aplikacji filtrowania.

Przykład działania:

Bez włączonego końcowego przetwarzania

Z włączonym końcowy przetwarzania

Mimo że miary są dosyć zbliżone, jednak korzystając z oceny wizualnej widać że maska binarna po końcowym przetworzeniu pozbawiona jest pozostałości z ramki wokół siatki oka oraz w samej siatce oka usunięte został elementy (większość), które były szumem po filtrowaniu.

- Uczenie maszynowe – klasyfikator

Podczas generowania klasyfikatora wczytujemy obrazy, które nie są użyte do późniejszego testowania klasyfikatora lub modelu UNet. Poddajemy je wstępnemu przetworzeniu (funkcja opisana wyżej), generujemy maskę FOV oraz wczytujemy maskę ekspercką.

```
images = []
masks = []
fov = []

for fname in train_files:
    image_path = os.path.join(IMAGE_DIR, fname)
    mask_path = os.path.join(MASK_DIR, os.path.splitext(fname)[0] + ".ah.ppm")

    if not os.path.exists(mask_path):
        continue

    image = cv2.imread(image_path)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    image_np = np.array(image)

    preprocessed = preprocess_image(image_np)

    mask_fov = generate_fov_mask_by_brightness(image_np)

    expert = Image.open(mask_path).convert("L")
    expert_mask = (np.array(expert) > 127).astype(np.uint8)

    images.append(preprocessed)
    masks.append(expert_mask*255)

    fov.append(mask_fov*255)
```

Po wczytaniu wszystkich obrazów wykonujemy na nich wyznaczenie wycinków obrazu oraz ekstrakcje ich cech.

```
X = []
y = []

for i in range(train_len):
    h, w = masks[i].shape
    process_with_fov(h, w, fov[i], masks[i], extract_labels, y)

    h, w = images[i].shape
    process_with_fov(h, w, fov[i], images[i], extract_patch_features, X)
```

W funkcji process_wih_fov generujemy wycinki (patch), ale tylko wtedy, kiedy dany piksel znajduje się w obrębie wygenerowanej maski FOV.

```
def process_with_fov(h, w, fov, data, fun, output):  2 usages
    for y in range(2, h - 2):
        for x in range(2, w - 2):
            if fov[y, x]:
                patch = data[y - 2 : y + 3, x - 2 : x + 3]
                output.append(fun(patch))
```

W funkcji tej przechodzimy po całym obrazie, jeśli piksel znajduje się w obrębie maski FOV, wykonujemy ekstrakcję cech z tego wycinka.

```python
def extract_patch_features(patch):  2 usages
    features = [np.mean(patch), np.var(patch)]
    moments = moments_central(patch)
    hu = moments_hu(moments)
    features.extend(hu)
    return np.array(features)


def extract_labels(patch):  1 usage
    if patch[2, 2] > 127:
        return 1
    else:
        return 0
```

Funkcja extract_patch_features jest dla wycinków z obrazu. Wyciągamy z niego średnią wartość, wariację oraz momenty Hu.

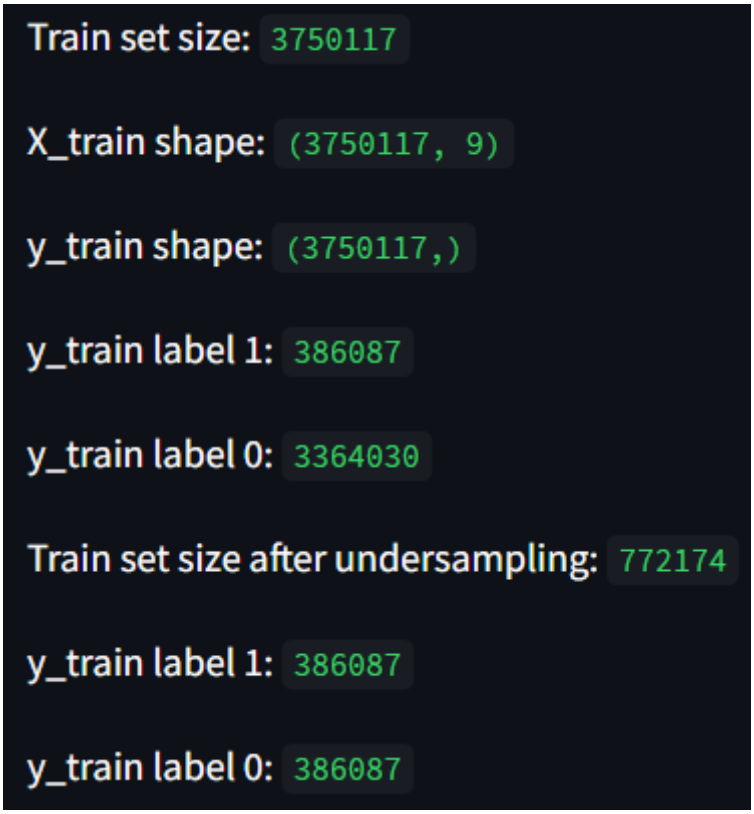
Funkcja extract_labels tak naprawdę sprawdza nam tylko czy środkowy element z wycinka maski eksperckiej, który ma wielkość 5x5, jest naczyniem.

Kolejnym krokiem jest podział zbioru danych na dane uczące i dane testowe.

```python
X_train, X_test, y_train, y_test = train_test_split( *arrays: X, y, test_size=0.2, random_state=42)

st.write("Train set size: ", len(X_train))
st.write('X_train shape: ', np.array(X_train).shape)
st.write('y_train shape: ', np.array(y_train).shape)

st.write('y_train label 1: ', np.count_nonzero(y_train))
st.write('y_train label 0: ', len(y_train) - np.count_nonzero(y_train))

rus = RandomUnderSampler(sampling_strategy=1, random_state=42)
X_train, y_train = rus.fit_resample(X_train, y_train)

st.write("Train set size after undersampling: ", len(X_train))
st.write('y_train label 1: ', np.count_nonzero(y_train))
st.write('y_train label 0: ', len(y_train) - np.count_nonzero(y_train))
```

Dane są podzielone w proporcji 8:2 czyli 80% to dane uczące a 20% to dane testowe.

Dane uczące są poddane undersampling-owi do zrównoważenia rozkładu klas w zbiorze uczącym.

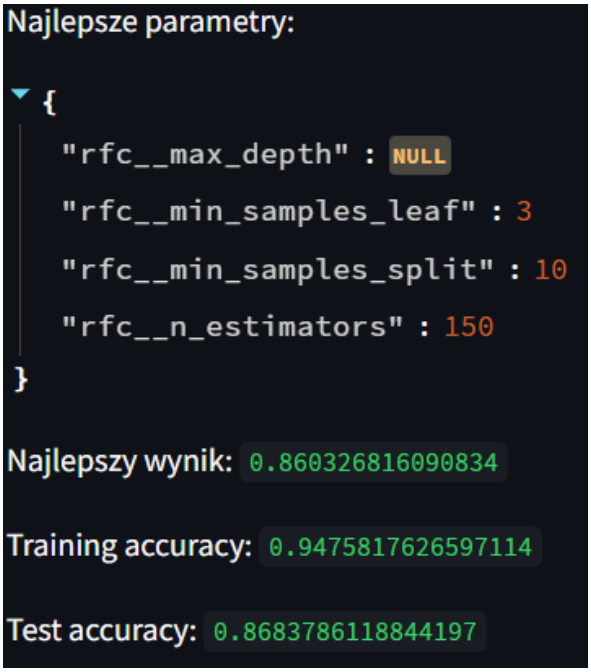


Definiujemy nasz model w postaci RandomForestClassifier() oraz jego możliwe parametry.

```
model = RandomForestClassifier()

pipeline = Pipeline([
    ('rfc', model)
])

param_grid = {
    'rfc__n_estimators': [5, 150],
    'rfc__max_depth': [None, 30],
    'rfc__min_samples_leaf': [0, 3],
    'rfc__min_samples_split': [1, 10]
}
```

Definiujemy obiekt GridSearchCV oraz dopasowujemy model do danych i następnie wyświetlamy najlepsze parametr oraz wynik testu krzyżowego. Jako ostatnie wyświetlane są wyniki accuracy dla zbioru treningowego oraz dla zbioru testowego. Po tych operacjach klasyfikator zostaje zapisany do pliku, aby móc go użyć w naszej aplikacji.

```
# Definiowanie obiektu GridSearchCV
grid_search = GridSearchCV(pipeline, param_grid=param_grid, n_jobs=-1)
grid_search.fit(X_train, y_train)
```



Predykcja maski binarnej na podstawie klasyfikatora:

Dzielimy obraz wejściowy na wycinki i wyciągamy z nich średnią wartość, wariację oraz momenty Hu. Na podstawie tych wartości klasyfikator dokonuje predykcji maski binarnej. Ostatnim elementem jest filtrowanie tej maski przez FOV obrazu wejściowego, aby uzyskać lepszy wynik.

```python
def predict_mask_from_model(image, model, fov_mask, patch_size=5):
    h, w = image.shape
    mask_pred = np.zeros( shape: (h, w), dtype=np.uint8)
    features = []

    coords = []
    for y in range(patch_size//2, h - patch_size//2):
        for x in range(patch_size//2, w - patch_size//2):
            if fov_mask[y, x]:
                patch = image[y - 2:y + 3, x - 2:x + 3]
                features.append(extract_patch_features(patch))
                coords.append((y, x))

    if not features:
        return mask_pred  # wszystko zostaje czarne

    features = np.array(features)
    predictions = model.predict(features)

    for (y, x), pred in zip(coords, predictions):
        if pred == 1:
            mask_pred[y, x] = 1

    return mask_pred * fov_mask
```

- Uczenie maszynowe – Unet

Wstepne przygotowanie danych, jest bardzo podobne jak w poprzednim przypadku, jednak tutaj nie używamy wstępnego przetworzenia obrazu oraz usunięcia elementów poza FOV'em, zdecydowaliśmy się to pominąć, aby model uczył się na obrazach wejściowych bez żadnych zmian, co poprawia też prędkość tego algorytmu.

```
images = []
masks = []

for fname in train_files:
    img_path = os.path.join(IMAGE_DIR, fname)
    mask_path = os.path.join(MASK_DIR, os.path.splitext(fname)[0] + ".ah.ppm")

    if not os.path.exists(mask_path):
        continue

    image = cv2.imread(img_path)

    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB) / 255

    mask = Image.open(mask_path).convert("L")
    mask = (np.array(mask) > 127).astype(np.uint8)

    images.append(image)
    masks.append(mask)
```

Kolejnym krokiem jest podział zbioru danych na dane uczące, walidujące oraz testowe. Są one podzielone w proporcji 8:1:1.

```
X_train_val, X_test, Y_train_val, Y_test = train_test_split( *arrays: images, masks, test_size=0.1, random_state=42)
X_train, X_val, Y_train, Y_val = train_test_split( *arrays: X_train_val, Y_train_val, test_size=1/9, random_state=42)
```

```
X_train shape: torch.Size([11, 3, 605, 700])
Y_train shape: torch.Size([11, 1, 605, 700])
X_valid shape: torch.Size([2, 3, 605, 700])
Y_valid shape: torch.Size([2, 1, 605, 700])
X_test shape: torch.Size([2, 3, 605, 700])
Y_test shape: torch.Size([2, 1, 605, 700])
```

Pierwsza wartość (2 lub 11) to liczba plików, które zostały przypisane do danego zbioru.

Druga wartość (3 lub 1) to liczba klas danych wejściowych, to znaczy czy w RGB czy w skali szarości.

Natomiast trzecia i czwarta wartość to wielkość obrazów.

Kolejnym krokiem jest definicja modelu, korzystamy tutaj już z zaimplementowanej sieci UNet z biblioteki segmentation_models_pytorch, która działa na bibliotece PyTorch.

```python
model = smp.Unet(
    encoder_name='vgg16',
    encoder_weights='imagenet',
    in_channels=3,
    classes=1,
    activation='sigmoid'
)

loss_fn = BCEJaccardLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

Vgg16 to model splotowy sieci neuronowej, nazwa vgg16 odnosi się do 16 warstw, które ten model zawiera do uczenia.

Encoder wag Imagenet współpracuje z vgg16.

In_channels określa nam ile jest klas wejściowych, u nas są 3, ponieważ działamy na RGB.

Classes są to natomiast klasy wyjściowe, u nas jest 1, ponieważ wyjście musi być w skali szarości / binarne.

Parametr activation o wartości sigmoid, jest to funkcja używana do dodania nieliniowości w modelu uczenia maszynowego.

Kolejnym krokiem algorytmu jest samo uczenie i walidacja modelu.

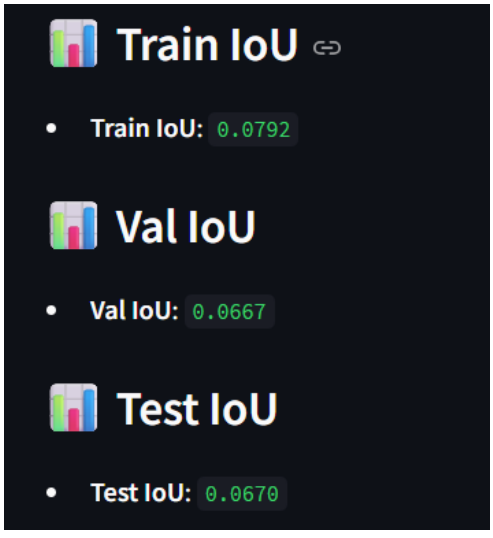
```python
best_val_loss = float("inf")
best_model = model
iter_worst = 0

for epoch in range(20):
    model.train()
    optimizer.zero_grad()
    preds = model(X_train)
    preds = torch.sigmoid(preds)
    loss = loss_fn(preds, Y_train)
    loss.backward()
    optimizer.step()

    model.eval()
    with torch.no_grad():
        val_preds = model(X_val)
        val_loss = loss_fn(torch.sigmoid(val_preds), Y_val).item()

    st.write(f"Epoch {epoch+1}, Train loss: {loss.item():.4f}, Val loss: {val_loss:.4f}")

    if val_loss < best_val_loss:
        best_val_loss = val_loss
        best_model = model
        iter_worst = 0
    else:
        iter_worst += 1
        if iter_worst == 3:
            torch.save(best_model, MODEL_PATH)
            break
```

Model uczony jest maksymalnie przez 20 epok (można dać więcej, ale poprawa jest niewielka), jednak jeśli w trakcie trwania treningu model zacznie się przeuczać, zbiór walidujący to zauważy, wtedy zakończy trening i zapisze model, który powstał przez zaczęciem przeuczania. Krótko mówiąc, jeśli w każdej kolejnej iteracji model będzie poprawiał swoją działanie na zbiorze walidującym, uczenie będzie kontynuowane, jeśli natomiast przez 3 iteracje wartość loss na zbiorze walidującym będzie większa od wcześniejszej najmniejszej to uczenie zostanie przerwane i model zostanie zapisany do pliku.

Weryfikacja modelu obiera się na wartości IoU, jest to współczynnik podobieństwa do mask eksperckich z danego zbioru.

```python
def iou_score(y_pred, y_true, threshold=0.5):  3 usages
    y_pred = (y_pred > threshold).float()
    y_true = y_true.float()

    intersection = (y_pred * y_true).sum((1, 2, 3))
    union = ((y_pred + y_true) >= 1).float().sum((1, 2, 3))
    iou = (intersection + 1e-7) / (union + 1e-7)
    return iou.mean().item()
```

```python
# Ewaluacja
best_model.eval()
with torch.no_grad():
    test_preds = torch.sigmoid(best_model(X_test))
    test_bin_preds = (test_preds > 0.5).float()
    test_iou = iou_score(test_bin_preds, Y_test)

best_model.eval()
with torch.no_grad():
    train_preds = torch.sigmoid(best_model(X_train))
    train_bin_preds = (train_preds > 0.5).float()
    train_iou = iou_score(train_bin_preds, Y_train)

best_model.eval()
with torch.no_grad():
    val_preds = torch.sigmoid(best_model(X_val))
    val_bin_preds = (val_preds > 0.5).float()
    val_iou = iou_score(val_bin_preds, Y_val)
```



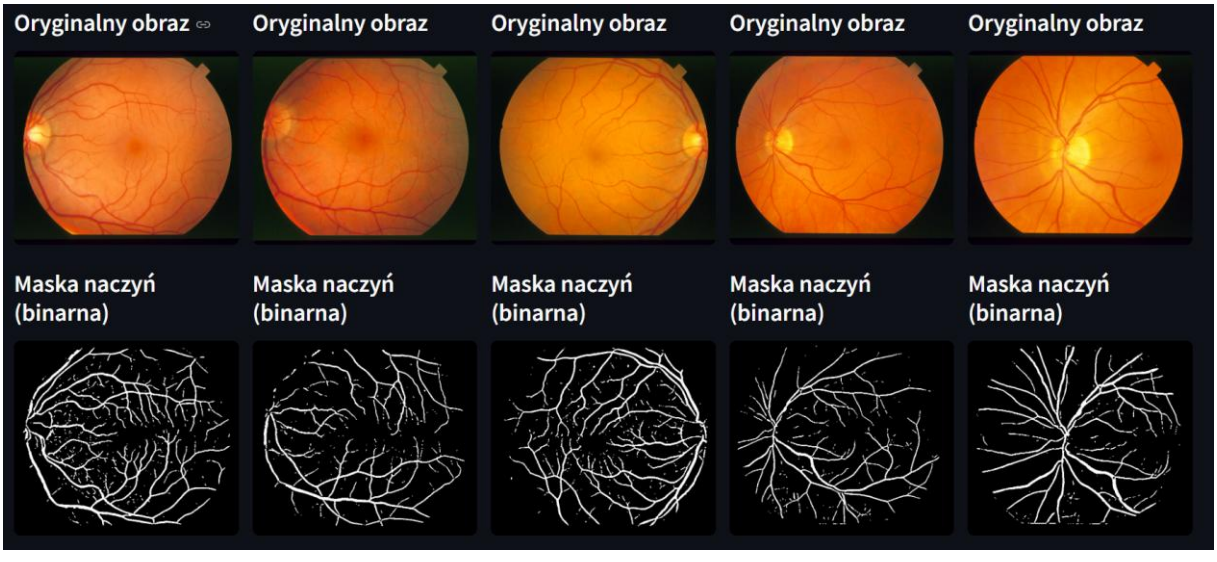
Wynik dla zbioru testowego jest delikatnie większy niż dla zbioru walidującego, ale można przyjąć że są takie same, ponieważ różnica jest marginalna. Natomiast różnica względem zbioru treningowego jest już znaczna, co oznacza, że model mógłby być lepiej wytrenowany np. zwiększając liczbę epok, jednak zdecydowaliśmy się na 20 epok, ponieważ czas trwania każdej epoki jest znaczny. Czas treningu na 20 epokach wynosił około 20 minut, więc łatwo idzie określić ile trwałby dla 50 epok. Jednak wyniki dla tego modelu są dobre, więc zdecydowaliśmy się nie zwiększać tego parametru treningowego.

Poniżej wartości loss dla zbioru treningowego i dla zbioru walidującego dla większości epok podczas trenowania modelu.

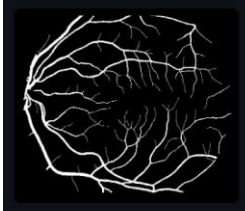
```
Epoch 6, Train loss: 1.7971, Val loss: 1.8714
Epoch 7, Train loss: 1.7922, Val loss: 1.8594
Epoch 8, Train loss: 1.7877, Val loss: 1.8486
Epoch 9, Train loss: 1.7836, Val loss: 1.8387
Epoch 10, Train loss: 1.7797, Val loss: 1.8298
Epoch 11, Train loss: 1.7761, Val loss: 1.8219
Epoch 12, Train loss: 1.7727, Val loss: 1.8148
Epoch 13, Train loss: 1.7695, Val loss: 1.8084
Epoch 14, Train loss: 1.7665, Val loss: 1.8026
Epoch 15, Train loss: 1.7637, Val loss: 1.7972
Epoch 16, Train loss: 1.7610, Val loss: 1.7921
Epoch 17, Train loss: 1.7585, Val loss: 1.7873
Epoch 18, Train loss: 1.7561, Val loss: 1.7829
Epoch 19, Train loss: 1.7537, Val loss: 1.7787
Epoch 20, Train loss: 1.7515, Val loss: 1.7749
```

4. **Wizualizacja wyników działania programów dla wybranych obrazów.** Wybrane obrazy do wizualizacji były usunięte ze zbiorów uczących dla klasyfikatora i modelu.

- Filtrowanie Frangi



Maska ekspercka

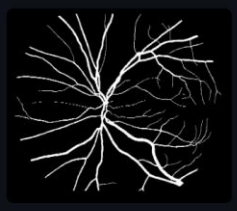


Maska ekspercka

Maska ekspercka

Maska ekspercka

Maska ekspercka



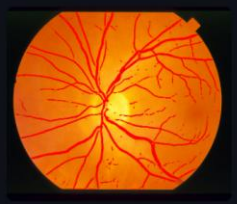
Obraz z nałożoną maską

Obraz z nałożoną maską

Obraz z nałożoną maską

Obraz z nałożoną maską

Obraz z nałożoną maską



Miary porównawcze z maską ekspercką

Accuracy: 0.9521
Sensitivity (Recall): 0.8728
Specificity: 0.9590
Średnia arytmetyczna (Se + Sp / 2): 0.9159
Średnia geometryczna (sqrt(Se × Sp)): 0.9149

Macierz pomyłek

	PN	PP
AN	373570	15989
AP	4316	29625

Miary porównawcze z maską ekspercką

Accuracy: 0.9648
Sensitivity (Recall): 0.7763
Specificity: 0.9800
Średnia arytmetyczna (Se + Sp / 2): 0.8782
Średnia geometryczna (sqrt(Se × Sp)): 0.8722

Macierz pomyłek

	PN	PP
AN	384063	7822
AP	7073	24542

Miary porównawcze z maską ekspercką

Accuracy: 0.9566
Sensitivity (Recall): 0.8555
Specificity: 0.9652
Średnia arytmetyczna (Se + Sp / 2): 0.9103
Średnia geometryczna (sqrt(Se × Sp)): 0.9087

Macierz pomyłek

	PN	PP
AN	376610	13580
AP	4814	28496

Miary porównawcze z maską ekspercką

Accuracy: 0.9654
Sensitivity (Recall): 0.7386
Specificity: 0.9828
Średnia arytmetyczna (Se + Sp / 2): 0.8607
Średnia geometryczna (sqrt(Se × Sp)): 0.8520

Macierz pomyłek

	PN	PP
AN	386562	6758
AP	7888	22292

Miary porównawcze z maską ekspercką

Accuracy: 0.9683
Sensitivity (Recall): 0.8717
Specificity: 0.9764
Średnia arytmetyczna (Se + Sp / 2): 0.9241
Średnia geometryczna (sqrt(Se × Sp)): 0.9226

Macierz pomyłek

	PN	PP
AN	381531	9218
AP	4201	28550

Legenda:

AN - Actual Negative

AP - Actual Positive

PN - Predicted Negative

PP - Predicted Positive

Podsumowanie zbiorcze wyników (średnie z 5 obrazów):

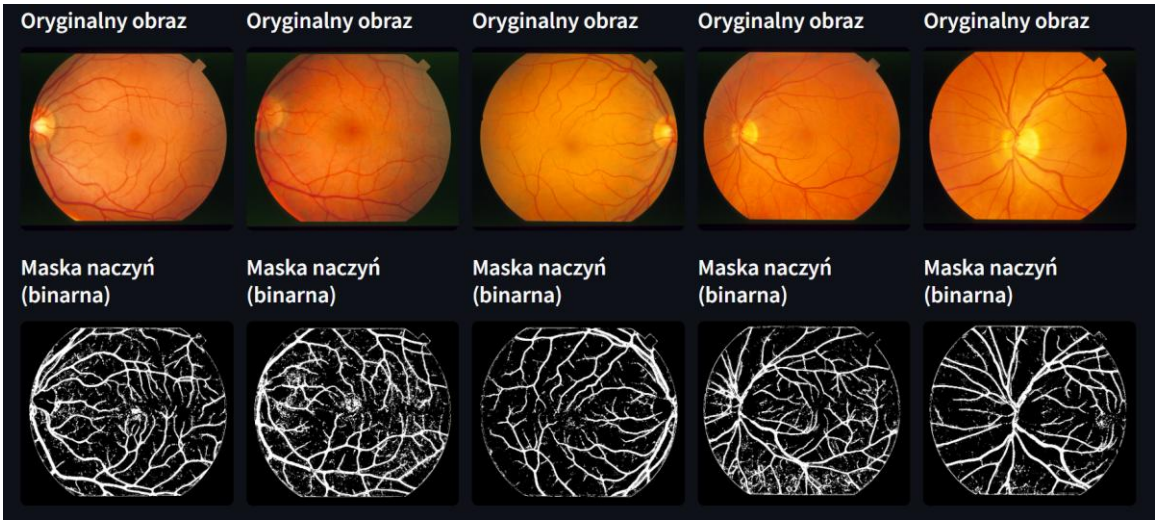
- **Średnia accuracy:** 0.9614
- **Średnia sensitivity (Recall):** 0.8230
- **Średnia specificity:** 0.9727
- **Średnia arytmetyczna (Se + Sp / 2):** 0.8978
- **Średnia geometryczna (sqrt(Se × Sp)):** 0.8941

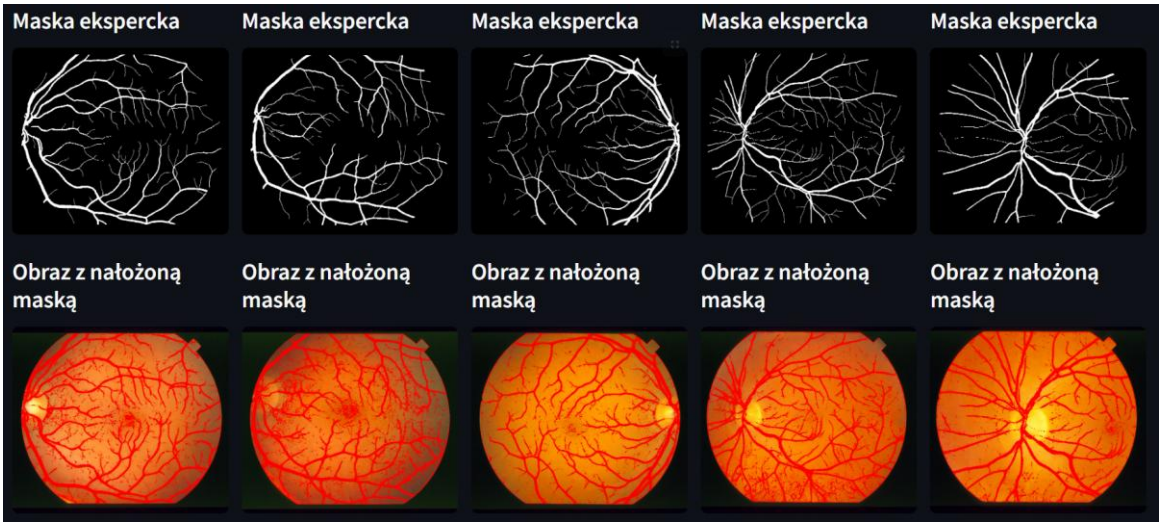
	Obraz	Accuracy	Sensitivity	Specificity	Średnia Arytmetyczna	Średnia Geometryczna
0	im0077.ppm	0.9521	0.8728	0.9590	0.9159	0.9149
1	im0081.ppm	0.9648	0.7763	0.9800	0.8782	0.8722
2	im0082.ppm	0.9566	0.8555	0.9652	0.9103	0.9087
3	im0162.ppm	0.9654	0.7386	0.9828	0.8607	0.8520
4	im0163.ppm	0.9683	0.8717	0.9764	0.9241	0.9226

Sumaryczna macierz pomyłek (łączna dla 5 obrazów):

	Predicted Negative	Predicted Positive
Actual Negative	1902336	53367
Actual Positive	28292	133505

- Uczenie maszynowe – klasyfikator





Miary porównawcze z maską ekspercką

Accuracy: 0.8962
Sensitivity (Recall): 0.9593
Specificity: 0.8907
Średnia arytmetyczna (Se + Sp / 2): 0.9250
Średnia geometryczna (sqrt(Se × Sp)): 0.9243

Macierz pomyłek

	PN	PP
AN	346964	42595
AP	1381	32560

Miary porównawcze z maską ekspercką

Accuracy: 0.8766
Sensitivity (Recall): 0.9638
Specificity: 0.8696
Średnia arytmetyczna (Se + Sp / 2): 0.9167
Średnia geometryczna (sqrt(Se × Sp)): 0.9155

Macierz pomyłek

	PN	PP
AN	340765	51120
AP	1145	30470

Miary porównawcze z maską ekspercką

Accuracy: 0.9235
Sensitivity (Recall): 0.9310
Specificity: 0.9228
Średnia arytmetyczna (Se + Sp / 2): 0.9269
Średnia geometryczna (sqrt(Se × Sp)): 0.9269

Macierz pomyłek

	PN	PP
AN	360080	30110
AP	2299	31011

Miary porównawcze z maską ekspercką

Accuracy: 0.8911
Sensitivity (Recall): 0.9234
Specificity: 0.8886
Średnia arytmetyczna (Se + Sp / 2): 0.9060
Średnia geometryczna (sqrt(Se × Sp)): 0.9058

Macierz pomyłek

	PN	PP
AN	349520	43800
AP	2313	27867

Miary porównawcze z maską ekspercką

Accuracy: 0.9141
Sensitivity (Recall): 0.9504
Specificity: 0.9111
Średnia arytmetyczna (Se + Sp / 2): 0.9307
Średnia geometryczna (sqrt(Se × Sp)): 0.9305

Macierz pomyłek

	PN	PP
AN	356011	34738
AP	1625	31126

Legenda:

AN - Actual Negative

AP - Actual Positive

PN - Predicted Negative

PP - Predicted Positive

Podsumowanie zbiorcze wyników (średnie z 5 obrazów):

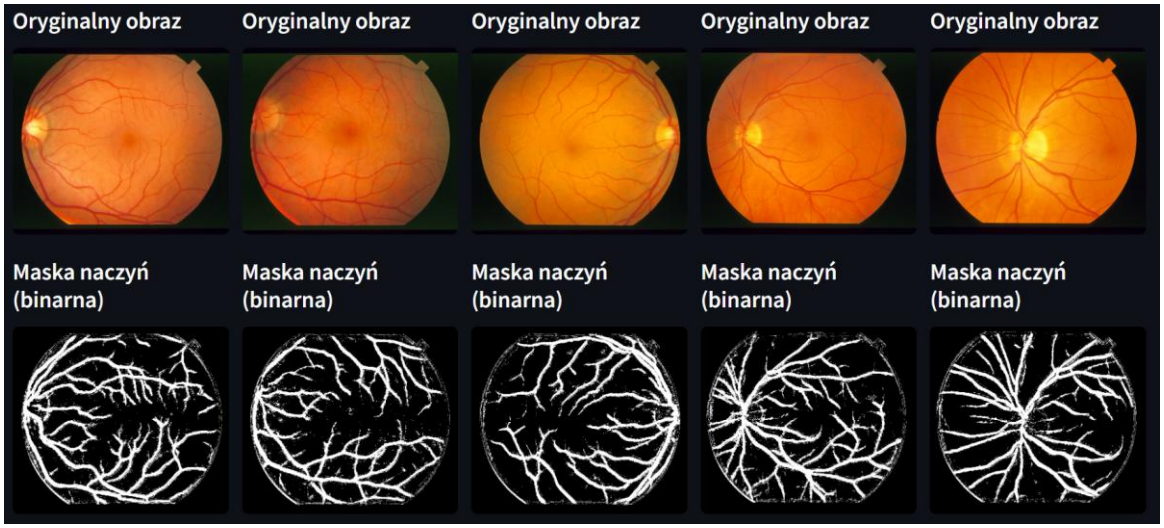
- **Średnia accuracy:** 0.9003
- **Średnia sensitivity (Recall):** 0.9456
- **Średnia specificity:** 0.8966
- **Średnia arytmetyczna (Se + Sp / 2):** 0.9211
- **Średnia geometryczna (sqrt(Se × Sp)):** 0.9206

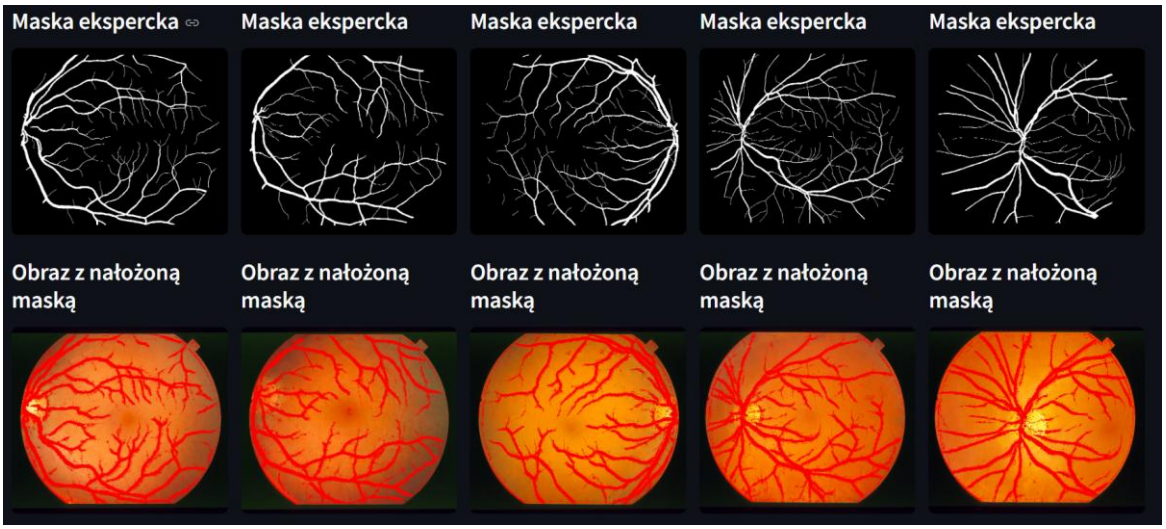
	Obraz	Accuracy	Sensitivity	Specificity	Średnia Arytmetyczna	Średnia Geometryczna
0	im0077.ppm	0.8962	0.9593	0.8907	0.9250	0.9243
1	im0081.ppm	0.8766	0.9638	0.8696	0.9167	0.9155
2	im0082.ppm	0.9235	0.9310	0.9228	0.9269	0.9269
3	im0162.ppm	0.8911	0.9234	0.8886	0.9060	0.9058
4	im0163.ppm	0.9141	0.9504	0.9111	0.9307	0.9305

Sumaryczna macierz pomyłek (łączna dla 5 obrazów):

	Predicted Negative	Predicted Positive
Actual Negative	1753340	202363
Actual Positive	8763	153034

- Uczenie maszynowe – model UNet





Miary porównawcze z maską ekspercką

Accuracy: 0.8794
Sensitivity (Recall): 0.9184
Specificity: 0.8760
Średnia arytmetyczna (Se + Sp / 2): 0.8972
Średnia geometryczna (sqrt(Se × Sp)): 0.8970

Macierz pomyłek

	PN	PP
AN	341259	48300
AP	2768	31173

Miary porównawcze z maską ekspercką

Accuracy: 0.8932
Sensitivity (Recall): 0.8869
Specificity: 0.8937
Średnia arytmetyczna (Se + Sp / 2): 0.8903
Średnia geometryczna (sqrt(Se × Sp)): 0.8903

Macierz pomyłek

	PN	PP
AN	350232	41653
AP	3577	28038

Miary porównawcze z maską ekspercką

Accuracy: 0.8915
Sensitivity (Recall): 0.8699
Specificity: 0.8933
Średnia arytmetyczna (Se + Sp / 2): 0.8816
Średnia geometryczna (sqrt(Se × Sp)): 0.8815

Macierz pomyłek

	PN	PP
AN	348554	41636
AP	4334	28976

Miary porównawcze z maską ekspercką

Accuracy: 0.8736
Sensitivity (Recall): 0.8471
Specificity: 0.8756
Średnia arytmetyczna (Se + Sp / 2): 0.8613
Średnia geometryczna (sqrt(Se × Sp)): 0.8612

Macierz pomyłek

	PN	PP
AN	344400	48920
AP	4616	25564

Miary porównawcze z maską ekspercką

Accuracy: 0.8888
Sensitivity (Recall): 0.9139
Specificity: 0.8867
Średnia arytmetyczna (Se + Sp / 2): 0.9003
Średnia geometryczna (sqrt(Se × Sp)): 0.9002

Macierz pomyłek

	PN	PP
AN	346467	44282
AP	2819	29932

Legenda:

AN - Actual Negative

AP - Actual Positive

PN - Predicted Negative

PP - Predicted Positive

Podsumowanie zbiorcze wyników (średnie z 5 obrazów):

- **Średnia accuracy:** 0.8853
- **Średnia sensitivity (Recall):** 0.8872
- **Średnia specificity:** 0.8851
- **Średnia arytmetyczna (Se + Sp / 2):** 0.8861
- **Średnia geometryczna (sqrt(Se × Sp)):** 0.8860

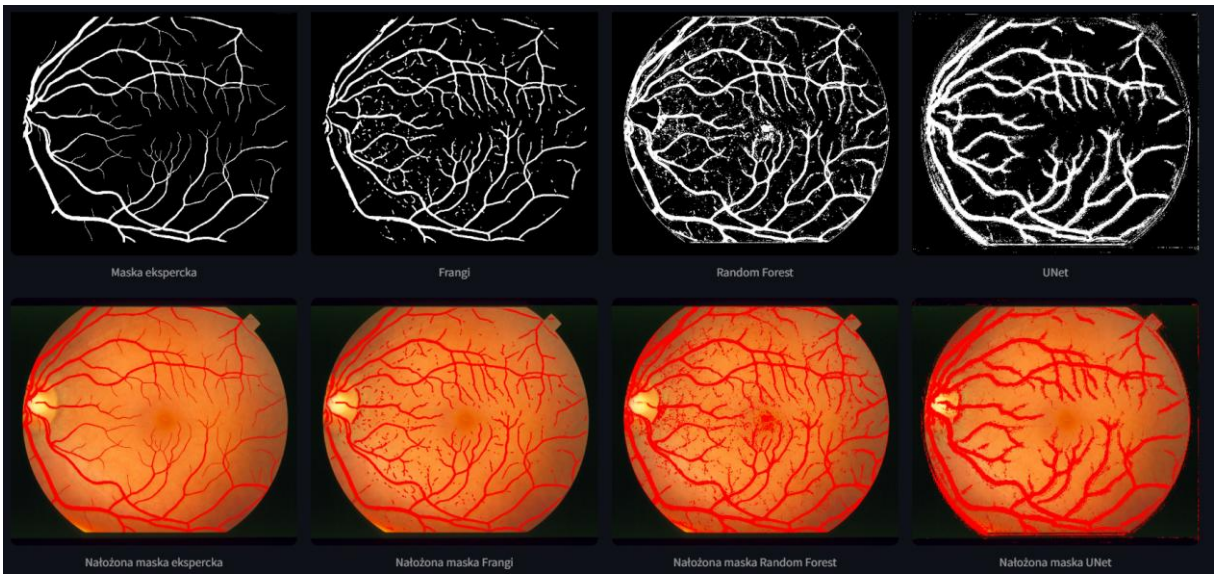
	Obraz	Accuracy	Sensitivity	Specificity	Średnia Arytmetyczna	Średnia Geometryczna
0	im0077.ppm	0.8794	0.9184	0.8760	0.8972	0.8970
1	im0081.ppm	0.8932	0.8869	0.8937	0.8903	0.8903
2	im0082.ppm	0.8915	0.8699	0.8933	0.8816	0.8815
3	im0162.ppm	0.8736	0.8471	0.8756	0.8613	0.8612
4	im0163.ppm	0.8888	0.9139	0.8867	0.9003	0.9002

Sumaryczna macierz pomyłek (łączna dla 5 obrazów):

	Predicted Negative	Predicted Positive
Actual Negative	1730912	224791
Actual Positive	18114	143683

5. Analiza wyników działania programu dla wybranych obrazów

- Obraz im0077.ppm



	Metoda	Accuracy	Sensitivity	Specificity	Mean Arithmetic	Mean Geometric
0	Frangi	0.9521	0.8728	0.9590	0.9159	0.9149
1	Random Forest	0.8962	0.9593	0.8907	0.9250	0.9243
2	UNet	0.8640	0.9185	0.8593	0.8889	0.8884

Macierze pomyłek

Frangi

	PN	PP
AN	373570	15989
AP	4316	29625

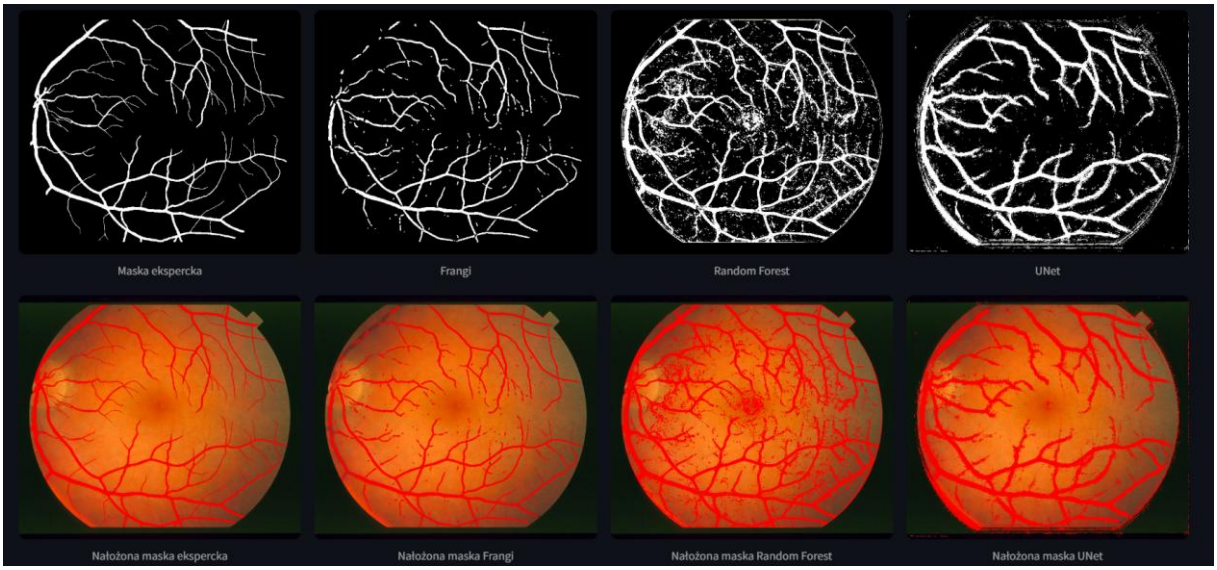
Random Forest

	PN	PP
AN	346964	42595
AP	1381	32560

UNet

	PN	PP
AN	334730	54829
AP	2767	31174

- Obraz im0081.ppm



	Metoda	Accuracy	Sensitivity	Specificity	Mean Arithmetic	Mean Geometric
0	Frangi	0.9648	0.7763	0.9800	0.8782	0.8722
1	Random Forest	0.8766	0.9638	0.8696	0.9167	0.9155
2	UNet	0.8783	0.8870	0.8776	0.8823	0.8823

Macierze pomyłek

Frangi

	PN	PP
AN	384063	7822
AP	7073	24542

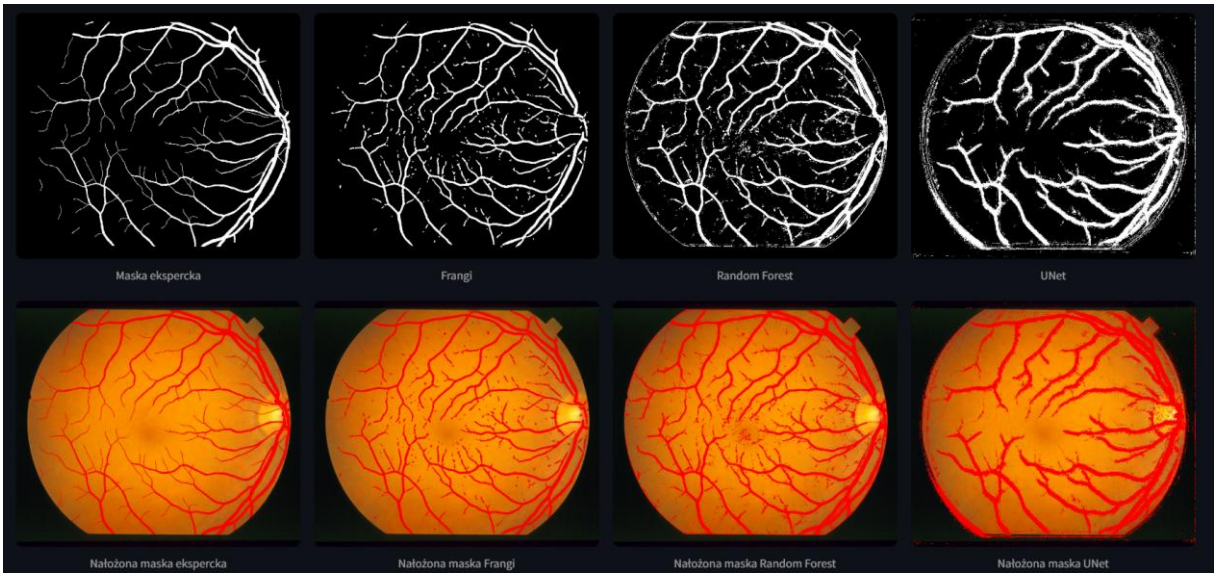
Random Forest

	PN	PP
AN	340765	51120
AP	1145	30470

UNet

	PN	PP
AN	343917	47968
AP	3573	28042

- Obraz im0082.ppm



	Metoda	Accuracy	Sensitivity	Specificity	Mean Arithmetic	Mean Geometric
0	Frangi	0.9566	0.8555	0.9652	0.9103	0.9087
1	Random Forest	0.9235	0.9310	0.9228	0.9269	0.9269
2	UNet	0.8758	0.8699	0.8763	0.8731	0.8731

🔍 Macierze pomyłek

Frangi

	PN	PP
AN	376610	13580
AP	4814	28496

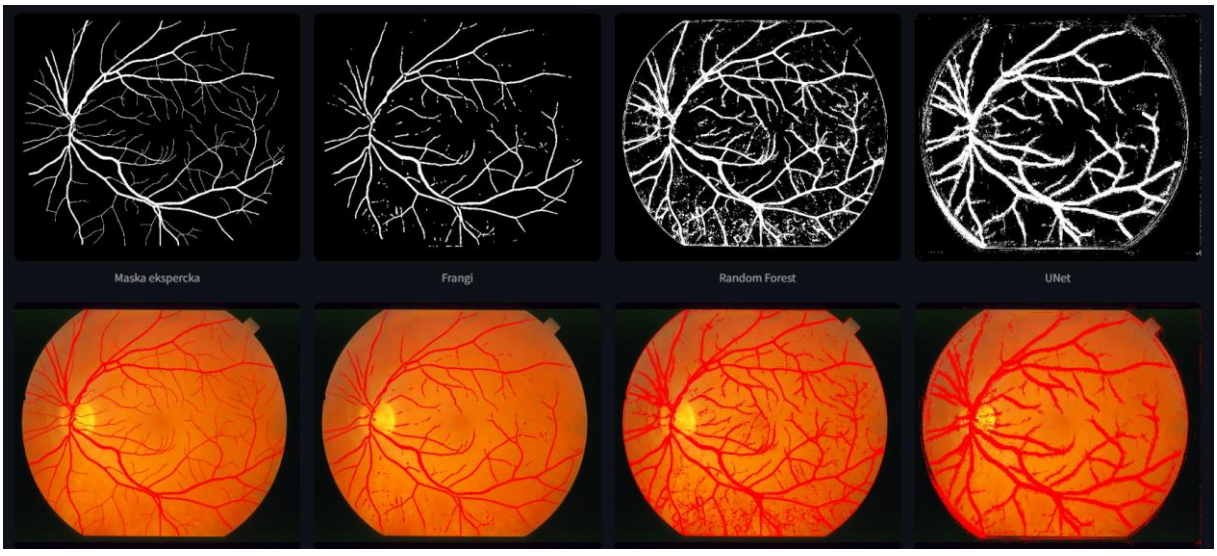
Random Forest

	PN	PP
AN	360080	30110
AP	2299	31011

UNet

	PN	PP
AN	341933	48257
AP	4334	28976

- Obraz im0162.ppm



	Metoda	Accuracy	Sensitivity	Specificity	Mean Arithmetic	Mean Geometric
0	Frangi	0.9654	0.7386	0.9828	0.8607	0.8520
1	Random Forest	0.8911	0.9234	0.8886	0.9060	0.9058
2	UNet	0.8589	0.8471	0.8598	0.8534	0.8534

🔍 Macierze pomyłek

Frangi

	PN	PP
AN	386562	6758
AP	7888	22292

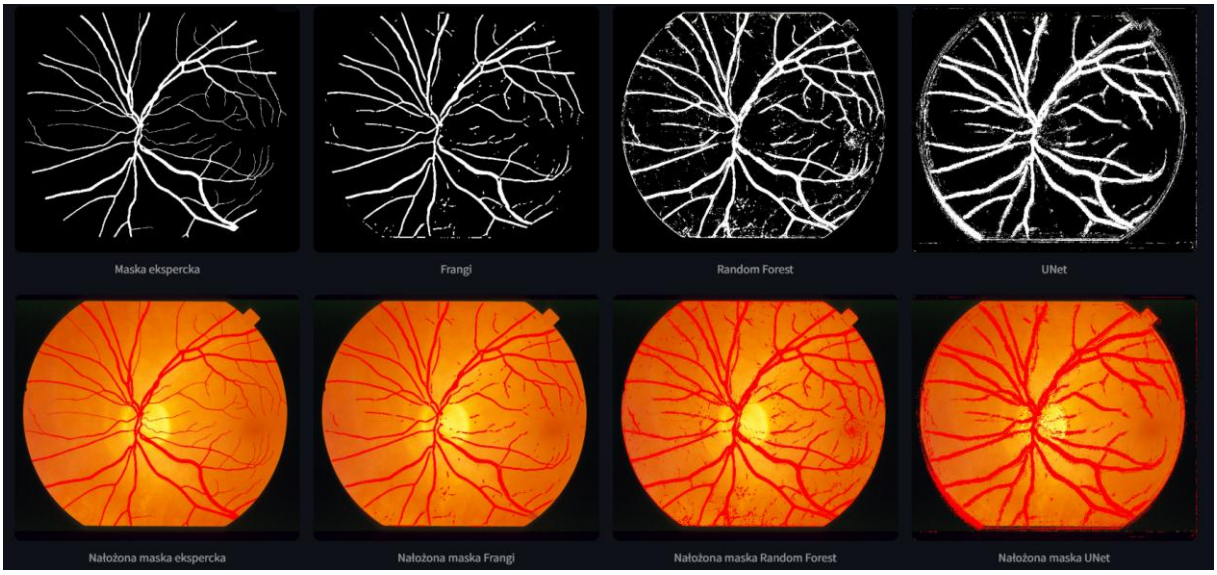
Random Forest

	PN	PP
AN	349520	43800
AP	2313	27867

UNet

	PN	PP
AN	338176	55144
AP	4616	25564

- Obraz im0163.ppm



	Metoda	Accuracy	Sensitivity	Specificity	Mean Arithmetic	Mean Geometric
0	Frangi	0.9683	0.8717	0.9764	0.9241	0.9226
1	Random Forest	0.9141	0.9504	0.9111	0.9307	0.9305
2	UNet	0.8721	0.9139	0.8686	0.8913	0.8910

🔍 Macierze pomyłek

Frangi

	PN	PP
AN	381531	9218
AP	4201	28550

Random Forest

	PN	PP
AN	356011	34738
AP	1625	31126

UNet

	PN	PP
AN	339423	51326
AP	2819	29932

- Podsumowanie

Podsumowanie zbiorcze wyników dla Frangi:

- **Średnia accuracy:** 0.9614
- **Średnia sensitivity (Recall):** 0.8230
- **Średnia specificity:** 0.9727
- **Średnia arytmetyczna (Se + Sp / 2):** 0.8978
- **Średnia geometryczna (sqrt(Se × Sp)):** 0.8941

Obraz	Accuracy	Sensitivity	Specificity	Średnia Ary	Średnia Geo
im0077.ppm	0.9521	0.8728	0.9590	0.9159	0.9149
im0081.ppm	0.9648	0.7763	0.9800	0.8782	0.8722
im0082.ppm	0.9566	0.8555	0.9652	0.9103	0.9087
im0162.ppm	0.9654	0.7386	0.9828	0.8607	0.8520
im0163.ppm	0.9683	0.8717	0.9764	0.9241	0.9226

Podsumowanie zbiorcze wyników dla Random Forest:

- **Średnia accuracy:** 0.9003
- **Średnia sensitivity (Recall):** 0.9456
- **Średnia specificity:** 0.8966
- **Średnia arytmetyczna (Se + Sp / 2):** 0.9211
- **Średnia geometryczna (sqrt(Se × Sp)):** 0.9206

Obraz	Accuracy	Sensitivity	Specificity	Średnia Ary	Średnia Geor
im0077.ppm	0.8962	0.9593	0.8907	0.9250	0.9243
im0081.ppm	0.8766	0.9638	0.8696	0.9167	0.9155
im0082.ppm	0.9235	0.9310	0.9228	0.9269	0.9269
im0162.ppm	0.8911	0.9234	0.8886	0.9060	0.9058
im0163.ppm	0.9141	0.9504	0.9111	0.9307	0.9305

Podsumowanie zbiorcze wyników dla UNet:

- **Średnia accuracy:** 0.8698
- **Średnia sensitivity (Recall):** 0.8873
- **Średnia specificity:** 0.8683
- **Średnia arytmetyczna (Se + Sp / 2):** 0.8778
- **Średnia geometryczna (sqrt(Se × Sp)):** 0.8776

Obraz	Accuracy	Sensitivity	Specificity	Średnia Aryt	Średnia Geo
im0077.ppm	0.8640	0.9185	0.8593	0.8889	0.8884
im0081.ppm	0.8783	0.8870	0.8776	0.8823	0.8823
im0082.ppm	0.8758	0.8699	0.8763	0.8731	0.8731
im0162.ppm	0.8589	0.8471	0.8598	0.8534	0.8534
im0163.ppm	0.8721	0.9139	0.8686	0.8913	0.8910

Sumaryczna macierz pomyłek dla Frangi:

	Predicted Negative	Predicted Positive
Actual Negative	1902336	53367
Actual Positive	28292	133505

Sumaryczna macierz pomyłek dla Random Forest:

	Predicted Negative	Predicted Positive
Actual Negative	1753340	202363
Actual Positive	8763	153034

Sumaryczna macierz pomyłek dla UNet:

	Predicted Negative	Predicted Positive
Actual Negative	1698179	257524
Actual Positive	18109	143688

Obiektywnie patrząc najlepsze dopasowania występuje dla filtru Frangi. Sensitivity (recall) jest mniejszy niż dla klasyfikatora Random Forest czy modelu UNet, jednak dokładność jak i ocena wizualna jest lepsza. Miary jakości dla UNet są nieznacznie gorsze niż dla Frangi czy klasyfikatora, natomiast efekty wizualne są lepsze, znacznie mniej artefaktów w środku obrazu, miary jakości są zakłócane przez artefakty występujące wokół granicy siatkówki oka oraz przez to że naczynia są znacznie grubsze niż w masce eksperckiej.