Informatyka w Medycynie

Laboratorium

Projekt drugi – wykrywanie naczyń krwionośnych dna oka

1. Skład grupy:
    a. Paweł Pytel 136786,
    b. Marcin Jaskulski 136560.
2. Użyte technologie:
    a. Język programowania: Python,
    b. Biblioteki: skimage, numpy, sklearn, torch.
3. Opis metod:
    a. Przetwarzanie obrazów:
        i. Skala szarości: łatwiej operować na jednym kanale wejściowym,
        ii. Filtr gaussowski: redukcja szumu,
        iii. Korekcja gamma: uwydatnienie krawędzi:

```
def convert2gray(img):
    return(img_as_float(rgb2gray(img)))
def preProcess(img):
    img=convert2gray(img)
    img=filters.gaussian(img,sigma=5)
    img=img**0.4
    return img
```

        iv. Filtr Sobela: wstępne wrysowanie krawędzi,
        v. Normalizacja: filtr Sobela spowodował znaczny spadek maksymalnej wartości na obrazie,
        vi. Threshhold: wartość progowa to 80. percentyl obrazu:

```
def process(img):
    img=filters.sobel(img)
    MIN = np.min(img)
    MAX = np.max(img)
    img = (img - MIN) / (MAX - MIN)
    img[img[:,:] > 1] = 1
    img[img[:,:] < 0] = 0
    img=img*(img>np.percentile(img,80))
    img=(img>0)*1.0

    return img
```

        vii. Dylatacja: wypełnienie grubszych żył między konturami,
        viii. Erozja: usunięcie niepożądanych efektów dylatacji,
        ix. Erozja maski i nałożenie jej na obraz: usunięcie kontur powstałych wokół oka:

```
def postProcess(img,mask):
    img=mp.dilation(img,selem=disk(10))
    img=mp.erosion(img,selem=disk(15))
    mask=convert2gray(mask)
    for i in range(20):
        mask=mp.erosion(mask)
    img=img*mask

    return img
```

b. Uczenie maszynowe (5.0):

i. Przygotowanie danych: z każdego dostępnego obrazu pobranie po 1000 próbek (500 pozytywnych i 500 negatywnych), razem 45 000 przypadków oraz pobranie etykiet z maski eksperckiej (manual). Pominięcie przykładów, które znajdowały się poza maską (mask), rozmiar jądra oraz ilość z każdego obrazu zostały dobrane tak aby jak najbardziej zoptymalizować zużycie RAM (podczas uczenia pamięć RAM była w ponad 90% pełna):

```
def divideImg(size=18,amount=1000):   #returns data and target in lists
    imagesPaths,masksPaths,manualPaths=takeAll()
    half=size//2
    transform=ToTensor()
    allPositive=[]
    allNegative=[]
    for image,maskP,manualP in zip(imagesPaths,masksPaths,manualPaths):
        img=img_as_float(readFile(image))

        mask=convert2gray(readFile(maskP))
        manual=convert2gray(readFile(manualP))
        positive=[]
        negative=[]
        while (len(positive)<int(amount*0.5)) or (len(negative)<int(amount*0.5)):
            x=random.randint(half,len(img)-size+half)
            y=random.randint(half,len(img[0])-size+half)
            if mask[x][y]==1:
                if manual[x][y]==1:
                    if len(positive)<int(amount*0.5):

                        positive.append(img[x-half:x+size-half,y-half:y+size-half])
                else:
                    if len(negative)<int(amount*0.5):
                        negative.append(img[x-half:x+size-half,y-half:y+size-half])
        allPositive+=positive
        allNegative+=negative
        labels=np.ones(len(allPositive)+len(allNegative))
        labels[len(allPositive):]=0
        allImgs=allPositive+allNegative
    return allImgs,labels
```

ii. Przygotowanie zbioru danych oraz klasy odpowiadającej za k-krotną walidację skrośną:

```
def prepareDataset(x,y):
    kfold=KFold(5,True,1)
    data=TensorDataset(torch.Tensor(x).permute(0, 3, 1, 2),torch.Tensor(y))    #permute for changing from NHWC to NCHW
    return kfold,data
def prepareLoaders(trainData,testData,batchSize):
    trainLoader=DataLoader(dataset=trainData,batch_size=batchSize,shuffle=True)
    testLoader=DataLoader(dataset=testData,batch_size=len(testData))
    return trainLoader,testLoader
```

iii. Użyto następującej architektury sieci neuronowej:

```
def prepareModel():
    layers=[]
    layers.append(nn.Conv2d(3,30,3,stride=1,padding=1))
    layers.append(nn.LeakyReLU())
    layers.append(nn.Conv2d(30,30,3,stride=1,padding=1))
    layers.append(nn.LeakyReLU())
    layers.append(nn.MaxPool2d(2))

    layers.append(nn.Conv2d(30,60,3,stride=1,padding=1))
    layers.append(nn.LeakyReLU())
    layers.append(nn.Conv2d(60,60,3,stride=1,padding=1))
    layers.append(nn.LeakyReLU())
    layers.append(nn.Flatten())
    layers.append(nn.Linear(60*9*9,60))
    layers.append(nn.LeakyReLU())
    layers.append(nn.Linear(60,2))
    model=nn.Sequential(*layers)
    cost=torch.nn.CrossEntropyLoss()
    opt=optim.Adam(model.parameters())
    return model,cost,opt
```

iv. Wytrenowanie modelu, oprócz 5-krotnej walidacji skrośnej zastosowano także early stopping,

v. Parametry:

1. k=5,
2. batch size=500,
3. ilość epok bez poprawy przy early stopping= 5,
4. maksymalna ilość epok=10 000:

```
def train(kfold,data):
    max_epoch = 10000
    no_improvement = 5
    batchSize=500
    models=[]
    accs=[]
    for i in range(5):   #k-fold-cross-validation
        print(str(i+1)+". iteration of cross validation")
        model,cost,opt=prepareModel()
        sets=next(kfold.split(data),None)
        trainLoader,testLoader=prepareLoaders(TensorDataset(data[sets[0]][0],data[sets[0]][1]),
                                              TensorDataset(data[sets[1]][0],data[sets[1]][1]),
                                               batchSize)
        train_loss = []
        validation_acc = []
        best_model = None
        best_acc = None
        best_epoch = None


        for n_epoch in range(max_epoch):
            model.train()
            epoch_loss = []
            for X_batch, y_batch in trainLoader:
                opt.zero_grad()
                logits = model(X_batch)
                loss = cost(logits, y_batch.long())
                loss.backward()
                opt.step()
                epoch_loss.append(loss.detach())
            train_loss.append(torch.tensor(epoch_loss).mean())
            model.eval()
            X, y = next(iter(testLoader))
            logits = model(X)
            acc = compute_acc(logits, y).detach()
            validation_acc.append(acc)
            if best_acc is None or acc > best_acc:
                print("New best epoch ", n_epoch, "acc", acc)
                best_acc = acc
                best_model = model.state_dict()
                best_epoch = n_epoch
            if best_epoch + no_improvement <= n_epoch:
                print("No improvement for", no_improvement, "epochs")
                break

        model.load_state_dict(best_model)
        models.append(best_model)
        accs.append(best_acc)
        del model
        del opt
        del cost
    return accs,models
```

vi. Uzyskane wyniki na zbiorach walidujących:

```
1. iteration of cross validation
New best epoch  0 acc tensor(0.8114)
New best epoch  1 acc tensor(0.8301)
New best epoch  2 acc tensor(0.8538)
New best epoch  4 acc tensor(0.8754)
New best epoch  6 acc tensor(0.8839)
New best epoch  8 acc tensor(0.8859)
New best epoch  10 acc tensor(0.8931)
New best epoch  11 acc tensor(0.8933)
New best epoch  12 acc tensor(0.8960)
New best epoch  13 acc tensor(0.8968)
New best epoch  14 acc tensor(0.9010)
New best epoch  15 acc tensor(0.9046)
No improvement for 5 epochs
2. iteration of cross validation
New best epoch  0 acc tensor(0.7161)
New best epoch  1 acc tensor(0.8398)
New best epoch  2 acc tensor(0.8433)
New best epoch  3 acc tensor(0.8610)
New best epoch  4 acc tensor(0.8627)
New best epoch  5 acc tensor(0.8703)
New best epoch  6 acc tensor(0.8778)
New best epoch  7 acc tensor(0.8788)
New best epoch  8 acc tensor(0.8872)
New best epoch  9 acc tensor(0.8874)
New best epoch  10 acc tensor(0.8912)
New best epoch  11 acc tensor(0.8946)
New best epoch  12 acc tensor(0.8961)
New best epoch  13 acc tensor(0.8971)
New best epoch  17 acc tensor(0.8993)
New best epoch  18 acc tensor(0.9029)
New best epoch  21 acc tensor(0.9038)
New best epoch  25 acc tensor(0.9066)
No improvement for 5 epochs
3. iteration of cross validation
New best epoch  0 acc tensor(0.7684)
New best epoch  1 acc tensor(0.8352)
New best epoch  3 acc tensor(0.8592)
New best epoch  5 acc tensor(0.8680)
New best epoch  7 acc tensor(0.8798)
New best epoch  9 acc tensor(0.8824)
New best epoch  11 acc tensor(0.8936)
New best epoch  12 acc tensor(0.8949)
New best epoch  14 acc tensor(0.8962)
New best epoch  15 acc tensor(0.8989)
New best epoch  16 acc tensor(0.9010)
New best epoch  20 acc tensor(0.9033)
New best epoch  21 acc tensor(0.9047)
New best epoch  25 acc tensor(0.9057)
No improvement for 5 epochs
```

```
4. iteration of cross validation
New best epoch  0 acc tensor(0.7237)
New best epoch  2 acc tensor(0.8422)
New best epoch  4 acc tensor(0.8493)
New best epoch  5 acc tensor(0.8588)
New best epoch  8 acc tensor(0.8593)
New best epoch  9 acc tensor(0.8668)
New best epoch  10 acc tensor(0.8718)
New best epoch  13 acc tensor(0.8789)
New best epoch  14 acc tensor(0.8840)
New best epoch  15 acc tensor(0.8898)
New best epoch  16 acc tensor(0.8963)
New best epoch  19 acc tensor(0.8984)
New best epoch  21 acc tensor(0.8991)
New best epoch  22 acc tensor(0.8992)
New best epoch  23 acc tensor(0.9009)
New best epoch  27 acc tensor(0.9021)
New best epoch  29 acc tensor(0.9036)
New best epoch  32 acc tensor(0.9052)
New best epoch  37 acc tensor(0.9082)
No improvement for 5 epochs
5. iteration of cross validation
New best epoch  0 acc tensor(0.7457)
New best epoch  1 acc tensor(0.8371)
New best epoch  2 acc tensor(0.8517)
New best epoch  3 acc tensor(0.8570)
New best epoch  4 acc tensor(0.8653)
New best epoch  7 acc tensor(0.8710)
New best epoch  8 acc tensor(0.8737)
New best epoch  9 acc tensor(0.8784)
New best epoch  10 acc tensor(0.8916)
New best epoch  12 acc tensor(0.8942)
New best epoch  13 acc tensor(0.8948)
New best epoch  15 acc tensor(0.8964)
New best epoch  17 acc tensor(0.9000)
New best epoch  18 acc tensor(0.9030)
No improvement for 5 epochs
```

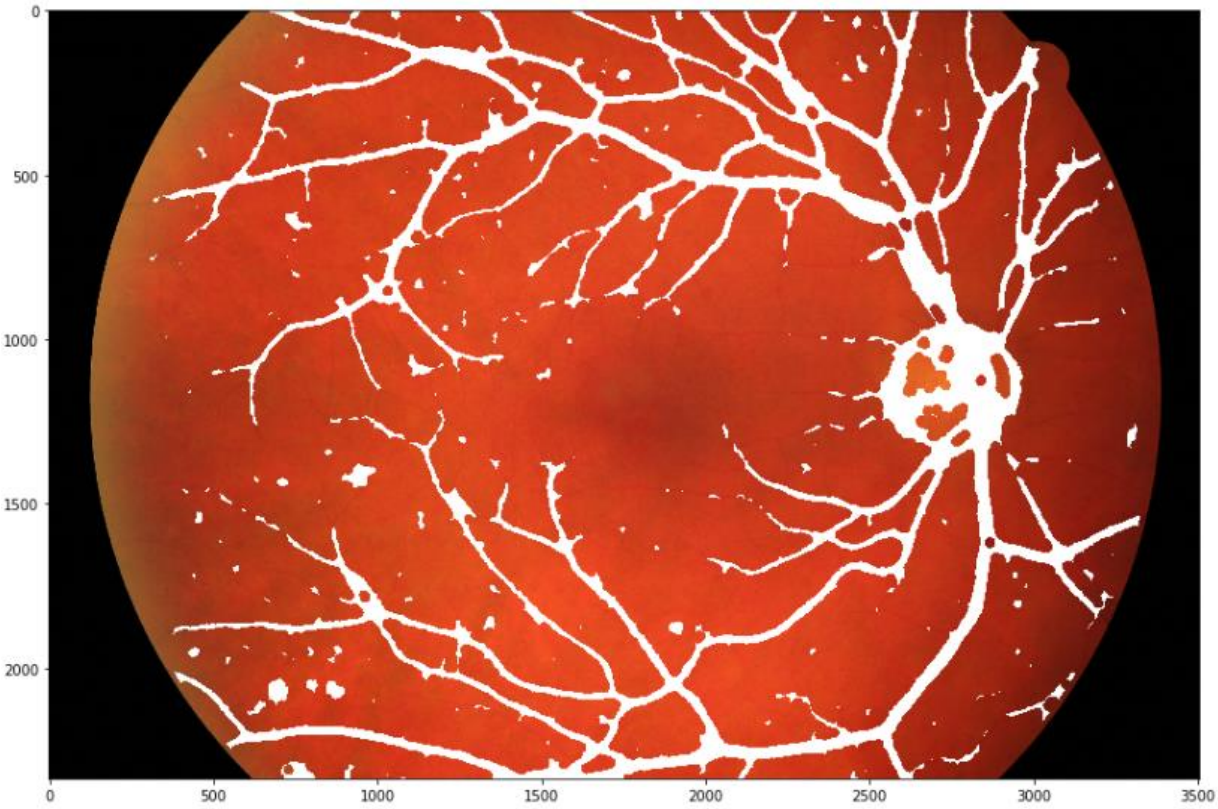
vii. Wybranie najlepszego modelu z pięciu iteracji:

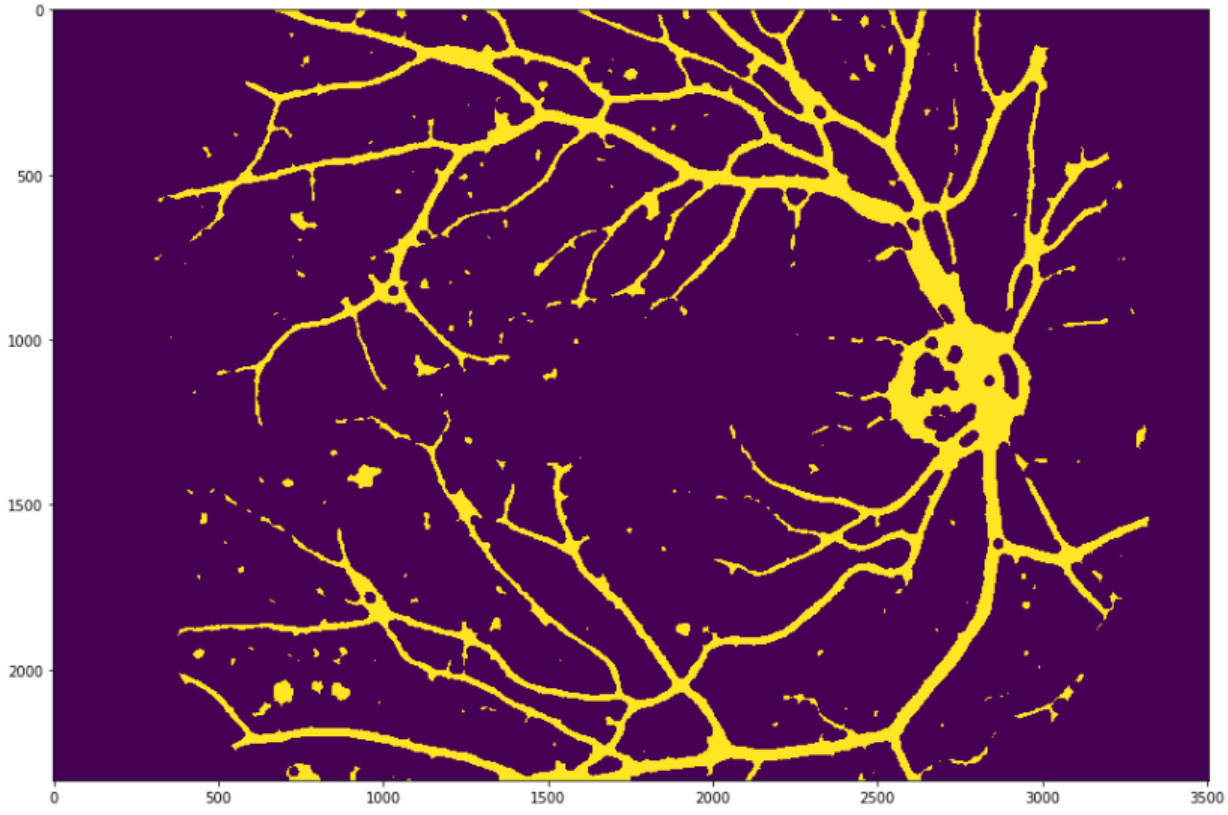
```
def chooseBestModel(models,accs):
    model,cost,opt=prepareModel()
    model.load_state_dict(models[accs.index(max(accs))])
    return model
```

viii. Wybraliśmy rozwiązanie sieci neuronowej oraz bibliotekę torach, ponieważ mieliśmy już styczność z tą technologią na przedmiocie Sztuczna Inteligencja.

4. Wizualizacja wyników:
    a. Przetwarzanie obrazu:
        i. Obraz pierwszy:

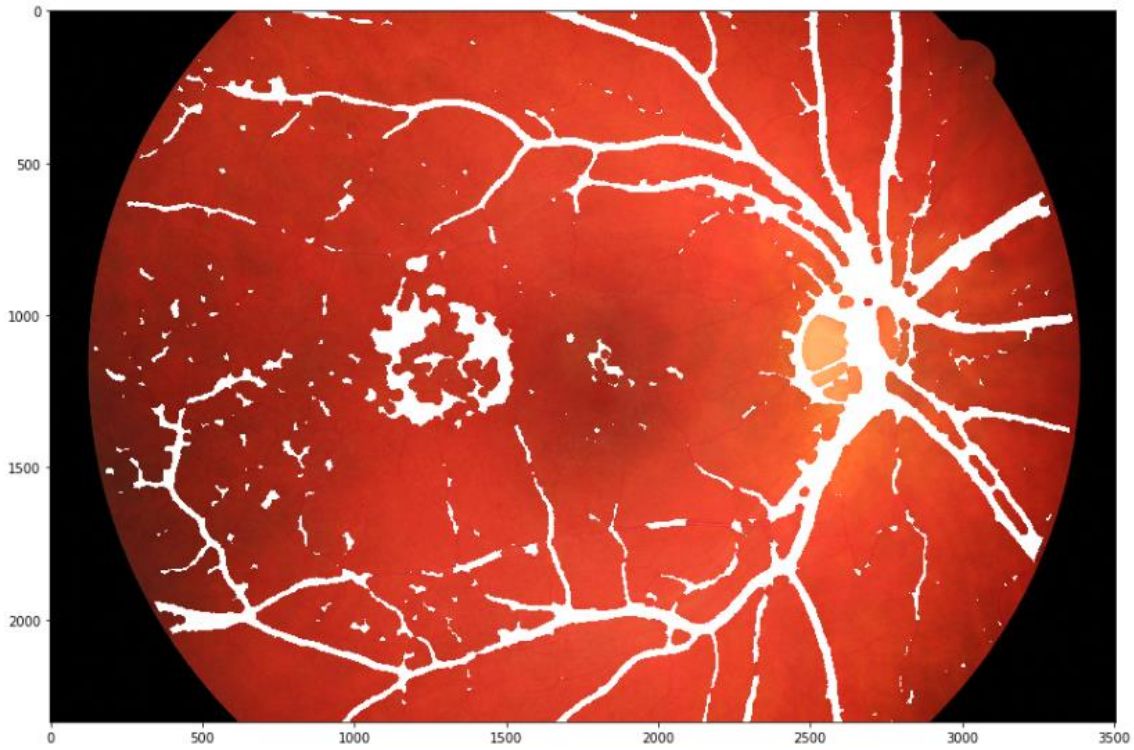
```
Accuracy: 0.9121732621607677
Sensitivity: 0.7614893037564394
Precision: 0.4511786310062227
Specificity: 0.9244614871123209
Mean of sensitivity and specificity: 0.8429753954343802
```

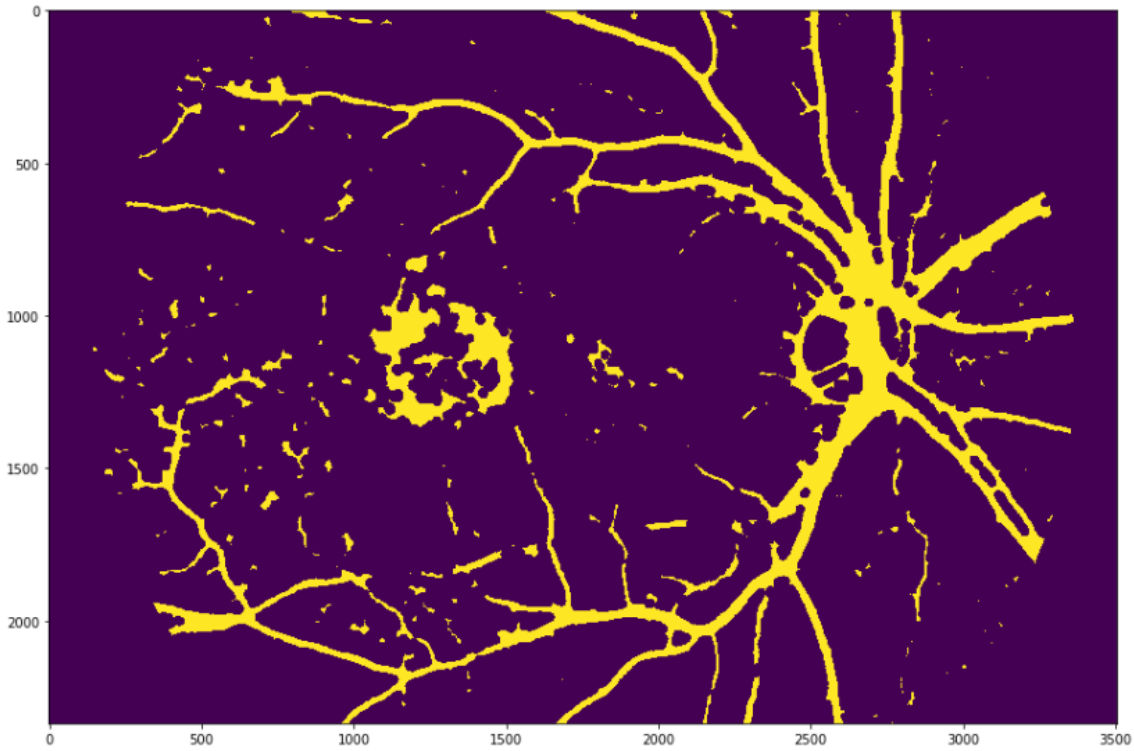




ii. Obraz drugi:

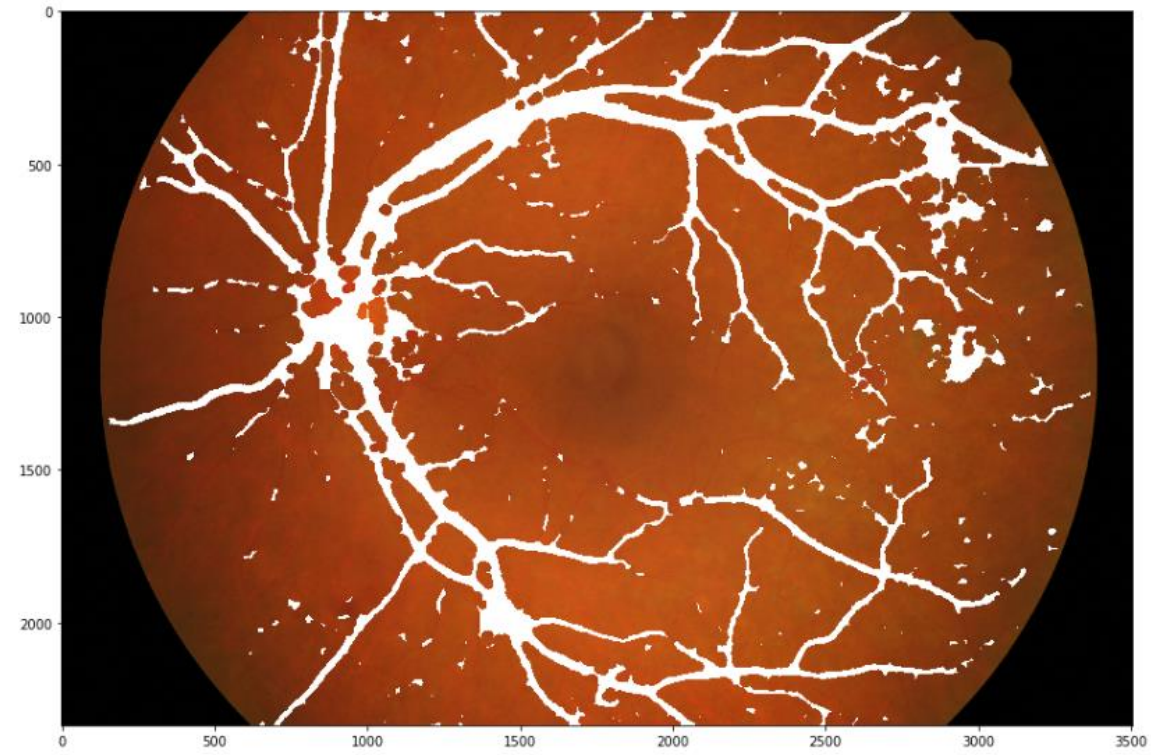
```
Accuracy: 0.8986662474359117
Sensitivity: 0.5803519982390377
Precision: 0.448244049218876
Specificity: 0.9299012913237661
Mean of sensitivity and specificity: 0.7551266447814019
```

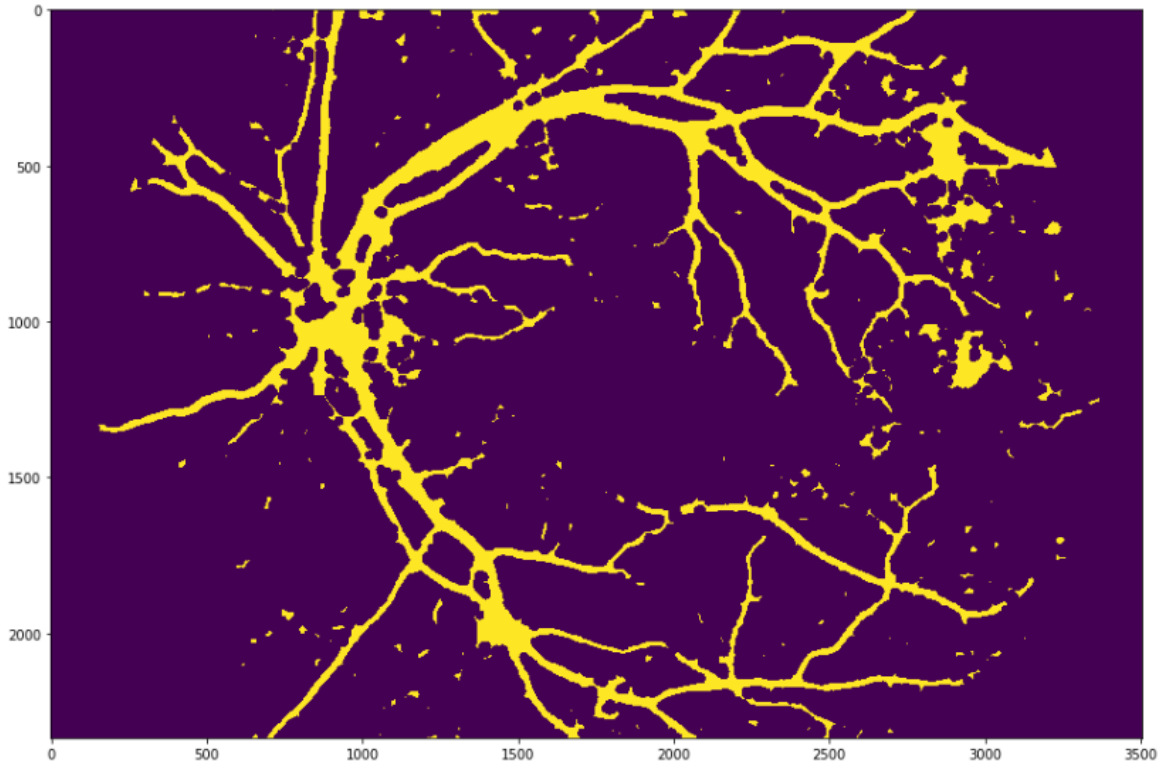




iii. Obraz trzeci:

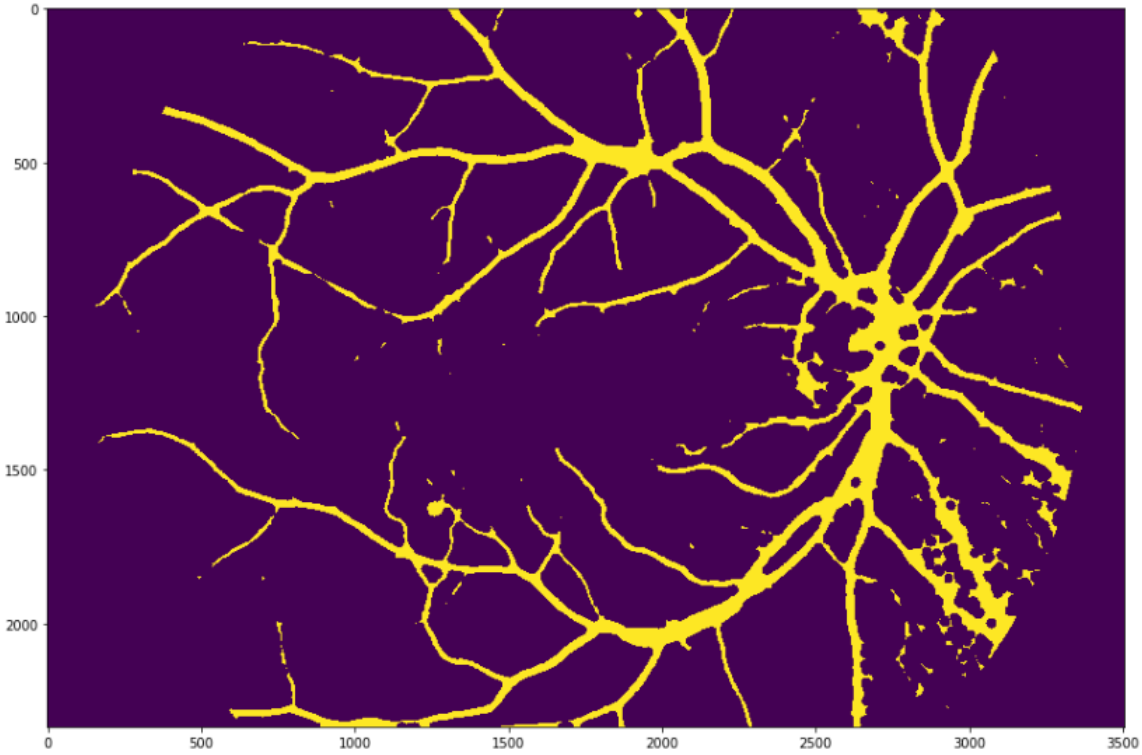
```
Accuracy: 0.9061252011100756
Sensitivity: 0.681295505539258
Precision: 0.44570532716420086
Specificity: 0.9258114206284553
Mean of sensitivity and specificity: 0.8035534630838567
```





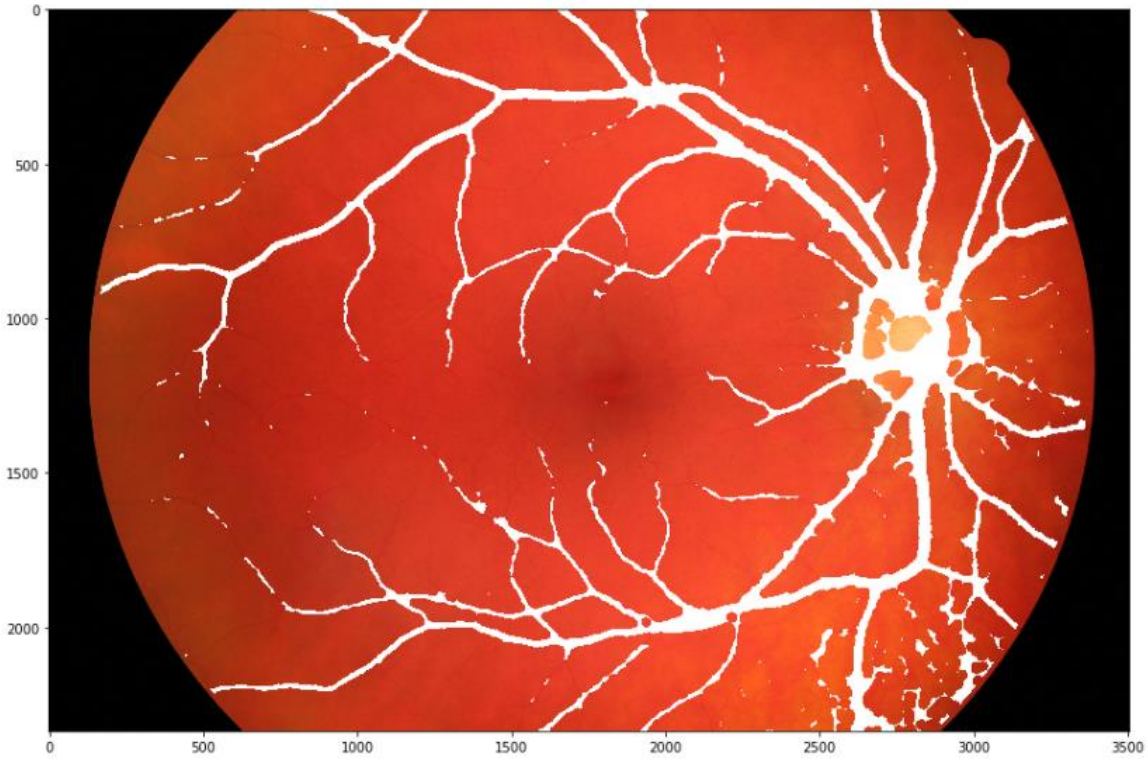
iv. Obraz czwarty:

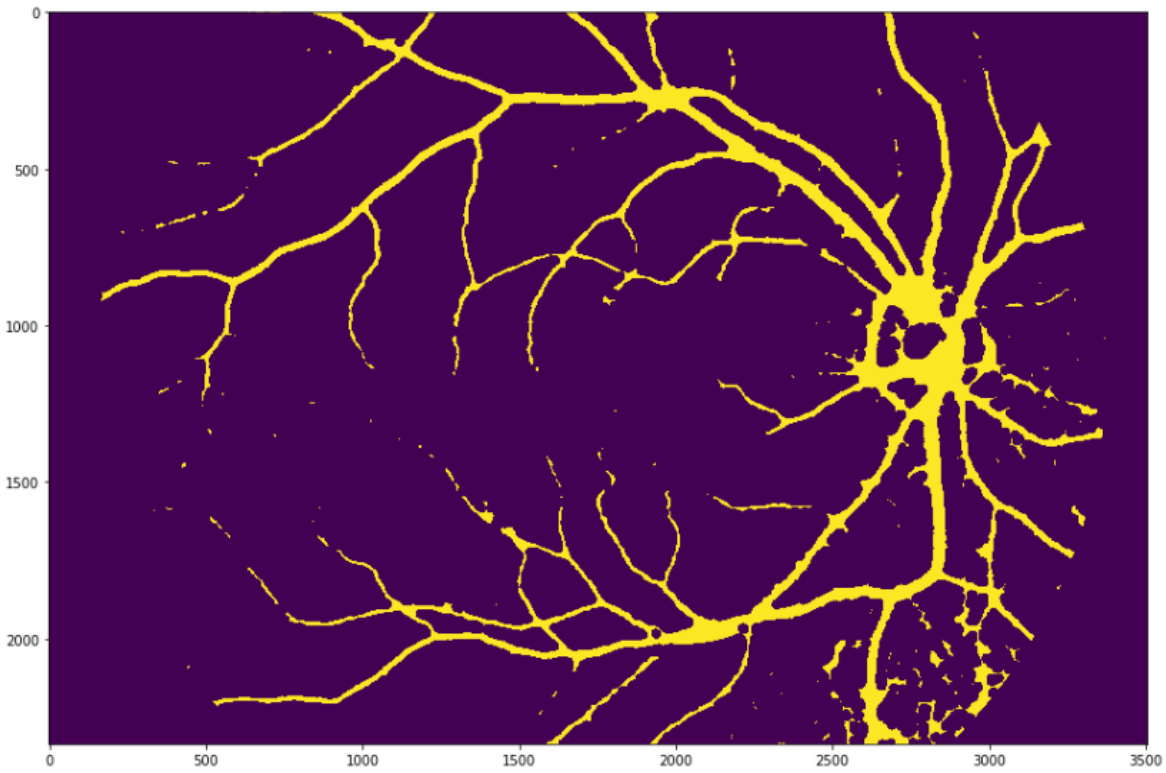
```
Accuracy: 0.9203855040234538
Sensitivity: 0.6944505279311971
Precision: 0.5473445880649782
Specificity: 0.9428631772338235
Mean of sensitivity and specificity: 0.8186568525825103
```



v. Obraz piąty:

```
Accuracy: 0.9345163160318521
Sensitivity: 0.6691412608332233
Precision: 0.6375019459446514
Specificity: 0.9614225034679588
Mean of sensitivity and specificity: 0.815281882150591
```

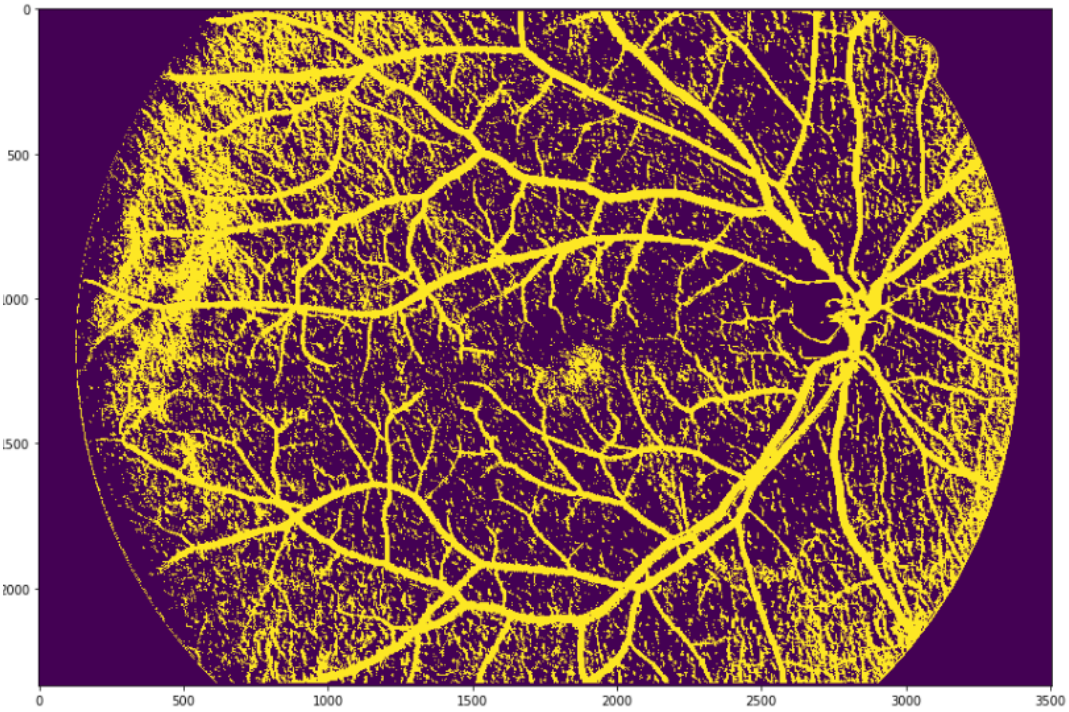




b. Uczenie maszynowe:

i. Obraz pierwszy:

```
Accuracy: 0.6988227547706161
Sensitivity: 0.9507631717760662
Precision: 0.19766161704266638
Specificity: 0.677788283669096
Mean of sensitivity and specificity: 0.8142757277225812
```



5. Analiza wyników:

a. Przetwarzanie obrazu:

i. Wnioski dla wszystkich obrazów:

Bardzo wysoka dokładność. Czułość zaniżona przez bardzo wąskie naczynia, które nie były wykrywalne przez filtr Sobela. Precyzja bardzo niska, ponieważ przypadków pozytywnych jest stosunkowo mało (mała wartość licznika), a w większości obrazów znajdowały się plamy, które niesłusznie były okonturowane (wpływa to na FP, czyli

wartość mianownika). Swoistość bardzo wysoka (tak jak w przypadku precyzji FP było bardzo dużą częścią mianownika, tak tutaj FP to jedynie jego mały odsetek, ponieważ przypadków TN jest dużo więcej niż TP).

ii. Obraz pierwszy:
Widać niedociągnięcia dla cienkich żył oraz niepoprawnie zaznaczone plamki (zwłaszcza w lewym dolnym rogu oraz na nerwie wzrokowym).

iii. Obraz drugi:
Czułość niższa niż w obrazie pierwszym, więcej naczyń nie zostało zaznaczonych (zwłaszcza w ciemniejszej części po prawej stronie).

iv. Obraz trzeci:
Klasyczny przypadek: brak cienkich żył oraz dużo niepotrzebnych plam.

v. Obraz czwarty:
Widać tutaj dużo mniej plam na oryginalnym zdjęciu, dlatego swoistość i precyzja się zwiększyły.

vi. Obraz piąty:
Według danych najwyższa dokładność, stosunkowo łatwy przypadek dla algorytmu, wyraźne żyły, brak plamek oprócz tarczy nerwu wzrokowego.

b. Uczenie maszynowe

i. Obraz pierwszy:
Niestety wyniki na innym obrazie nie są tak dobre jak na zbiorach walidujących. Została osiągnięta znacznie mniejsza dokładność. Mimo wszystko są też plusy tego algorytmu. Należy zauważyć, że bardzo dokładnie odwzorowane są przypadki pozytywne. Świadczy o tym wysoka czułość. Niestety oprócz naczyń krwionośnych algorytm zaznaczył bardzo wiele niepożądanych fragmentów (FP). Ucierpiała na tym precyzja oraz swoistość. Jest to spowodowane zbyt małą liczbą przypadków negatywnych w zbiorze testowych. Niestety sprzęt nie pozwolił na więcej. Przypadków pozytywnych jest o wiele mniej, dlatego przy równych proporcjach klas została pokryta ich większa część. Dodatkowo, losowanie próbek ze wszystkich obrazów mogło spowodować niepotrzebne pobieranie podobnych próbek oraz pominięcie wielu przypadków szczególnych.

c. Podsumowanie:

i. Metoda wykorzystująca przetwarzanie obrazów okazała się „bardziej ostrożna" co przy takiej nierównoważności klas okazało się bardzo korzystne (ostrożne zaznaczanie przypadków pozytywnych).

ii. Uczenie maszynowe prawdopodobnie osiągnęłoby lepsze wyniki, gdyby zbiór uczący mógł być większy.

iii. Największym problemem przy uczeniu maszynowym były ograniczenia sprzętowe:

1. Potrzeba restartowania serwera notebooka przez alokację zbyt dużej ilości pamięci,
2. Wiele godzin oczekiwań na wyniki (różne architektury sieci były testowane na niższych liczbach epok, co i tak nie zajmowało mało czasu, jednak ostateczne uczenie zajęło około czterech godzin, a przetestowanie modelu na obrazku w całości około trzech), dlatego zdecydowano się na tylko jeden obrazek w punkcie 4.b,

3. Maksymalne obciążenie procesora uniemożliwiało korzystanie z komputera podczas obliczeń.