

# 数字图像处理大作业 2

518030910340 唐屠天

518030910397 史睿

518030910326 艾欣

518030910400 王超维

518030910341 王晨辉

## 摘要

实验实现了对彩色眼底视网膜图像的预处理，包含尺寸、位置、颜色的归一化，用于以后的病灶识别任务，任务分为以下几部分：

- 空域对齐：检测视盘中心和黄斑中心，将所有图像的视盘中心和黄斑中心对齐；
- 颜色归一化：根据参考图像，使图像的颜色直方图与参考图像相近；
- 血管检测与移除：分割出血管区域后将血管从原图上抹去。

实验任务流程如图1所示，其中 (a) 为参考图像，(b) 为原始图像，(c) 为空域对齐后的图像，(d) 为颜色归一化后的图像，(e) 为图像中的血管分布，(f) 为移除血管后的图像。

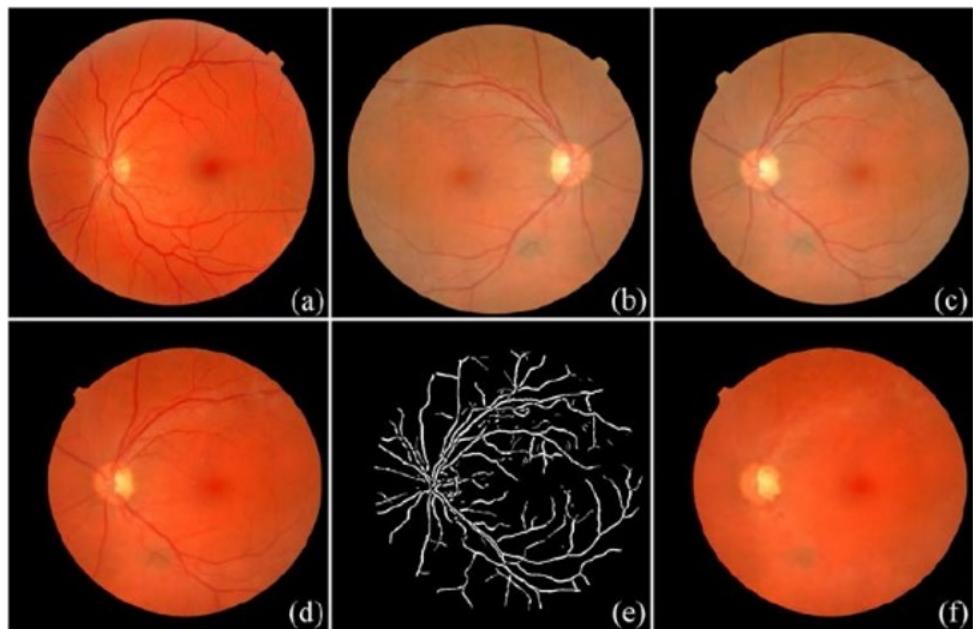


图 1 实验任务流程图

## 目录

<b>一、颜色归一化</b> . . . . .	3
<b>二、视盘中心检测</b> . . . . .	3
<b>三、黄斑中心检测</b> . . . . .	4
<b>四、空域对齐</b> . . . . .	4
<b>五、血管检测</b> . . . . .	4
<b>六、血管区域填充</b> . . . . .	7
<b>七、效果展示</b> . . . . .	8
<b>附录 A 颜色归一化</b> . . . . .	10
<b>附录 B 视盘、黄斑中心检测与空域对齐</b> . . . . .	11
<b>附录 C 血管检测</b> . . . . .	14
<b>附录 D 血管区域填充</b> . . . . .	14

## 一、颜色归一化

由于不同眼底图像是在不同光照下采集的，需要对图像进行颜色归一化处理，保证算法可适应不同成像条件下的眼底视网膜图像。从数据集中选定参考图像，计算参考图像的均值  $\mu_1$  和方差  $\sigma_1$ ，对于任意一张图片  $F_2$ ，计算其均值  $\mu_2$  和方差  $\sigma_2$  后，可通过公式1进行颜色归一化，使图像  $F_2$  和参考图像具有相同的均值和方差，即在整体上具有相似的颜色对比度。

$$F_1 = \frac{\sigma_1}{\sigma_2}(F_2 - \mu_2) + \mu_1 \quad (1)$$

颜色归一化的效果如图2所示，(a) 为选定的参考图像，(b) 为颜色归一化前的眼底视网膜图像，与参考图像在颜色和对比度上有着较大的差异，进行颜色归一化后如(c) 所示。观察发现 (a) 和 (c) 两幅图像整体亮度和颜色是相似的，颜色归一化的处理大大减小了由于不同视网膜光照变化带来的干扰。

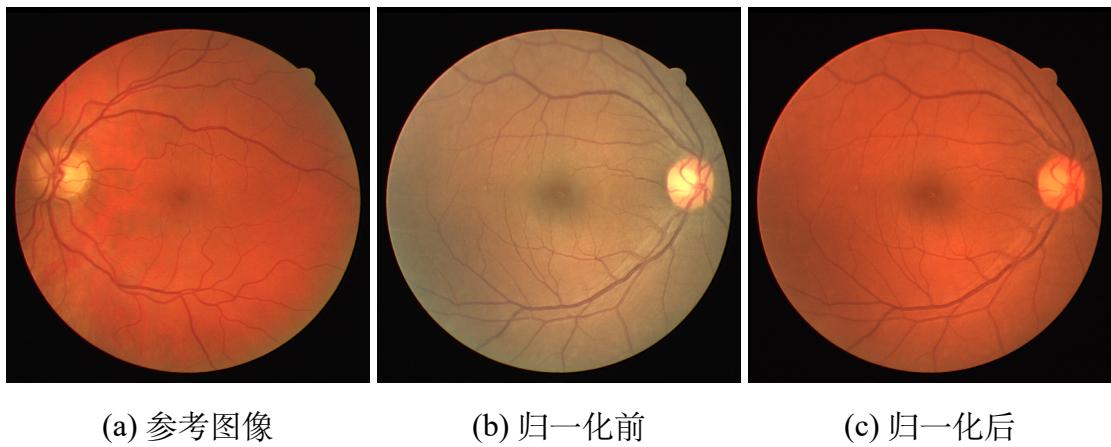


图 2 颜色归一化

## 二、视盘中心检测

视盘在眼底图像中表现为一个高亮度的圆形区域，为了在眼底图像中检测到视盘中心，进行以下操作：

1. 使用合适大小的结构元素对图像进行闭操作以去除血管和部分高亮噪声
2. 分别对彩色图片的三个颜色通道计算信息熵，选择熵最大的通道对应的灰度图进行后续操作
3. 使用合适的阈值进行二值化，得到的灰度图中仅有视盘区域和少量噪声高亮
4. 遍历整张二值图，每个高亮点对以其为中心，半径为视盘平均半径的圆形区域投一票，在得到的统计结果中，选择票数最多的点，计算它们位置的均值，作为视盘中心位置

整个处理过程中对结果准确度影响较大的因素有血管的去除效果、二值化阈值选择、投票半径选择。最终我们选择结构元素为  $65 \times 65$  的方形结构元素，二值化阈值为 254，投票半径为 250，得到了较好的效果，如图 3 所示。

### 三、黃斑中心检测

黃斑处于人眼的光学中心区，在眼底图像中颜色较暗。根据临床统计，若将视盘直径记为 DD，在图像中黃斑中心与视盘中心的水平距离介于 1.5DD 至 2.75DD，垂直距离介于-0.5DD 至 1.5DD。为了在眼底图像中检测到黃斑中心，进行以下操作：

1. 通过视盘中心在图像中的位置确定图像是左眼或右眼
2. 选择视盘中心检测时使用的熵最大的颜色通道对应的灰度图进行后续操作
3. 根据左眼或右眼的信息和视盘中心的位置，在图上截取黃斑中心的位置范围
4. 在截取出的灰度图中选择 1% 最暗的像素作为黃斑区域，记录其位置并求均值，作为黃斑中心的位置。

检测结果如图 3 所示。

### 四、空域对齐

空域对齐是将一张图像进行翻折、旋转、放缩操作，使其视盘中心、黃斑中心位置分别与给定的参考图像对齐。为此，编写函数在给定待处理图像和参考图像的视盘中心、黃斑中心位置时对图像进行运算，步骤如下：

1. 根据视盘中心和黃斑中心的相对位置确定两张图像是否是同一侧眼，如果不是则将待处理图像水平翻折，并计算视盘中心、黃斑中心在翻折后图像中的坐标位置
2. 将图像平移，使视盘中心和黃斑中心之间连线段的中点与参考图像的对齐
3. 以平移之后视盘中心和黃斑中心之间连线段的中点为中心，旋转图像并进行放缩，即可使视盘中心和黃斑中心分别对齐。其中旋转的角度和放大倍数都由已知各点位置坐标计算得到

图 3 展示了视盘中心检测、黃斑中心检测、空域对齐三者的结果。

### 五、血管检测

血管检测当然可以使用传统方法，不过我们还是使用了目前比较流行的深度学习方法进行这部分实验。我们选用的网络是 UNet<sup>1</sup>，在仅有 20 张训练图片的 DRIVE 数据集

<sup>1</sup>参考文档 <https://github.com/lee-zq/VesselSeg-Pytorch>

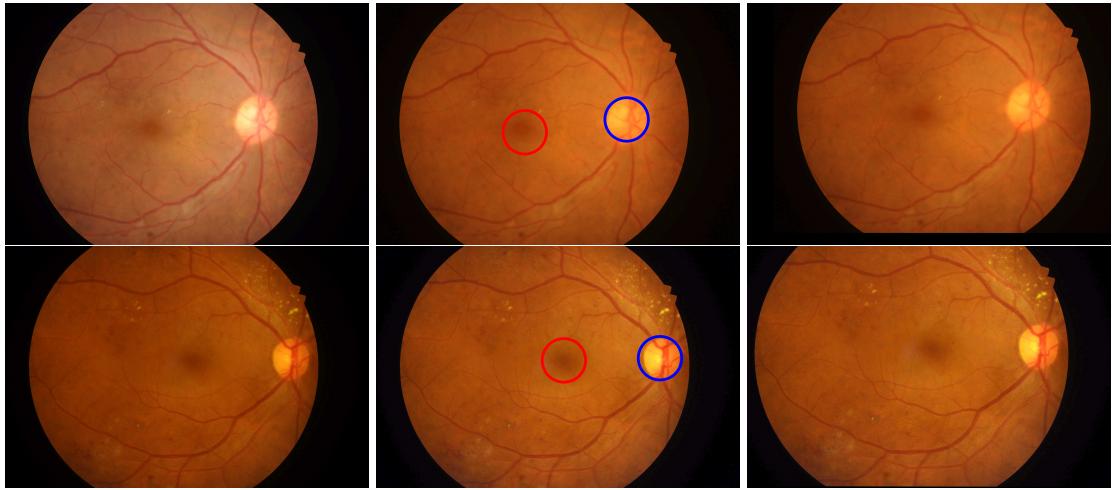


图 3 视盘中心检测、黄斑中心检测、空域对齐。从左到右：原图、检测、对齐

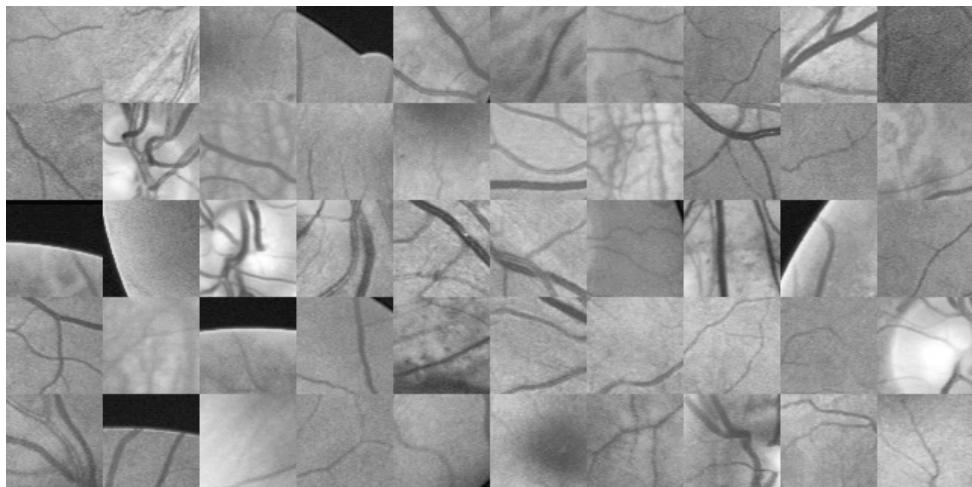


图 4 训练所用样本示意图

上，取得了比较好的效果。这也是另我们小组成员感到震惊的。医疗影像处理真是博大精深！

按照惯例，我们把整幅图像转换到灰阶后，裁剪成一个个 patch 的形式，送进网络。图 4 是一部分训练网络所用的 patches。图 5 展示了所对应的监督标注。

我们选用了 UNet 神经网络，经过几轮调参，我们发现选择 Adam 优化器，在批大小 (Batch-Size) 为 64 时，选择 0.0005 作为学习率，能够达到比较好的效果。每轮 (Epoch) 训练样本数为 200000，每个 patch 的长宽均为 64，一共进行 50 轮训练。另外，我们采用了余弦学习率方案，以保证比较好的收敛效果。

当训练完成时，在测试集上的精度达到了 0.9628，F1 指数为 0.8428，ROC 达到了 0.9876。

整个训练过程中的 Loss 变化见图 6。精度、ROC、F1 指数变化情况见图 7。

图 8 展示了从测试集中随机抽取的四个检测结果。每行图片中，从左到右依次为：



图 5 训练监督标注示意图

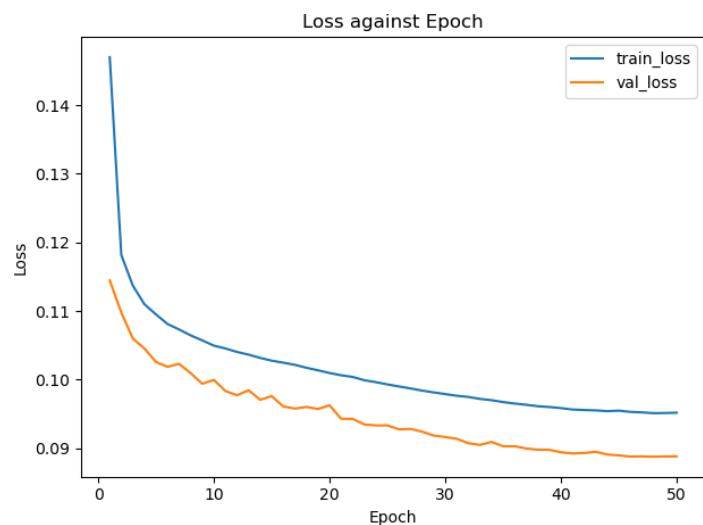


图 6 UNet 训练过程中的 Loss 曲线

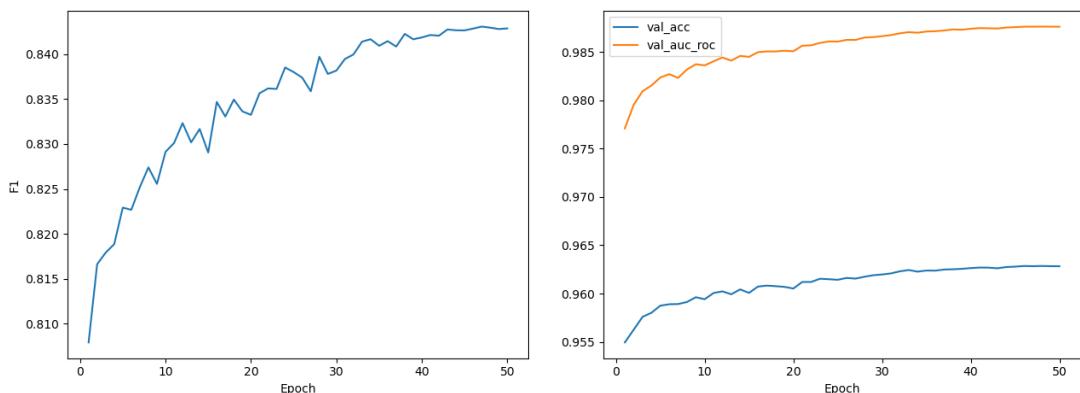


图 7 UNet 训练过程中的精度、ROC、F1 指数变化曲线

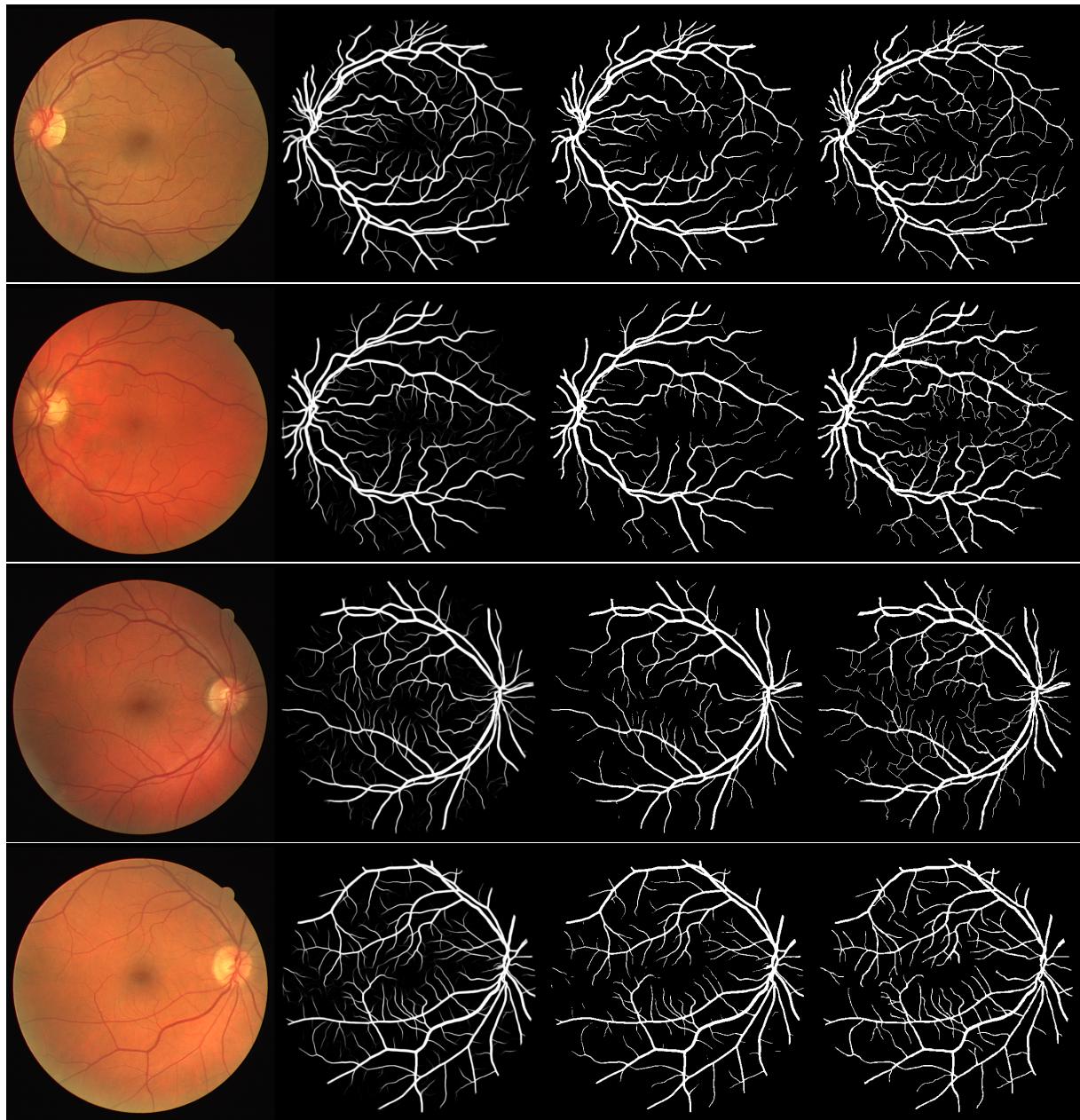


图 8 UNet 血管分割。从左到右：原图、预测概率、分割结果、专家标注。

原图、预测概率、分割结果、专家标注。

## 六、血管区域填充

该部分做的是将血管区域从原图中去除，我们实现了两种思路，最后选择了效果较好的一种想法。

我们实现的第一种思路是先在原图上将血管区域直接抹去，变成黑色。然后采用形态学中的闭操作，核心选用  $13 * 13$  的圆形。这样可以抹除掉血管区域，但是会造成模糊以及血管位置和周围的颜色差异明显，如图 9c 所示。

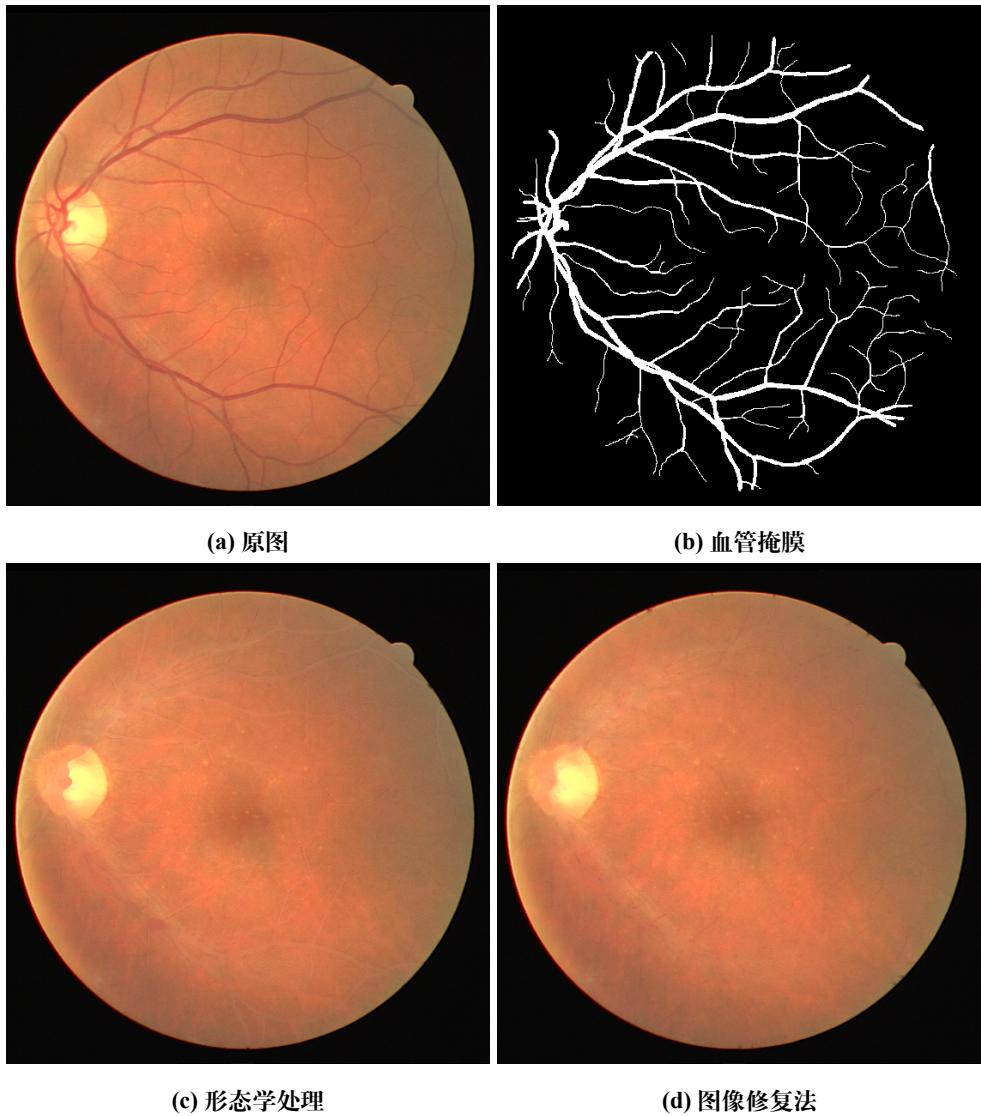


图 9 血管填充处理方法示意图

我们实现的第二种思路是采用图片修复法。这是一种由 Alexandru Telea 发明的基于快速行进方法 (Fast Marching Method, FMM) 的图像修复技术，是传统 (早期) 图像修复方法中，基于结构的图像修复中的基于插值的修复方法。它在流行的 OpenCV 库中有实现 `cv2.inpaint()`。其中参数 `size` 取 8 左右，效果较好，如图 9d 所示。

## 七、效果展示

我们在 DRIVE 数据集的测试集上测试了我们的整个流程，其中效果比较好的部分如图 10 所示。

当然，我们也发现了我们所采用方法的一些不足。比如图 11 中展示的结果中，我们的算法并没有成功地提取到视盘，因此无法做尺度归一化。

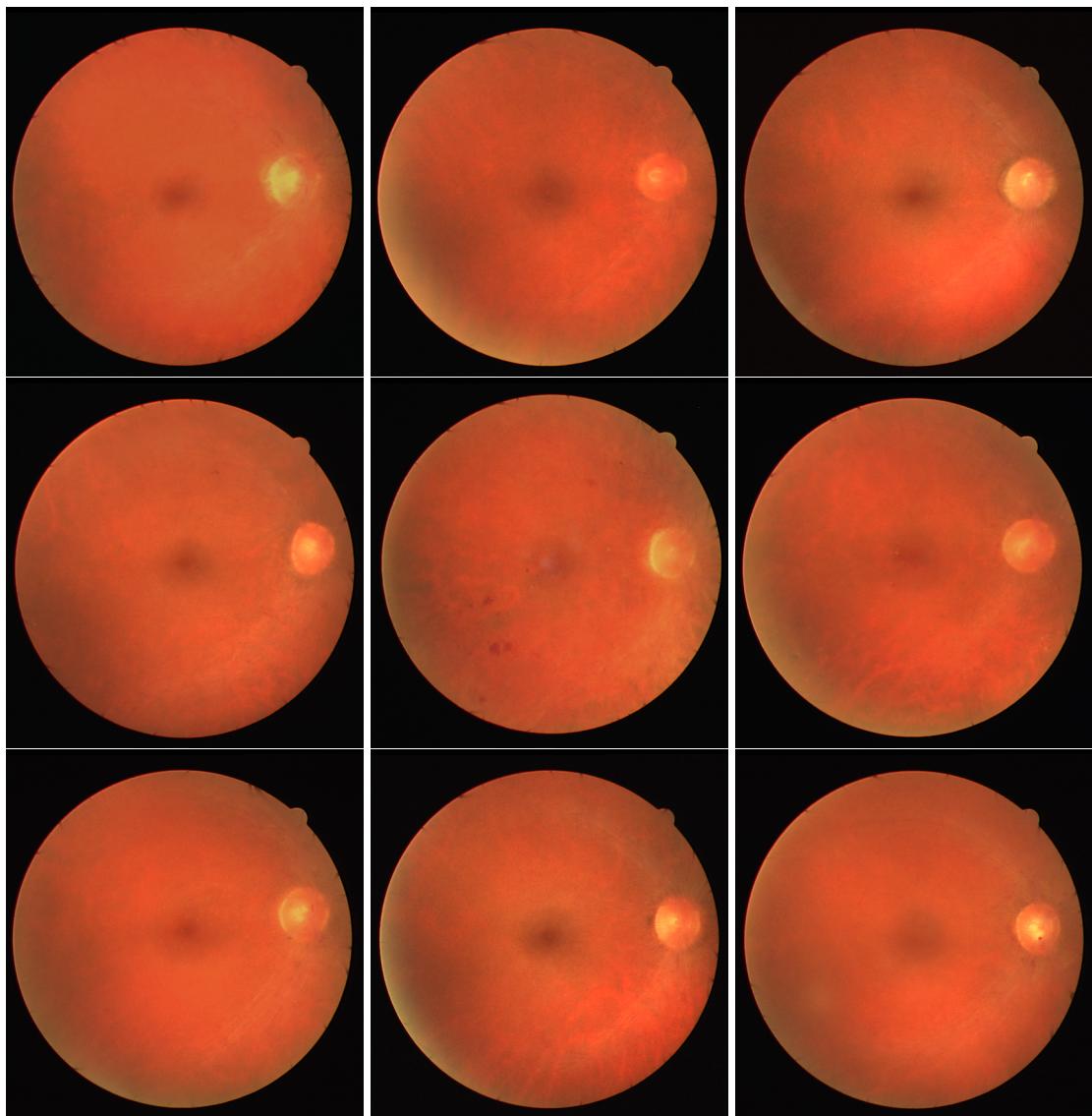


图 10 DRIVE 数据集效果选

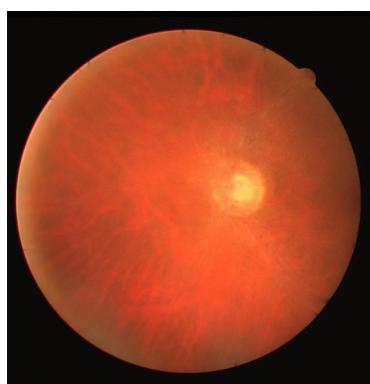


图 11 失败案例

## 附录 A 颜色归一化

```
def get_object_mask(img_I):
    mask = (img_I > 0).astype(np.uint8)
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    return mask

def get_mean_std(img, mask):
    img = img * np.stack([mask, mask, mask], -1)
    mean = np.mean(np.array(img), axis=(0, 1))
    var = np.std(np.array(img), axis=(0, 1))
    return mean, var

def get_reference(ref_file):
    img = cv2.imread(ref_file)
    img_I = np.mean(img, axis=-1)
    img_I = img_I.astype(np.float32)
    mask = get_object_mask(img_I)
    ref_mean, ref_var = get_mean_std(img, mask)

    return ref_mean, ref_var

def color_transform(img, ref_mean, ref_var):
    img = img.astype(np.float32)
    img_I = np.mean(img, axis=-1)
    mask = get_object_mask(img_I)
    mean, var = get_mean_std(img, mask)
    img_c = (img - mean) / var * ref_var + ref_mean
    img_c = img_c * np.stack([mask, mask, mask], -1)
    return img_c

def main(img_path, mu_0, sigma_0):
    img = cv2.imread(img_path)

    img_c = color_transform(img, mu_0, sigma_0)
    img_c = np.clip(img_c, 0, 255).astype(np.uint8)

    return img_c
```

## 附录 B 视盘、黄斑中心检测与空域对齐

```
import cv2
import numpy as np
import math

def pre_process():
    image = cv2.imread('IDRiD_017.jpg')
    kernel1 = np.ones((5, 5), np.uint8)
    kernel2 = np.ones((9, 9), np.uint8)
    kernel3 = np.ones((65, 65), np.uint8)
    # result = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel3)
    result = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel3)
    cv2.imwrite('IDRiD_017-res65onlyCLOSE.jpg', result)
    cv2.waitKey(0)

def RGB_count(img):
    # The number of '255's in the green channel, for example, is int(g[255])
    (B, G, R) = cv2.split(img)
    b = cv2.calcHist([B], [0], None, [256], [0, 256])
    g = cv2.calcHist([G], [0], None, [256], [0, 256])
    r = cv2.calcHist([R], [0], None, [256], [0, 256])
    return np.reshape(r, (256,)), np.reshape(g, (256,)), np.reshape(b, (256,))

def maxEntropyChannel(img):
    (B, G, R) = cv2.split(img)
    b = np.reshape(cv2.calcHist([B], [0], None, [256], [0, 256]), (256,))
    g = np.reshape(cv2.calcHist([G], [0], None, [256], [0, 256]), (256,))
    r = np.reshape(cv2.calcHist([R], [0], None, [256], [0, 256]), (256,))
    total = img.shape[0] * img.shape[1]
    b /= total
    g /= total
    r /= total

    Hb, Hg, Hr = 0.0, 0.0, 0.0
    for i in range(256):
        if b[i] > 0:
            Hb -= b[i] * math.log(b[i], 2)
        if g[i] > 0:
            Hg -= g[i] * math.log(g[i], 2)
        if r[i] > 0:
            Hr -= r[i] * math.log(r[i], 2)
    # print(Hb, Hg, Hr)
```

```

if Hr >= Hg and Hr >= Hb:
    return R
if Hg >= Hb and Hg > Hr:
    return G
return B

def vote(image, threshold, radius):
    shape = image.shape
    kernel = np.zeros((2 * radius, 2 * radius), dtype=np.float32)
    cv2.circle(kernel, (radius, radius), radius, 1)
    _, image = cv2.threshold(image, threshold, 1, cv2.THRESH_BINARY)
    return cv2.filter2D(image, -1, kernel)

def positionWithMostVotes(votes):
    shape = votes.shape
    most_votes = -1
    positions = []
    for i in range(shape[0]):
        for j in range(shape[1]):
            if votes[i][j] > most_votes:
                most_votes = votes[i][j]
                positions = [np.array((i, j))]
            if votes[i][j] == most_votes:
                positions.append(np.array((i, j)))
    return np.average(positions, axis=0)

def find_centers(img):
    """
    :param img: file name of 4288*2848 image
    :return: OD Center and Fovea Center, in two arrays, like [1218.21865533 795.40400832]
             [1663.22873953 1954.15065074]
    """
    image = cv2.imread(img)
    shape = image.shape
    maxE_image = maxEntropyChannel(image)
    ODCenter = positionWithMostVotes(vote(maxE_image, 254, 250))

    if ODCenter[1] > shape[1] / 2:
        FSR = maxE_image[max((int(ODCenter[0]) - 225), 0):min((int(ODCenter[0]) + 675),
                                                               shape[0]),
                         (int(ODCenter[1]) - 1237):(int(ODCenter[1]) - 675)]
        distribution = np.reshape(cv2.calcHist([FSR], [0], None, [256], [0, 256]), (256,))

```

```

darkest_num = FSR.shape[0] * FSR.shape[1] / 100
s = 0
threshold = 0
while s < darkest_num:
    s += distribution[threshold]
    threshold += 1
positions = []
for i in range(FSR.shape[0]):
    for j in range(FSR.shape[1]):
        if FSR[i][j] < threshold:
            positions.append(np.array((i, j)))
center = np.average(positions, axis=0)
FCenter = np.array((max((int(ODCenter[0]) - 225, 0) + center[0], int(ODCenter[1]) - 1237 + center[1])))

else:
    FSR = maxE_image[max((int(ODCenter[0]) - 225, 0):min((int(ODCenter[0]) + 675, shape[0]), (int(ODCenter[1]) + 675):(int(ODCenter[1]) + 1237)])
distribution = np.reshape(cv2.calcHist([FSR], [0], None, [256], [0, 256]), (256,))
darkest_num = FSR.shape[0] * FSR.shape[1] / 100
s = 0
threshold = 0
while s < darkest_num:
    s += distribution[threshold]
    threshold += 1
positions = []
for i in range(FSR.shape[0]):
    for j in range(FSR.shape[1]):
        if FSR[i][j] < threshold:
            positions.append(np.array((i, j)))
center = np.average(positions, axis=0)
FCenter = np.array((max((int(ODCenter[0]) - 225, 0) + center[0], int(ODCenter[1]) + 675 + center[1])))

return ODCenter, FCenter

def transform(img, origin1, origin2, target1, target2):
    """
    :param img: image parameter of 4288*2848 image
    :param origin1: OD Center of the image being processed, 2d array like [1260.60227367 826.91868854]
    :param origin2: Fovea Center of the image being processed, 2d array
    :param target1: OD Center of the reference image, 2d array
    :param target2: Fovea Center of the reference image, 2d array
    :return: image parameter of the result of transformation process
    """

```

```

result = img
new_origin1, new_origin2 = origin1, origin2
if (origin1[1] - origin2[1]) * (target1[1] - target2[1]) < 0:
    result = cv2.flip(result, 1)
new_origin1[1] = 4287 - origin1[1]
new_origin2[1] = 4287 - origin2[1]
mat_translation = np.float32([[1, 0, int((target1[1] + target2[1] - new_origin1[1] -
new_origin2[1]) / 2)],
[0, 1, int((target1[0] + target2[0] - new_origin1[0] -
new_origin2[0]) / 2)]])
result = cv2.warpAffine(result, mat_translation, (4288, 2848))
mat_rotation = cv2.getRotationMatrix2D((target1[0] + target2[0]) * 0.5, (target1[1] +
target2[1]) * 0.5,
math.atan((new_origin1[1] - new_origin2[1]) / (
new_origin1[0] - new_origin2[0])) - math.atan(
(target1[1] - target2[1]) / (target1[0] -
target2[0])), math.sqrt(
(target1[0] - target2[0]) * (target1[0] - target2[0]) + (target1[1] - target2[1]) * (
target1[1] - target2[1])) / math.sqrt(
(new_origin1[0] - new_origin2[0]) * (new_origin1[0] - new_origin2[0]) + (
new_origin1[1] - new_origin2[1]) * (new_origin1[1] - new_origin2[1])))
result = cv2.warpAffine(result, mat_rotation, (4288, 2848))
return result

if __name__ == '__main__':
01, F1 = find_centers('IDRiD_026.jpg')
print(01, F1)
02, F2 = find_centers('IDRiD_370.jpg')
print(02, F2)
img = transform(cv2.imread('IDRiD_026.jpg'), 01, F1, 02, F2)
# img = transform(cv2.imread('IDRiD_001.jpg'), [1805, 2858], [1970, 1494], [1119, 943],
[1329, 2270]) # 001->002
cv2.imwrite('result.jpg', img)

```

## 附录 C 血管检测

该部分运用到了神经网络，工程量比较大，因此随附件一起提交。

## 附录 D 血管区域填充

```

import cv2
import numpy as np

```

```
import imageio

def repairimg(imgpath,maskpath):
    gifmask = imageio.imread(maskpath)
    imgmask = np.array(gifmask[0],dtype=np.uint8)
    img = cv2.imread(imgpath)
    fina_ok = cv2.inpaint(img, imgmask, 8, cv2.INPAINT_TELEA)
    return fina_ok, imgmask, img

# 图像修复法处理
maskpath='./1st_manual/21_manual1.gif'
imgpath='./images/21_training.tif'
fina_ok, imgmask, img=repairimg(imgpath,maskpath)
cv2.imshow(' ',fina_ok)
cv2.waitKey(0)

# 形态学处理
img[np.where(imgmask==255)]=[0,0,0]
img_copy=img
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (13, 13))
fina_ok = cv2.morphologyEx(img_copy, cv2.MORPH_CLOSE, kernel)

fina_ok[np.where(imgmask==0)]=img[np.where(imgmask==0)]
cv2.imwrite('exm_bad.png', fina_ok)
```