AUTOMATED HARD EXUDATES SEGMENTATION IN RETINAL IMAGES USING PATCH BASED U-NET

1. 论文解读

论文主要研究糖尿病视网膜病变 (Diabetic Retinopathy, DR) 的自动病灶分割问题。研究的关键点如下：

研究背景

- 糖尿病视网膜病变是糖尿病患者常见的严重并发症，若不及时治疗，可能导致视力丧失。
- 传统的临床诊断依赖于眼科医生的经验，但检测过程耗时且依赖专业人员，容易受主观因素影响。
- 计算机辅助诊断 (Computer-Aided Diagnosis, CAD) 系统能提高检测效率，降低成本，减轻医疗资源压力。
- 研究的目标是基于 **U-Net 神经网络** 自动分割眼底图像中的 **硬性渗出物** (Hard Exudates, EX)。

方法

数据预处理

- 研究使用 **143 张视网膜眼底图像**（其中 54 张包含硬性渗出物）。
- 由于 **病灶区域在整张图像中的占比非常小**，直接训练 U-Net 容易导致类别不均衡问题。
- 采用 **Patch-based 方法**，将原始 4288×2848 的大图切分成 **512×512 小块**，并采用 **256 像素的步长进行滑动**，确保病灶区域能更完整地被覆盖。
- 采用 **数据增强** (Data Augmentation) 技术，例如：随机左右翻转、上下翻转、亮度变化等，以增加数据的多样性，提高模型泛化能力。

U-Net 网络架构

- 采用 U-Net 进行病灶分割，它由 **收缩路径** (Contracting Path) **和 扩展路径** (Expanding Path) 组成。
- **收缩路径**：逐步下采样特征图（类似 CNN），提取深层次语义信息。
- **扩展路径**：通过上采样 (Upsampling) 逐步恢复图像分辨率，并结合收缩路径中的高分辨率特征，以增强局部信息的恢复能力。
- 采用 **加权交叉熵损失函数** (Weighted Binary Cross Entropy Loss)，对病灶区域赋予更高的权重，以减少小病灶被忽略的情况。

实验结果

- **评价指标**：
  - **灵敏度** (Sensitivity)：88.609%
  - **特异度** (Specificity)：99.944%
  - **阳性预测值** (PPV, Positive Predictive Value)：82.689%
- 通过 FROC (Free Response Operating Characteristic) **曲线** 选择最佳分割阈值，以优化 **真阳性率** (True Positive Rate, TPR) 和 **假阳性率** (False Positive Rate, FPR)。

- 训练采用 TensorFlow **框架**，硬件环境为 NVIDIA GTX 1080 GPU。

贡献

- 该研究提出了一种 Patch-based U-Net **方法**，能在 **有限的训练数据** 下实现 **高精度的糖尿病视网膜病变病灶分割**。
- 通过 **滑动窗口** (Sliding Window) **和** Patch Overlapping 技术，使病灶区域能更完整地参与训练，提高了模型性能。
- 采用 **加权交叉熵损失** 解决病灶区域占比小的问题，提高了分割精度。

---

2. 数据预处理复现 (Data Preprocessing)

复现环境：
操作系统:Windows11
CUDA:9.0
cuDNN:7.3.0
TenserFlow:1.12.0
Python:3.6.1
GPU:RTX 3060 Laptop 4GB

本节中，数据集经历了三个主要的预处理步骤：

1. **二值化地面真值掩码** (将掩码转换为二值图像)
2. **从高分辨率视网膜图像和对应掩码中提取重叠的图像块**，同时保持病灶和非病灶图像块之间的平衡比例
3. **将图像块分割为训练集和验证集**，并将它们的文件名存储在 `train.csv` 和 `test.csv` 中

二值化地面真值掩码

为确保病灶分割模型接收明确定义的二值标签，我们将地面真值掩码二值化为两个强度级别：**0（背景）**和**255（病灶区域）**。这确保了非病灶区域完全为黑色，病灶区域完全为白色，提高了模型区分它们的能力。

关键代码：`gt_binary.py`

```
from PIL import Image
import os

def threshold_mask(infile, output_dir="training_set/"):
    outfile = os.path.splitext(infile)[0]
    extension = os.path.splitext(infile)[1]

    if extension != ".tif":
        return

    infile_path = os.path.join(input_dir, infile)

    if infile != outfile:
        try:
            im = Image.open(infile_path)
            gray = im.convert('L')  # Convert to grayscale
            bw = gray.point(lambda x: 0 if x < 50 else 255, '1')  # Apply binary
thresholding
            bw.save(output_dir + outfile + extension, "JPEG", quality=100)
        except IOError:
            print("Cannot process image:", infile)

    print("Processed image:", infile)

if __name__=="__main__":
    output_dir = "training_set/"
    dir = os.getcwd()
    input_dir = os.path.join(dir, "mask")

    if not os.path.exists(os.path.join(dir, output_dir)):
        os.mkdir(output_dir)

    for file in os.listdir(input_dir):
        threshold_mask(file, output_dir)
```

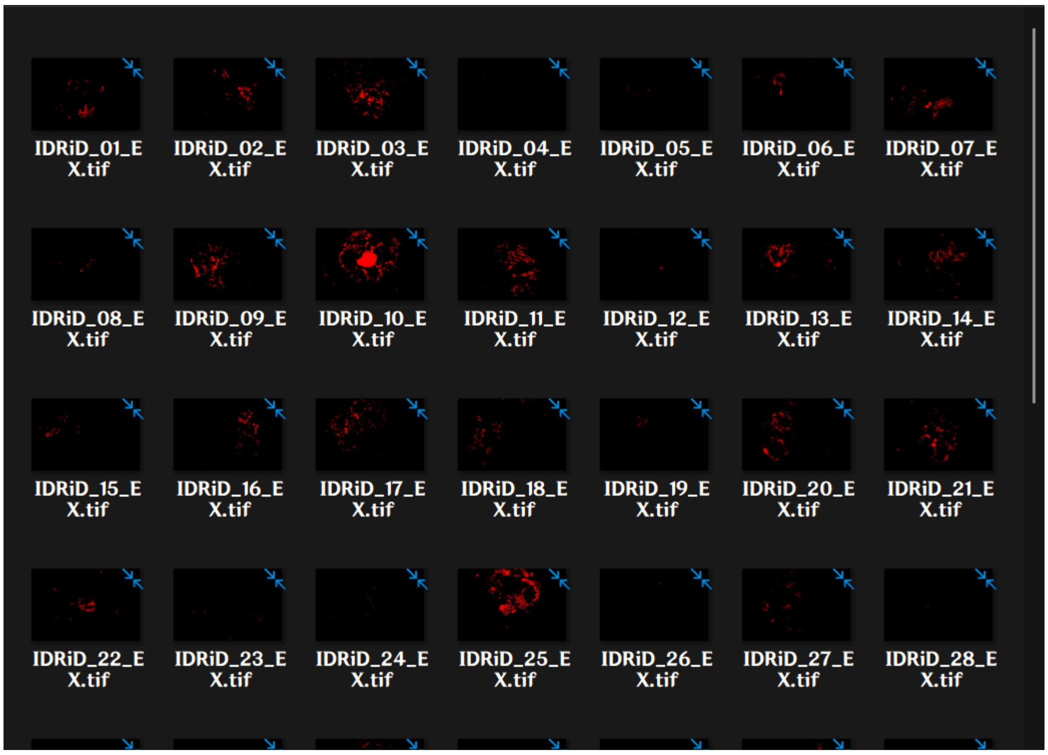
关键代码部分说明

- **灰度转换** (`im.convert('L')`):
  - 掩码图像转换为灰度，去除任何颜色信息，确保所有像素强度都在 0-255 范围内。
- **二值化阈值处理** (`gray.point(lambda x: 0 if x < 50 else 255, '1')`):
  - 应用像素强度阈值 `50` :
    - 值**低于** 50 的像素设为 `0` （黑色）。
    - 值**为 50 及以上**的像素设为 `255` （白色）。
  - 这确保了病灶和非病灶区域之间的**清晰区分**。
- **保存处理后的图像** (`bw.save(...)`):
  - 处理后的二值掩码以 JPEG 格式保存在 `training_set/` 目录中。

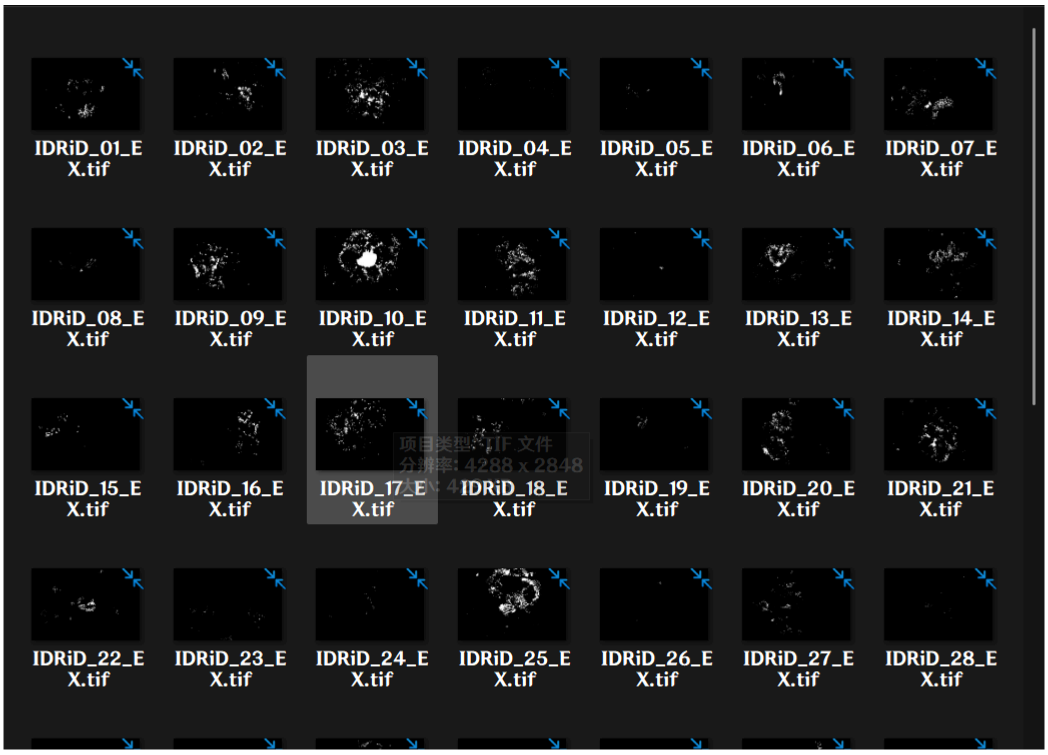
本步骤中的工作

- 我将**掩码图像转换为灰度**并应用二值化阈值处理，以区分病灶区域和背景。

处理前:



处理后:



提取和分割图像块

由于视网膜图像**分辨率非常高**，由于内存限制，在全尺寸图像上训练深度学习模型是不切实际的。
因此，从**原始图像**和**二值掩码**中提取**大小为** `512 × 512` **像素的重叠图像块**。
维持包含病灶的图像块和不包含病灶的图像块之间的**平衡比例**，以确保训练不偏倚。

关键代码：`extract_patches.py`

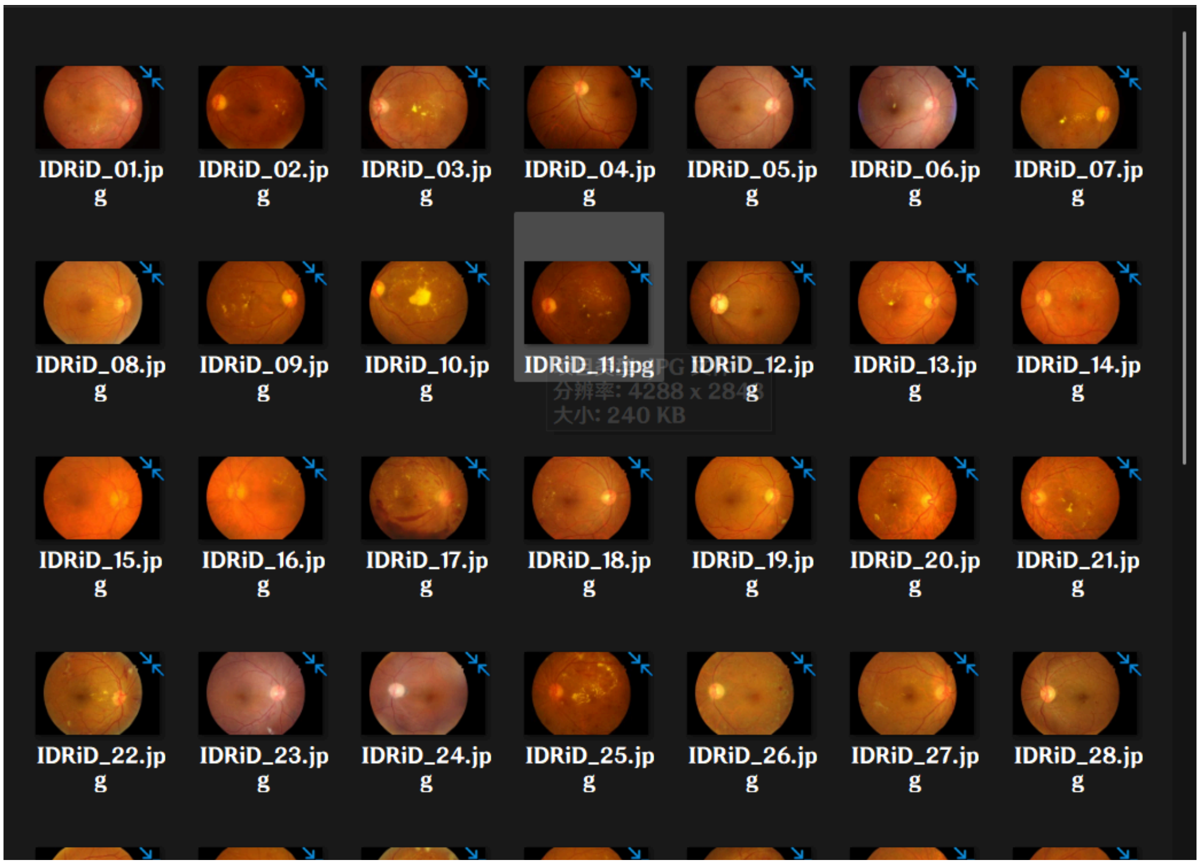
```
from PIL import Image
import os
import numpy as np

dir = os.getcwd()
output_dir_data = "patches/"
output_dir_mask = "labels/"

if not os.path.exists(os.path.join(dir, output_dir_data)):
    os.mkdir(output_dir_data)
if not os.path.exists(os.path.join(dir, output_dir_mask)):
    os.mkdir(output_dir_mask)

dir_data = os.path.join(dir, "images/")
dir_mask = os.path.join(dir, "training_set/")

negative_patches = []
positive_count = 0

for file in os.listdir(dir_mask):
    outfile = os.path.splitext(file)[0]
    extension = os.path.splitext(file)[1]
    if extension != ".tif":
        continue

    image_file = outfile.replace("_EX", "") + ".jpg"
    if not os.path.exists(os.path.join(dir_data, image_file)):
        print(f"Warning: Corresponding image file {image_file} not found for mask
{file}")
        continue

    im = Image.open(os.path.join(dir_mask, file))
    imd = Image.open(os.path.join(dir_data, image_file))

    patch_id = 0
    for i in range(10):
        for j in range(16):
            top_y = i * 256
            if i == 9:
                top_y = 2336
            top_x = j * 256
            if j == 15:
                top_x = 3776

            im_crop = im.crop((top_x, top_y, top_x + 512, top_y + 512))
            imd_crop = imd.crop((top_x, top_y, top_x + 512, top_y + 512))

            im_crop.save(os.path.join(output_dir_mask, f"{outfile}_p{patch_id}.tif"),
"TIFF", quality=100)
            imd_crop.save(os.path.join(output_dir_data, f"{outfile}_p{patch_id}.jpg"),
"JPEG", quality=100)

            if np.sum(np.array(im_crop)) < 100:
                negative_patches.append(output_dir_mask + outfile + "_p" +
str(patch_id) + extension)
            else:
                positive_count += 1

            patch_id += 1
```

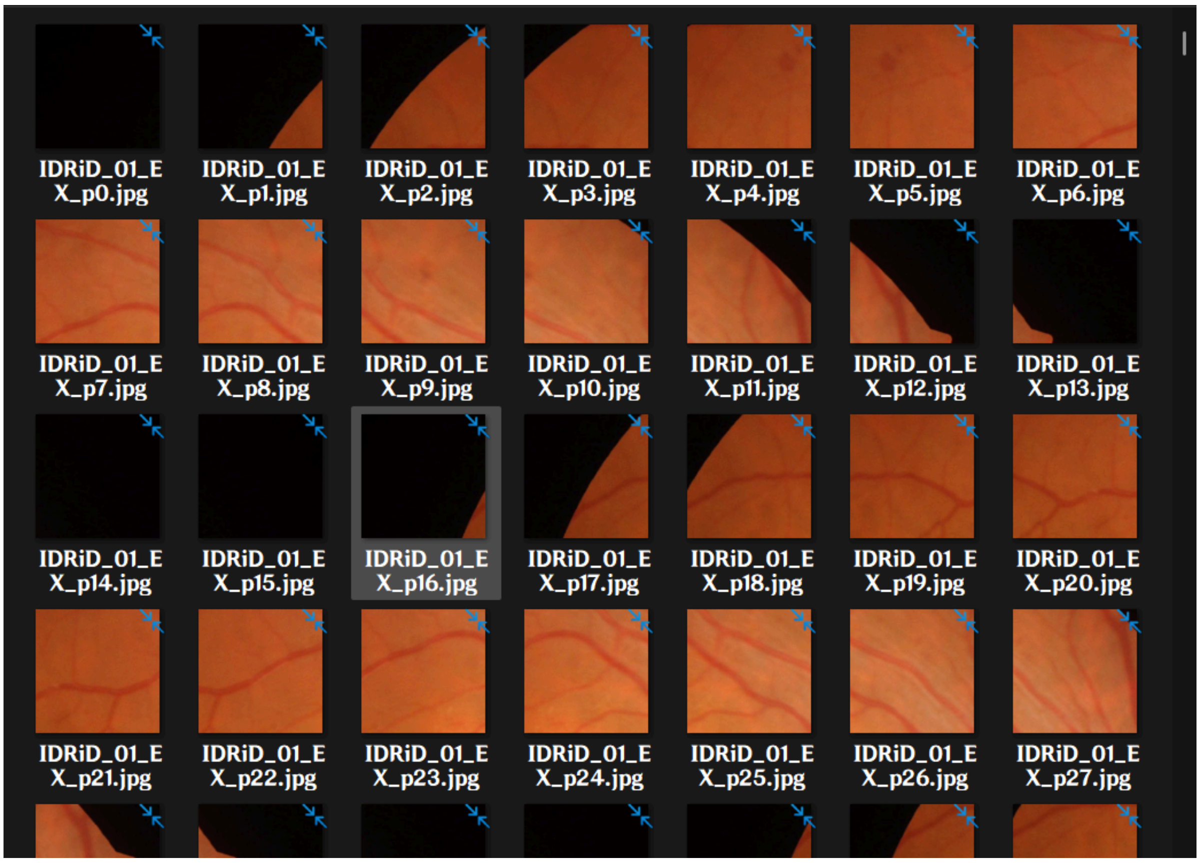
关键代码部分说明

- **滑动窗口图像块提取**
  - 图像被分割成 `512 × 512` 的**重叠图像块**，确保整个图像都被覆盖。
  - 256 **像素的步长**确保图像块重叠 50%，这有助于保留空间信息。
- **病灶与非病灶过滤**
  - **检查掩码**（`np.sum(np.array(im_crop)) < 100`）以确定图像块是否包含病灶。
  - 如果像素值总和**太低**，则将其分类为**负（非病灶）图像块**。

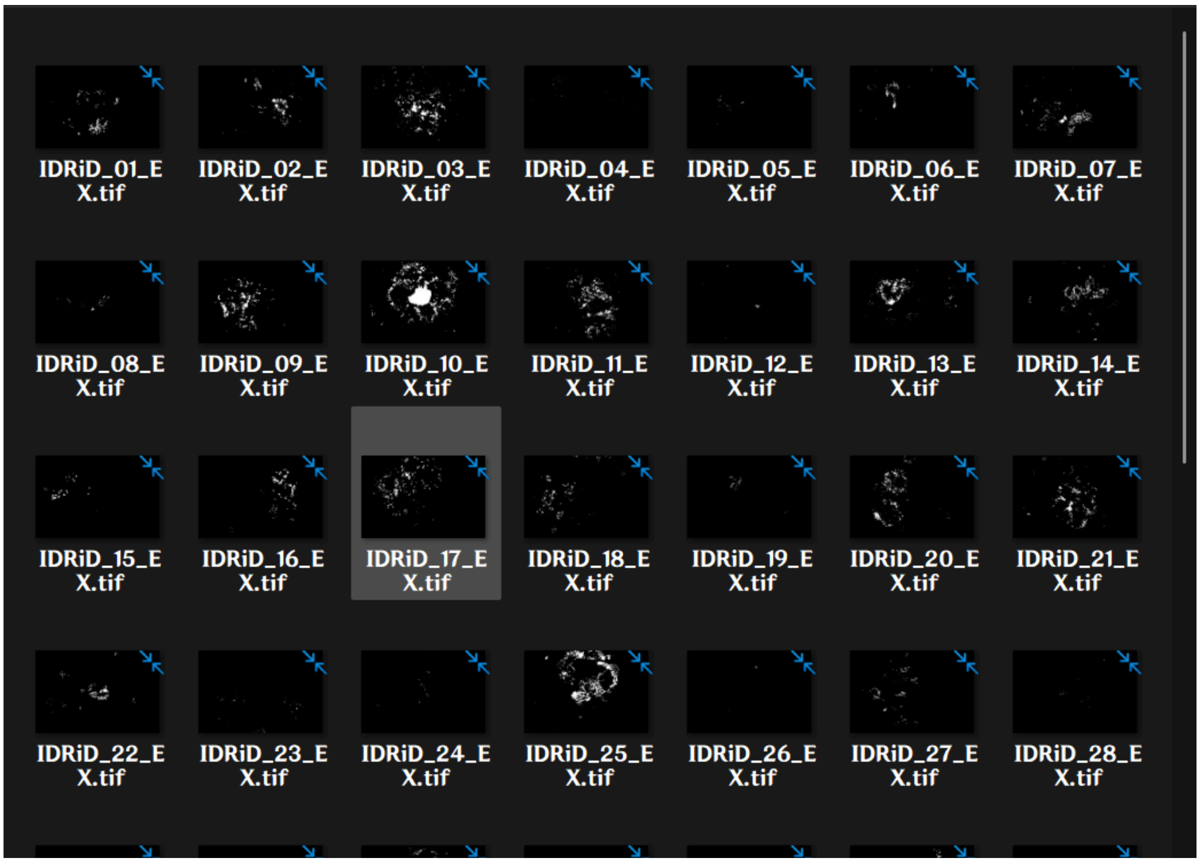
本步骤中的工作

- 我使用此脚本将**原始图像和灰度掩码**分割成更小的 `512×512` 图像块，同时确保数据集平衡。

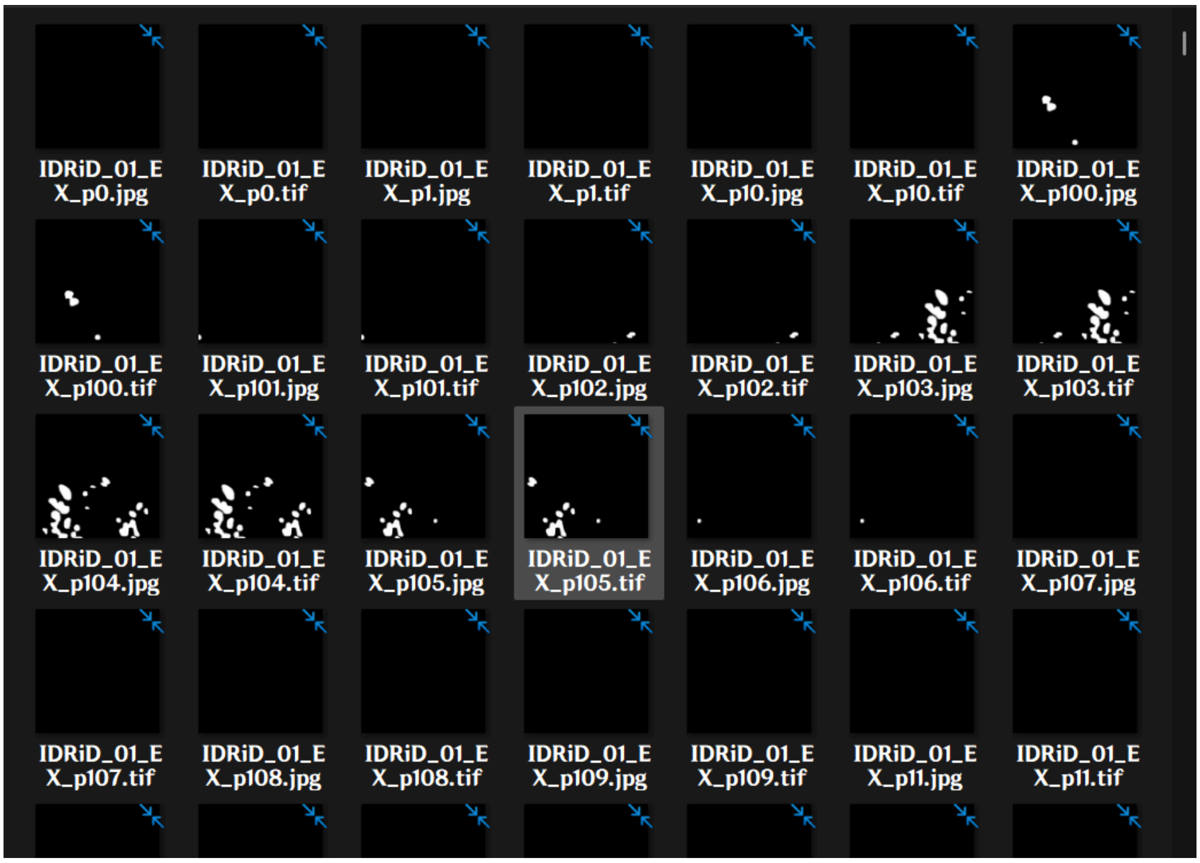




IDRiD_01_EX_p0.jpg IDRiD_01_EX_p1.jpg IDRiD_01_EX_p2.jpg IDRiD_01_EX_p3.jpg IDRiD_01_EX_p4.jpg IDRiD_01_EX_p5.jpg IDRiD_01_EX_p6.jpg
IDRiD_01_EX_p7.jpg IDRiD_01_EX_p8.jpg IDRiD_01_EX_p9.jpg IDRiD_01_EX_p10.jpg IDRiD_01_EX_p11.jpg IDRiD_01_EX_p12.jpg IDRiD_01_EX_p13.jpg
IDRiD_01_EX_p14.jpg IDRiD_01_EX_p15.jpg IDRiD_01_EX_p16.jpg IDRiD_01_EX_p17.jpg IDRiD_01_EX_p18.jpg IDRiD_01_EX_p19.jpg IDRiD_01_EX_p20.jpg
IDRiD_01_EX_p21.jpg IDRiD_01_EX_p22.jpg IDRiD_01_EX_p23.jpg IDRiD_01_EX_p24.jpg IDRiD_01_EX_p25.jpg IDRiD_01_EX_p26.jpg IDRiD_01_EX_p27.jpg



IDRiD_01_EX.tif IDRiD_02_EX.tif IDRiD_03_EX.tif IDRiD_04_EX.tif IDRiD_05_EX.tif IDRiD_06_EX.tif IDRiD_07_EX.tif
IDRiD_08_EX.tif IDRiD_09_EX.tif IDRiD_10_EX.tif IDRiD_11_EX.tif IDRiD_12_EX.tif IDRiD_13_EX.tif IDRiD_14_EX.tif
IDRiD_15_EX.tif IDRiD_16_EX.tif IDRiD_17_EX.tif IDRiD_18_EX.tif IDRiD_19_EX.tif IDRiD_20_EX.tif IDRiD_21_EX.tif
IDRiD_22_EX.tif IDRiD_23_EX.tif IDRiD_24_EX.tif IDRiD_25_EX.tif IDRiD_26_EX.tif IDRiD_27_EX.tif IDRiD_28_EX.tif



将图像块分割为训练集和验证集

提取图像块后，需要将它们分割为**训练集和验证集**，并将文件名存储在 `train.csv` 和 `test.csv` 中。这确保模型在**不同的图像块**上进行训练和评估，以防止过拟合。

关键代码：`traintest_split.py`

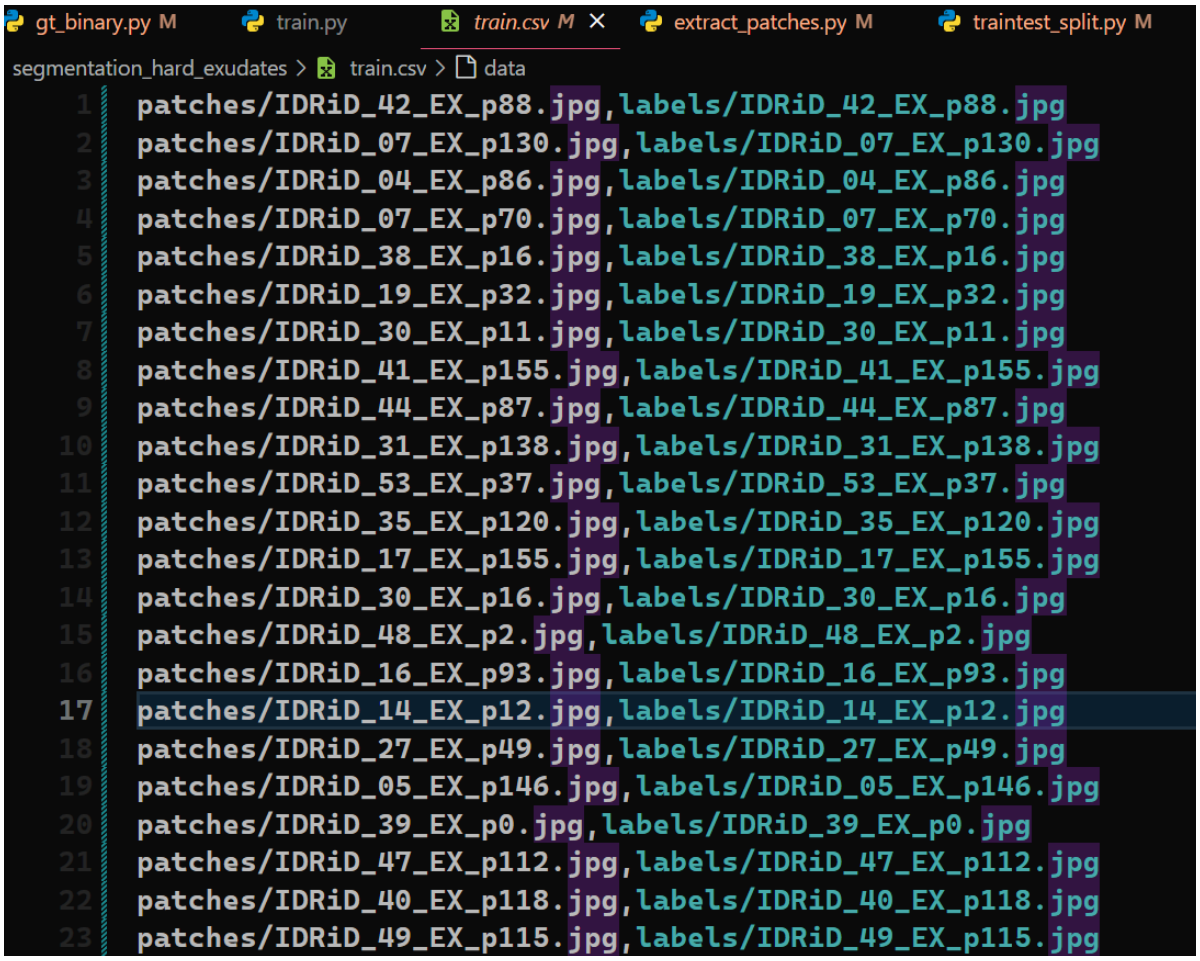
```python
import os
import random
import numpy as np

output_dir_data = "patches/"
output_dir_mask = "labels/"

all_patches = os.listdir(output_dir_data)
random.shuffle(all_patches)

split_ratio = 0.8
split_index = int(len(all_patches) * split_ratio)

train_patches = all_patches[:split_index]
test_patches = all_patches[split_index:]

np.savetxt("train.csv", train_patches, delimiter=",", fmt="%s")
np.savetxt("test.csv", test_patches, delimiter=",", fmt="%s")

print(f"Training patches: {len(train_patches)}, Testing patches: {len(test_patches)}")
```

关键代码部分说明

- **随机分割**（`random.shuffle(all_patches)`）
  - 图像块被随机打乱，以防止顺序偏差。
- 80-20 **数据分割**（`split_ratio = 0.8`）
  - 80% 的图像块用于训练。
  - 20% 的图像块用于验证。
- **保存图像块文件名**（`np.savetxt(...)`）
  - **训练图像块**的文件名保存在 `train.csv` 中。
  - **测试图像块**的文件名保存在 `test.csv` 中。

本步骤中的工作

- 我使用此脚本将**提取的图像块存储在** `train.csv` **和** `test.csv` **中**，用于训练和验证索引。



```
patches/IDRiD_42_EX_p88.jpg,labels/IDRiD_42_EX_p88.jpg
patches/IDRiD_07_EX_p130.jpg,labels/IDRiD_07_EX_p130.jpg
patches/IDRiD_04_EX_p86.jpg,labels/IDRiD_04_EX_p86.jpg
patches/IDRiD_07_EX_p70.jpg,labels/IDRiD_07_EX_p70.jpg
patches/IDRiD_38_EX_p16.jpg,labels/IDRiD_38_EX_p16.jpg
patches/IDRiD_19_EX_p32.jpg,labels/IDRiD_19_EX_p32.jpg
patches/IDRiD_30_EX_p11.jpg,labels/IDRiD_30_EX_p11.jpg
patches/IDRiD_41_EX_p155.jpg,labels/IDRiD_41_EX_p155.jpg
patches/IDRiD_44_EX_p87.jpg,labels/IDRiD_44_EX_p87.jpg
patches/IDRiD_31_EX_p138.jpg,labels/IDRiD_31_EX_p138.jpg
patches/IDRiD_53_EX_p37.jpg,labels/IDRiD_53_EX_p37.jpg
patches/IDRiD_35_EX_p120.jpg,labels/IDRiD_35_EX_p120.jpg
patches/IDRiD_17_EX_p155.jpg,labels/IDRiD_17_EX_p155.jpg
patches/IDRiD_30_EX_p16.jpg,labels/IDRiD_30_EX_p16.jpg
patches/IDRiD_48_EX_p2.jpg,labels/IDRiD_48_EX_p2.jpg
patches/IDRiD_16_EX_p93.jpg,labels/IDRiD_16_EX_p93.jpg
patches/IDRiD_14_EX_p12.jpg,labels/IDRiD_14_EX_p12.jpg
patches/IDRiD_27_EX_p49.jpg,labels/IDRiD_27_EX_p49.jpg
patches/IDRiD_05_EX_p146.jpg,labels/IDRiD_05_EX_p146.jpg
patches/IDRiD_39_EX_p0.jpg,labels/IDRiD_39_EX_p0.jpg
patches/IDRiD_47_EX_p112.jpg,labels/IDRiD_47_EX_p112.jpg
patches/IDRiD_40_EX_p118.jpg,labels/IDRiD_40_EX_p118.jpg
patches/IDRiD_49_EX_p115.jpg,labels/IDRiD_49_EX_p115.jpg
```

3. 网络训练

在 `Network Training` 部分，我们基于 `U-Net` 进行模型训练，同时在训练过程中进行数据增强，并计算损失函数（IOU 和 BCE）。

数据增强（`image_augmentation`）:

`image_augmentation` 函数对输入图像及其相应掩码应用一系列随机变换。增强包括：

- **随机翻转（左 ↔ 右和上 ↔ 下）**：这通过引入水平和垂直不变性来帮助模型泛化。
- **随机亮度**：随机调整图像亮度可以模拟各种光照条件，提高鲁棒性。
- **随机色相**：改变图像色相引入颜色变化，这对颜色很重要的分割任务很重要。

```python
def image_augmentation(image, mask):
    concat_image = tf.concat([image, mask], axis=-1)
    maybe_flipped = tf.image.random_flip_left_right(concat_image)
    maybe_flipped = tf.image.random_flip_up_down(concat_image)

    image = maybe_flipped[:, :, :-1]
    mask = maybe_flipped[:, :, -1:]

    image = tf.image.random_brightness(image, 0.7)
    image = tf.image.random_hue(image, 0.3)

    return image, mask
```

输入管道（`get_image_mask`）:

`get_image_mask` 函数负责从 CSV 文件加载图像和掩码，将它们解码成张量，并在指定时应用数据增强。

- **CSV 输入**：输入图像及其对应的掩码路径存储在 CSV 文件中。使用 `tf.TextLineReader` 读取 CSV 文件。
- **JPEG 解码**：使用 `tf.image.decode_jpeg` 将图像和掩码从 JPEG 格式解码为张量表示。
- **归一化**：掩码通过除以其最大值进行归一化，以将其缩放到 0 和 1 之间。
- **数据增强**：如果启用增强，它会调用 `image_augmentation` 函数。

```python
def get_image_mask(queue, augmentation=True):
    text_reader = tf.TextLineReader(skip_header_lines=1)
    _, csv_content = text_reader.read(queue)

    image_path, mask_path = tf.decode_csv(csv_content, record_defaults=[[""], [""]])

    image_file = tf.read_file(image_path)
    mask_file = tf.read_file(mask_path)

    image = tf.image.decode_jpeg(image_file, channels=3)
    image.set_shape([512, 512, 3])
    image = tf.cast(image, tf.float32)

    mask = tf.image.decode_jpeg(mask_file, channels=1)
    mask.set_shape([512, 512, 1])
    mask = tf.cast(mask, tf.float32)
    mask = mask / (tf.reduce_max(mask) + 1e-7)

    if augmentation:
        image, mask = image_augmentation(image, mask)
    return image, mask
```

卷积块（`conv_conv_pool`）:

`conv_conv_pool` 函数定义了一个典型的卷积块，它包括：

- **卷积**：具有指定滤波器数量和 3×3 核大小的 2D 卷积层。
- **批量归一化**：在每个卷积后添加，通过归一化激活来帮助稳定训练。
- **激活**：在批量归一化后应用 ReLU 激活函数。
- **池化（可选）**：应用最大池化操作以下采样特征图。

```
def conv_conv_pool(input_, n_filters, training, flags, name, pool=True,
activation=tf.nn.relu):
    net = input_

    with tf.variable_scope("layer{}".format(name)):
        for i, F in enumerate(n_filters):
            net = tf.layers.conv2d(
                net,
                F, (3, 3),
                activation=None,
                padding='same',
                kernel_regularizer=tf.contrib.layers.l2_regularizer(flags.reg),
                name="conv_{}".format(i + 1))
            net = tf.layers.batch_normalization(
                net, training=training, name="bn_{}".format(i + 1))
            net = activation(net, name="relu{}_{}".format(name, i + 1))

        if pool is False:
            return net

        pool = tf.layers.max_pooling2d(
            net, (2, 2), strides=(2, 2), name="pool_{}".format(name))

        return net, pool
```

上采样和连接（`upconv_concat`，`upconv_2D`）:

在 U-Net 中，下采样的特征图会上采样并与编码器路径（收缩路径）中的相应特征图连接，以提供详细的空间信息。

- **上卷积**：`upconv_2D` 函数使用转置卷积（也称为反卷积）执行上采样操作，将输入张量的空间维度加倍。
- **连接**：`upconv_concat` 函数将上采样的特征图与来自网络编码器部分的相应特征图连接起来。

```python
def upconv_concat(inputA, input_B, n_filter, flags, name):
    up_conv = upconv_2D(inputA, n_filter, flags, name)
    return tf.concat([up_conv, input_B], axis=-1, name="concat_{}".format(name))

def upconv_2D(tensor, n_filter, flags, name):
    return tf.layers.conv2d_transpose(
        tensor,
        filters=n_filter,
        kernel_size=2,
        strides=2,
        kernel_regularizer=tf.contrib.layers.l2_regularizer(flags.reg),
        name="upsample_{}".format(name))
```

U-Net 架构 (`make_unet`):

`make_unet` 函数通过堆叠多个卷积和池化层，然后是上采样和连接层，构建完整的 U-Net 模型。

- 网络架构由编码器（下采样部分）、瓶颈层和解码器（上采样部分）组成。每个编码器块由两个卷积层和一个最大池化操作组成。解码器块使用转置卷积（上卷积）并在每一步连接来自编码器的特征图。

```python
def make_unet(X, training, flags=None):
    net = X / 127.5 - 1  # Normalize input image to [-1, 1]
    conv1, pool1 = conv_conv_pool(net, [16, 16], training, flags, name=1)
    conv2, pool2 = conv_conv_pool(pool1, [32, 32], training, flags, name=2)
    conv3, pool3 = conv_conv_pool(pool2, [64, 64], training, flags, name=3)
    conv4, pool4 = conv_conv_pool(pool3, [128, 128], training, flags, name=4)
    conv5, pool5 = conv_conv_pool(pool4, [256, 256], training, flags, name=5)
    conv6, pool6 = conv_conv_pool(pool5, [512, 512], training, flags, name=6)
    conv7 = conv_conv_pool(pool6, [1024, 1024], training, flags, name=7, pool=False)

    up8 = upconv_concat(conv7, conv6, 512, flags, name=8)
    conv8 = conv_conv_pool(up8, [512, 512], training, flags, name=8, pool=False)

    up9 = upconv_concat(conv8, conv5, 256, flags, name=9)
    conv9 = conv_conv_pool(up9, [256, 256], training, flags, name=9, pool=False)

    up10 = upconv_concat(conv9, conv4, 128, flags, name=10)
    conv10 = conv_conv_pool(up10, [128, 128], training, flags, name=10, pool=False)

    up11 = upconv_concat(conv10, conv3, 64, flags, name=11)
    conv11 = conv_conv_pool(up11, [64, 64], training, flags, name=11, pool=False)

    up12 = upconv_concat(conv11, conv2, 32, flags, name=12)
    conv12 = conv_conv_pool(up12, [32, 32], training, flags, name=12, pool=False)

    up13 = upconv_concat(conv12, conv1, 16, flags, name=13)
    conv13 = conv_conv_pool(up13, [16, 16], training, flags, name=13, pool=False)

    return tf.layers.conv2d(conv13, 1, (1, 1), name='final', activation=None,
padding='same')
```

损失函数和指标（`BCE_`，`IOU_`）：

- **BCE 损失**：`BCE_` 函数计算带有类别权重的二元交叉熵损失。它为背景类别分配更高的权重（比例为 9:1）。
- **IoU（交并比）**：`IOU_` 函数计算 IoU 分数，这是分割任务常用的指标。它计算预测掩码和真实掩码之间的交集区域与并集区域的比率。

```
def BCE_(y_pred, y_true):
    class_weights = tf.constant([8], dtype=tf.float32)
    tensor_one = tf.constant([1], dtype=tf.float32)

    pred_flat = tf.reshape(y_pred, [-1, 1])
    true_flat = tf.reshape(y_true, [-1, 1])

    weight_map = tf.multiply(true_flat, class_weights)
    weight_map = tf.add(weight_map, tensor_one)

    loss_map = tf.nn.sigmoid_cross_entropy_with_logits(logits=pred_flat,
labels=true_flat)
    loss_map = tf.multiply(loss_map, weight_map)
    loss = tf.reduce_mean(loss_map)
    return loss

def IOU_(y_pred, y_true):
    H, W, _ = y_pred.get_shape().as_list()[1:]

    pred_flat = tf.reshape(y_pred, [-1, H * W])
    true_flat = tf.reshape(y_true, [-1, H * W])

    intersection = 2 * tf.reduce_sum(pred_flat * true_flat, axis=1) + 1e-7
    denominator = tf.reduce_sum(pred_flat, axis=1) + tf.reduce_sum(true_flat, axis=1) +
1e-7

    return tf.reduce_mean(intersection / denominator)
```

训练操作（`make_train_op`）：

`make_train_op` 函数定义了训练操作。它计算损失（使用 BCE 损失函数）并使用 Adam 优化器最小化损失。

```
def make_train_op(y_pred, y_true):
    loss = BCE_(y_pred, y_true)
    global_step = tf.train.get_or_create_global_step()
    optim = tf.train.AdamOptimizer(1e-4)
    return optim.minimize(loss, global_step=global_step)
```

训练循环（`main`）：

`main` 函数协调整个训练过程：

- 读取训练和测试 CSV 文件。
- 初始化 TensorFlow 图并启动会话。
- 迭代指定的 epoch 数，计算损失和 IOU 分数，并保存模型检查点。

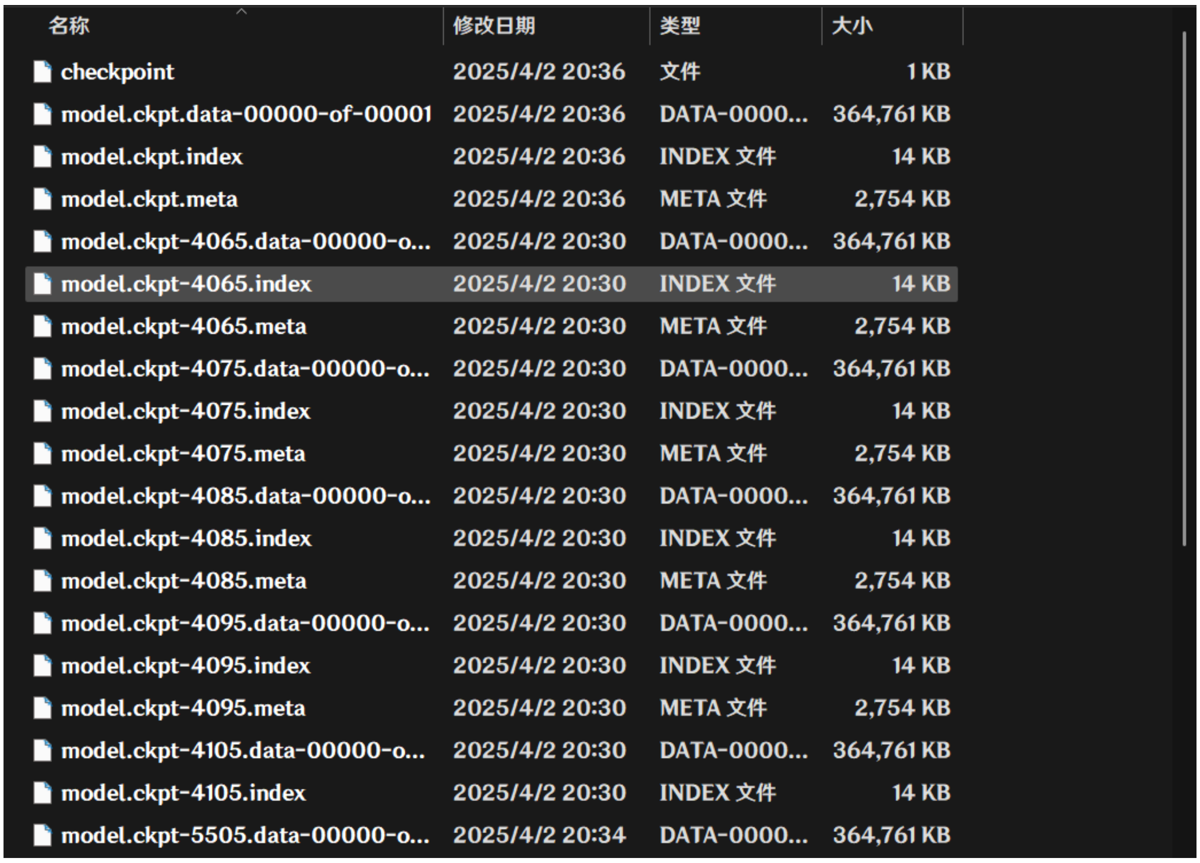
```python
def main(flags):
    # Initialization and training loop
    ...
    for epoch in range(flags.epochs):
        for step in range(0, n_train, flags.batch_size):
            ...
    saver.save(sess, "{}/model.ckpt".format(flags.ckdir))
```

这段代码训练了一个用于语义分割的 U-Net 模型，使用数据增强技术进行数据预处理，并采用带有类别权重的二元交叉熵损失和交并比作为指标。

训练问题：

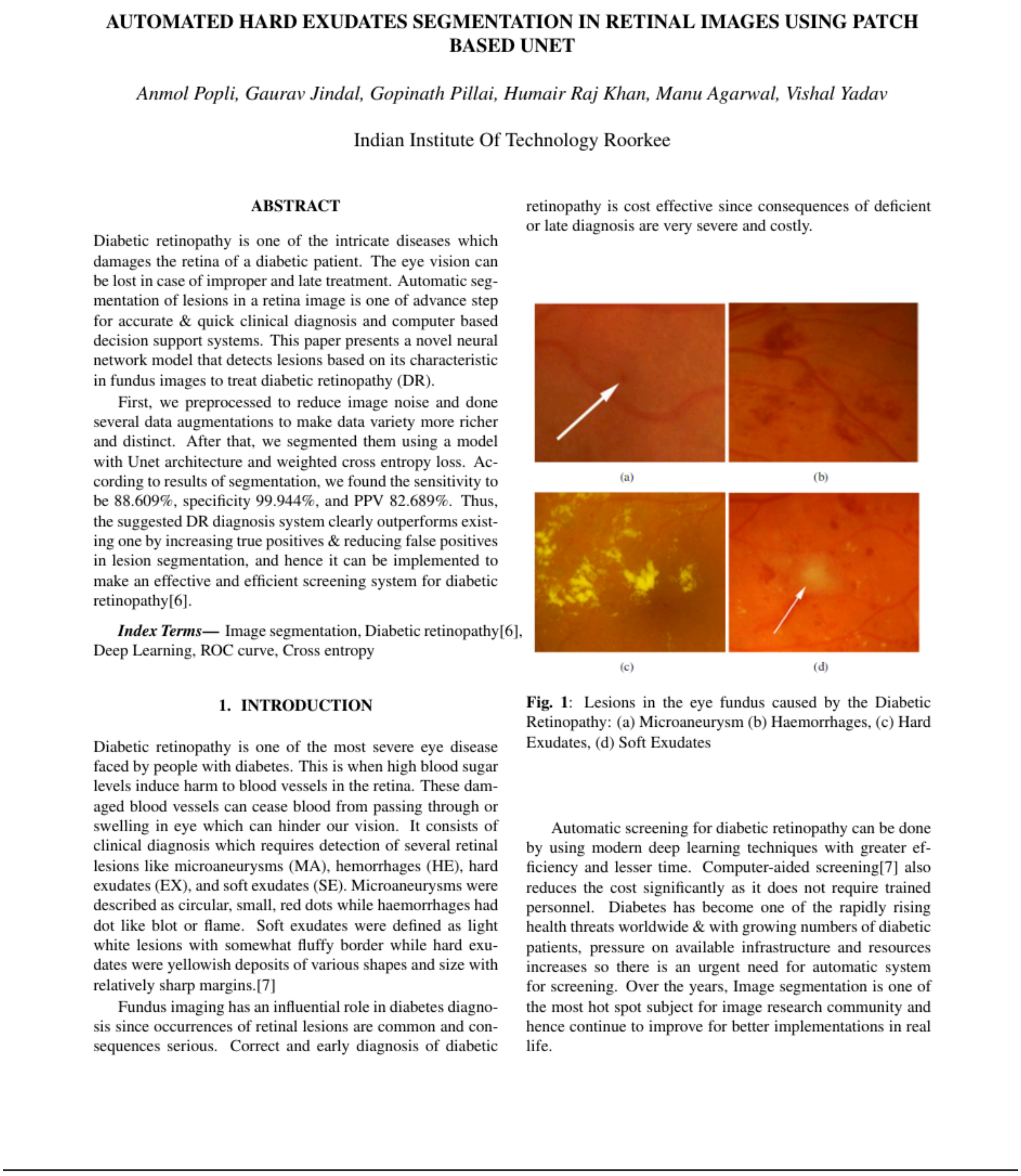
- **IOU 计算出现** NaN: 可能是输入数据的问题，例如 mask 全为 0，导致 `denominator = 0`。
- Windows **兼容性问题**: 论文环境较老，可能需要调整 TensorFlow 版本或改用 Linux 运行。
- **GPU显存不够**：所用的GPU显存只有4GB，无法达到原实验GTX1080的显存要求。

训练结果(暂未完成)：



名称	修改日期	类型	大小
checkpoint	2025/4/2 20:36	文件	1 KB
model.ckpt.data-00000-of-00001	2025/4/2 20:36	DATA-0000...	364,761 KB
model.ckpt.index	2025/4/2 20:36	INDEX 文件	14 KB
model.ckpt.meta	2025/4/2 20:36	META 文件	2,754 KB
model.ckpt-4065.data-00000-o...	2025/4/2 20:30	DATA-0000...	364,761 KB
model.ckpt-4065.index	2025/4/2 20:30	INDEX 文件	14 KB
model.ckpt-4065.meta	2025/4/2 20:30	META 文件	2,754 KB
model.ckpt-4075.data-00000-o...	2025/4/2 20:30	DATA-0000...	364,761 KB
model.ckpt-4075.index	2025/4/2 20:30	INDEX 文件	14 KB
model.ckpt-4075.meta	2025/4/2 20:30	META 文件	2,754 KB
model.ckpt-4085.data-00000-o...	2025/4/2 20:30	DATA-0000...	364,761 KB
model.ckpt-4085.index	2025/4/2 20:30	INDEX 文件	14 KB
model.ckpt-4085.meta	2025/4/2 20:30	META 文件	2,754 KB
model.ckpt-4095.data-00000-o...	2025/4/2 20:30	DATA-0000...	364,761 KB
model.ckpt-4095.index	2025/4/2 20:30	INDEX 文件	14 KB
model.ckpt-4095.meta	2025/4/2 20:30	META 文件	2,754 KB
model.ckpt-4105.data-00000-o...	2025/4/2 20:30	DATA-0000...	364,761 KB
model.ckpt-4105.index	2025/4/2 20:30	INDEX 文件	14 KB
model.ckpt-5505.data-00000-o...	2025/4/2 20:34	DATA-0000...	364,761 KB

论文图例：



AUTOMATED HARD EXUDATES SEGMENTATION IN RETINAL IMAGES USING PATCH BASED UNET

Anmol Popli, Gaurav Jindal, Gopinath Pillai, Humair Raj Khan, Manu Agarwal, Vishal Yadav

Indian Institute Of Technology Roorkee

ABSTRACT

Diabetic retinopathy is one of the intricate diseases which damages the retina of a diabetic patient. The eye vision can be lost in case of improper and late treatment. Automatic segmentation of lesions in a retina image is one of advance step for accurate & quick clinical diagnosis and computer based decision support systems. This paper presents a novel neural network model that detects lesions based on its characteristic in fundus images to treat diabetic retinopathy (DR).

First, we preprocessed to reduce image noise and done several data augmentations to make data variety more richer and distinct. After that, we segmented them using a model with Unet architecture and weighted cross entropy loss. According to results of segmentation, we found the sensitivity to be 88.609%, specificity 99.944%, and PPV 82.689%. Thus, the suggested DR diagnosis system clearly outperforms existing one by increasing true positives & reducing false positives in lesion segmentation, and hence it can be implemented to make an effective and efficient screening system for diabetic retinopathy[6].

***Index Terms*—** Image segmentation, Diabetic retinopathy[6], Deep Learning, ROC curve, Cross entropy

1. INTRODUCTION

Diabetic retinopathy is one of the most severe eye disease faced by people with diabetes. This is when high blood sugar levels induce harm to blood vessels in the retina. These damaged blood vessels can cease blood from passing through or swelling in eye which can hinder our vision. It consists of clinical diagnosis which requires detection of several retinal lesions like microaneurysms (MA), hemorrhages (HE), hard exudates (EX), and soft exudates (SE). Microaneurysms were described as circular, small, red dots while haemorrhages had dot like blot or flame. Soft exudates were defined as light white lesions with somewhat fluffy border while hard exudates were yellowish deposits of various shapes and size with relatively sharp margins.[7]

Fundus imaging has an influential role in diabetes diagnosis since occurrences of retinal lesions are common and consequences serious. Correct and early diagnosis of diabetic retinopathy is cost effective since consequences of deficient or late diagnosis are very severe and costly.

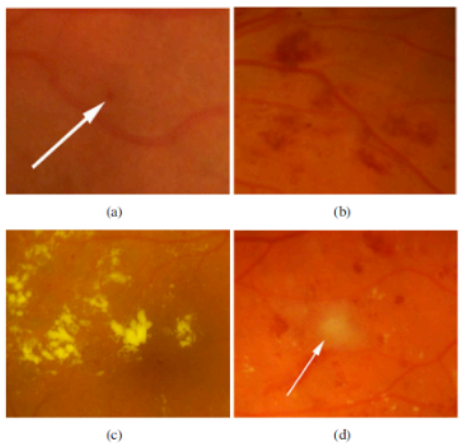


Fig. 1: Lesions in the eye fundus caused by the Diabetic Retinopathy: (a) Microaneurysm (b) Haemorrhages, (c) Hard Exudates, (d) Soft Exudates

Automatic screening for diabetic retinopathy can be done by using modern deep learning techniques with greater efficiency and lesser time. Computer-aided screening[7] also reduces the cost significantly as it does not require trained personnel. Diabetes has become one of the rapidly rising health threats worldwide & with growing numbers of diabetic patients, pressure on available infrastructure and resources increases so there is an urgent need for automatic system for screening. Over the years, Image segmentation is one of the most hot spot subject for image research community and hence continue to improve for better implementations in real life.

2. PRE-PROCESSING OF RETINAL IMAGES

The training dataset contains 143 fundus images and its hard exudates masks. Each image is of size (Width * Height) of (4288 * 2848). Out of all 143 images, 54 images has hard exudates lesions while others do not. An example of fundus image and its lesion mask is shown below. Images of diabetic

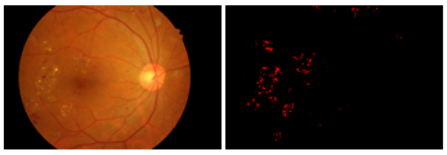

(a) Fundus Image (b) Lesion Mask

Fig. 2: Retinal Fundus and mask

retinopathy are very few and the area containing hard exudates lesions in the fundus image is very low as compared to total area of the image. So in order to increase the quantity of retinal images and reduce the negative segments, we have divided the images in various small patches. To maintain a balance in the training data, we have reduced the number of non-lesion patches to make it equal to the number of lesion patches we were able to extract. We have created a patch of size (512* 512) pixels and slide it with stride of 256 pixels as done in the case of convolution filters. Through this, we have created 160 fundus image patches and its corresponding lesion masks before reduction, which helps us to increase our data. We have slided with stride of 256 pixels which gives us overlapping patches and hence enables us to capture the surrounding context of lesions that are halved by one patch but those lesions will be fully included with its surroundings in the next patch. Hence, overlapping patches of stride 256 gives better results than distinct patches of stride 512. For the training data of hard exudates, this patch extraction algorithm produced 6056 patches from the available 143 images.

To increase the diversity in our data, we have included several augmentations like random left-right flip, random up-down flip, random brightness and hue changes etc. This technique of randomly mirroring and jittering the images helps us to increase variations in our data and make our model more robust and efficient to unseen images.

3. LESION SEGMENTATION TECHNIQUE

The architecture used in the model is Unet[10] which includes a shrinking path to capture context of surrounding and a symmetric expanding path that enables accurate localization. This is an example of Unet architecture shown above here: We improve and extend this architecture such that it works with very few training images and produce more accurate lesion masks.[10] The fundamental idea is to supplement a usual contracting network by successive layers, where up-sampling operators replaces pooling operators. Hence, these layers enhance the resolution of the output image. For localisation, high resolution features from the contracting layers are added in upsampled output. A successive convolution layer can then learn to yield a more precise result.

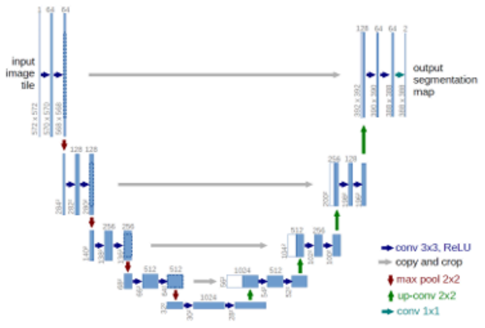


Fig. 3: U-net architecture (example for 32x32 pixels in the lowest resolution).

The network architecture consists of a contracting path (left side) and an expansive path (right side). The contracting path includes the standard structure of a convolutional network & the iterated application of two 3x3 convolutions. Each contracting layer is followed by batch normalisation[9] and then activation function as rectified linear unit (ReLU). It also consists of a 2x2 max pooling operation with stride 2 for downsampling[10]. We started with 16 feature channels of 512*512 patch image and at each downsampling step we double the number of feature channels till 1024 feature channels. Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 up-convolution that halves the number of feature channels. In expansive path the upsampling output is concatenated with the corresponding feature map[10] from the contracting path, and two 3x3 convolutions, each followed by a batch normalisation layer & ReLU. At the final layer a 1x1 convolution is used to map each 16-component feature vector to the desired number of classes.[8]

In the model, we have used pixelwise weighted binary cross entropy as loss function. In this, weightage given to losses of lesion and non lesion pixels is in the ratio 3:1. we have given more weightage to lesion pixels than normal as they are very less in area. We dont want to miss any lesion in the image so more weightage is given to them. The optimum weightage was chosen after iterating over several other weights. We train by Adam Optimizer with an initial learning rate of 1e-4, and exponential decay rates for 1st and 2nd moment estimates as 0.9 and 0.999 respectively. We used a batch size of 8 for 100 epochs.

The output results were the probability corresponding to each pixel to be a lesion pixel. So after taking a certain threshold, we got the lesion mask for each image. This optimum threshold is chosen from FROC curve at which true positive rate is maximum without compromising on false positive rate. All models are trained and tested with Tensorflow on a single NVIDIA GTX 1080 GPU.

4. PERFORMANCE EVALUATION OF LESION SEGMENTATION

Intensity image with particular abnormality segmented as foreground and rest part of an image as background. Methods used for lesion segmentation of diabetic retinopathy are evaluated by using sensitivity, specificity, positive predictive value[9] and finally, the detection performance are summarized using Free Response Operating Characteristic (FROC) curves. Sensitivity is the percentage of actual lesion area classified as lesion in the predicted mask and specificity is the percentage of actual normal retinal area classified as normal(not lesion) in predicted mask. PPV is the percentage of predicted lesion area which is actual lesion area. The FROC curve is created by plotting the true positive rate (TPR) against the false positive (FP) area per image at various threshold settings.

$$Sensitivity = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{TN + FP}$$

$$PPV = \frac{TP}{TP + FP}$$

Sensitivity, specificity and PPV can be calculated as where TP is the area of actual lesions found as lesions in predicted mask , TN is the area of not lesions found as not lesions, FP is the area of not lesions found as lesions (false positives) and FN is the area of actual lesions found as not lesions (false negatives).

5. RESULTS

The output results are the probability corresponding to each pixel to be a lesion pixel. So after taking a certain threshold, we got the lesion mask for each image. This optimum threshold is chosen from FROC curve at which true positive rate is maximum without compromising on false positive rate. All models are trained and tested with Tensorflow on a single NVIDIA GTX 1080 GPU.

After training the model and tuning its parameter, we have segmented almost all the lesions with exact size and shape at precise locations. These are the sensitivity, specificity and PPV value: The loss function was continuously decreasing

Table 1: Evaluation Metrics

Sensitivity	0.88609
Specificity	0.99944
Positive Predictive Value	0.82689

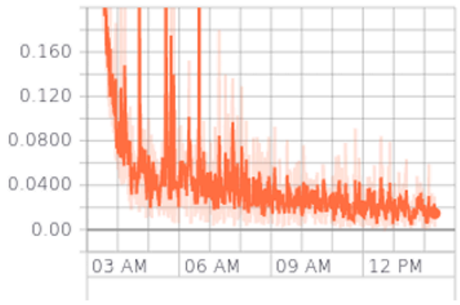


Fig. 4: Binary Cross Entropy loss plotted against training time

as we further train over epochs. This is the FROC curve which helps us to find the optimum threshold for making the mask.

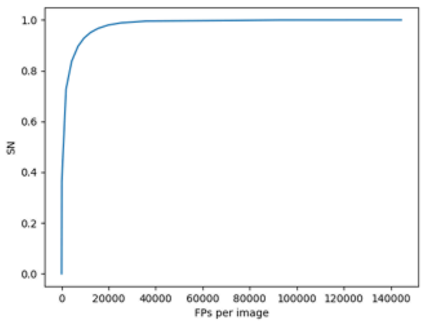


Fig. 5: FROC Curve: SN plotted against average FP area per image

6. CONCLUSION

This paper proposed a novel method for automated lesion detection in fundus images. To develop this robust lesion segmentation model, we have done data augmentation and provided the small patches of image instead of single large image. After training the model and hyper parameter tuning, we have detected almost all the lesions with correct shape and