B档模型Duck-Net

--by 2213409 石彬辰

准备工作

由于本次课程提供的两个数据集都是png格式,在ImageLoader目录下找到ImageLoader2D.py文件,修改dataset数据集类型判定,如果传入dataset为aiot,图像后缀都为.png

ImageLoader2D.py

```python
'''
    由于本次课程aiot使用的数据集都是png格式,增加数据集类型aiot
'''
if dataset == 'cvc-colondb' or dataset == 'etis-laribpolypdb' or dataset == 'aiot':
# if dataset == 'cvc-colondb' or dataset == 'etis-laribpolypdb':
    train_ids = glob.glob(IMAGES_PATH + "*.png")
```

设置自己的数据集路径

ImageLoader2D.py

```python
'''
    修改训练数据集路径
'''
#folder_path = "data/BUSI-256/"  # Add the path to your data directory
folder_path="data/isic2018/train/"
```

由于训练过程中mask标签图像必须是灰度图,即8位深度.但部分图像是24位深度RGB图像,故在ImageLoader目录下创建masksConvert.py来做mask图像转换(需要在训练前运行masksConvert.py转换完成)

masksConvert.py

```python
from PIL import Image
import numpy as np
import os
def maskConvert(mask_path):
    mask=Image.open(mask_path)
    if mask.mode=='RGB':
```

```
        mask=mask.convert('L')
        print(mask_path)
        mask.save(mask_path)
if __name__=='__main__':
    #masks_path='../data/BUSI-256/masks/'
    masks_path='../data/isic2018/train/masks/'
    images=os.listdir(masks_path)
    for img in images:
        maskConvert(masks_path+img)
```

创建config目录,在config目录下创建log.py用于记录训练过程

logs函数根据调用时刻创建日志目录,例如logs/2025_04_20_14_11_57/目录下保存训练的模型,其子目录train_log用于保存训练loss数据train_loss.txt

EpochLossLog继承keras.callbacks包中的Callback类,创建on_train_begin函数用于在训练开始时以写方式打开train_loss.txt,创建on_epoch_end函数用于在单论训练结束时将本轮损失率loss写入train_loss.txt,同时判断当前验证集上损失率val_loss是不是最低,如果最低打印信息表示最佳模型更新,创建on_train_end函数用于在训练完成时关闭train_loss.txt文件

log.py

```
import datetime
import os
from keras.callbacks import Callback
def logs():
    now = datetime.datetime.now()
    formatted_date = now.strftime("%Y_%m_%d_%H_%M_%S")
    log_path = 'logs/'
    if not os.path.exists(log_path):
        os.mkdir(log_path)
    os.mkdir(log_path + formatted_date)
    os.mkdir(os.path.join(log_path + formatted_date,'train_log'))
    return log_path+formatted_date

class EpochLossLog(Callback):
    def __init__(self,file_path,model_path):
        super().__init__()
        self.file_path = file_path
        self.file = None
        self.best_val_loss = None
        self.model_path=model_path

    def on_train_begin(self, logs=None):
        self.file = open(self.file_path, 'w', encoding='utf-8')

    def on_epoch_end(self, epoch, logs=None):
```

```python
            self.file.write(f"{logs['loss']}\n")
            self.file.flush()
            if logs is not None:
                print(f"Total Loss: {logs.get('loss'):.3f}")
            val_loss = logs.get('val_loss')
            if self.best_val_loss is None or val_loss < self.best_val_loss:
                self.best_val_loss = val_loss
                print(f"Saving best model to {self.model_path}")

        def on_train_end(self, logs=None):
            self.file.close()
```

在config目录下创建Progressbar.py来做一个可实时查看训练进度的进度条

EpochProgressBar继承tensorflow包下的tensorflow.keras.callbacks.Callback类,定义on_epoch_begin函数在每轮训练开始时新建进度条,on_train_batch_end用于在epoch内每步batch结束时更新进度条,on_epoch_end用于在epoch结束时关闭进度条

Progressbar.py

```python
import tensorflow as tf
from tqdm import tqdm
class EpochProcessBar(tf.keras.callbacks.Callback):
    def __init__(self,total_epochs):
        super().__init__()
        self.total_epochs=total_epochs
        self.epoch_bar=None

    def on_epoch_begin(self, epoch, logs=None):
        self.epoch_pbar = tqdm(
            total=self.params['steps'],
            desc=f"Epoch {epoch + 1}/{self.total_epochs}",
            unit='batch',
            dynamic_ncols=True
        )

    def on_train_batch_end(self, batch, logs=None):
        # 更新进度条并显示当前指标
        self.epoch_pbar.update(1)
        self.epoch_pbar.set_postfix({
            'loss': f"{logs['loss']:.4f}",
            'accuracy': f"{logs['accuracy']:.4f}"
        })

    def on_epoch_end(self, epoch, logs=None):
        self.epoch_pbar.close()
```

```
Epoch 65/100: 100%|██████████| 252/252 [01:27<00:00,  2.89batch/s, loss=0.0755, accuracy=0.9883]
Total Loss: 0.076
Saving best model to best_epoch_model.h5
```

在log目录下创建epoch_loss.py根据train_loss.txt来生成轮次损失图epoch_loss.png

plot_train_curve函数接受loss数据所在目录位置,生成两条曲线,train-loss曲线(红色实线)和平滑smooth train-loss曲线(绿色虚线)

epoch_loss.py

```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
def plot_training_curve(train_log_path):
    train_file = os.path.join(train_log_path, 'train_loss.txt')
    output_path=os.path.join(train_log_path,'epoch_loss.png')
    loss_values=[]
    with open(train_file,'r') as f:
        for line in f:
            loss=float(line.strip())
            loss_values.append(loss)
    if not loss_values:
        print("训练日志为空，无法生成图表")
        return
    epochs=np.arange(1,len(loss_values)+1)
    #print(epochs)
    #print(loss_values)
    log_df=pd.DataFrame({'epoch': epochs, 'loss': loss_values})
    #print(log_df)
    # 动态获取轮次和损失范围
    max_epoch = log_df['epoch'].max()
    min_loss = log_df['loss'].min()
    max_loss = log_df['loss'].max()

    # 自动调整纵轴范围（留10%缓冲空间）
    loss_buffer = (max_loss - min_loss) * 0.1
    y_min = max(0, min_loss - loss_buffer)  # 确保不低于0
    y_max = max_loss + loss_buffer

    # 自动调整横轴刻度步长
    epoch_step = max(1, int(max_epoch / 5))  # 至少显示5个刻度

    # 动态调整平滑参数（基于数据长度）
    sigma = max(1.0, 30 / len(log_df))
```

```python

    # 生成平滑曲线
    smooth_loss = gaussian_filter1d(log_df['loss'], sigma=sigma)

    # 创建图表
    plt.figure(figsize=(10, 6))
    plt.plot(log_df['epoch'], log_df['loss'], label='train loss',
linewidth=2.5, color='#d62728')
    plt.plot(log_df['epoch'], smooth_loss, label='smooth train loss',
linewidth=2.5, linestyle='--',color='#007B00')

    # 设置动态坐标轴
    plt.xlim(0, max_epoch)
    plt.ylim(y_min, y_max)
    plt.xticks(range(0, max_epoch + 1, epoch_step))

    # 样式设置
    plt.xlabel("Epoch", fontsize=12, fontweight='bold')
    plt.ylabel("Loss", fontsize=12, fontweight='bold')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend(frameon=True, shadow=True)

    plt.tight_layout()
    plt.savefig(output_path, dpi=350, bbox_inches='tight')
    plt.close()

if __name__=='__main__':
    #功能测试
    plot_training_curve("log")
```

创建train.py用于训练模型

设置参数os.environ["CUDA_VISIBLE_DEVICES"] = "0"来启用gpu加速,img_shape标定图像两个方向的像素量,data_set设置数据集类型, epochs 设置训练轮数,batch_size 设置训练步长,start_filters设置过滤器数量,由于本次训练使用的显卡是3060laptop 6G,经过调整batch_size = 2, start_filters = 8是能运行的最高设置,可根据配置调整,调用ImageLoader2D.py中的load_data函数用于载入数据集并将数据集向量化,设置split来调整训练集和验证集的比例,调用Duck_Net.py中的create_model函数用于创建模型,model.compile编译模型,callbacks内,调用了进度条和日志函数,三个ModelCheckpoint检查点分别用于在本轮验证集损失率为全局最小时更新最佳模型,每轮训练结束时更新最新模型,每5轮训练保存过程模型,注释的早停EalyStoppping用于在patience轮训练后模型性能没有提升(验证集损失率为出现最低记录)时停止训练,防止过拟合,不过前面已有更新最佳模型的函数调用,故注释早停函数用于观察模型训练损失率的变化, plot_training_curve函数用于在训练完全结束时生成训练轮数-训练损失率图像epoch-loss.png

```python
import tensorflow as tf
import os
from ImageLoader.ImageLoader2D import load_data
from ModelArchitecture.DUCK_Net import create_model
from ModelArchitecture.DiceLoss import dice_metric_loss
from config.Progressbar import EpochProcessBar
from keras.callbacks import ModelCheckpoint
from config.log import logs,EpochLossLog
from config.epoch_loss import plot_training_curve
if __name__=='__main__':
    log_dir=logs()
    os.environ["CUDA_VISIBLE_DEVICES"] = "0"
    img_shape = [256, 256]
    batch_size = 2
    epochs = 100
    start_filters = 8
    dataset = 'aiot'
    X_train, Y_train = load_data(
        img_height=img_shape[0],
        img_width=img_shape[1],
        images_to_be_loaded=-1,
        dataset=dataset
    )
    split = int(0.8 * len(X_train))
    X_train, X_val = X_train[:split], X_train[split:]
    Y_train, Y_val = Y_train[:split], Y_train[split:]

    model = create_model(
        img_height=img_shape[0],
        img_width=img_shape[1],
        input_chanels=3,
        out_classes=1,
        starting_filters=start_filters
    )

    model.compile(
        optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
        loss=dice_metric_loss,
        metrics=['accuracy']
    )
    steps_per_epoch = len(X_train) // batch_size
    callbacks = [
        EpochProcessBar(total_epochs=epochs),
EpochLossLog(os.path.join(log_dir,'train_log/train_loss.txt'),'best_epoch_model.h5'),
        ModelCheckpoint(
```

```
            os.path.join(log_dir, 'best_epoch_model.h5'),
            monitor='val_loss',
            save_best_only=True,
            save_weights_only=False,  # 保存完整模型
            mode='min',
            verbose=0,
            message='111111'
        ),
        ModelCheckpoint(
            os.path.join(log_dir, 'last_epoch_model.h5'),
            save_weights_only=False,
            save_freq='epoch',  # 每个epoch保存一次
            verbose=0
        ),
        ModelCheckpoint(
            os.path.join(log_dir, 'epoch_{epoch:03d}.h5'),
            save_freq=5 * steps_per_epoch,  # 每5个epoch保存一次
            save_weights_only=False,
            verbose=0
        ),
        #tf.keras.callbacks.EarlyStopping(patience=10, restore_best_weights=True)#早停,耐心值为10,在10个epoch中性能没有提升会停止
    ]
    history = model.fit(
        X_train,
        Y_train,
        validation_data=(X_val, Y_val),
        batch_size=batch_size,
        epochs=epochs,
        verbose=0,  # 禁用默认输出
        callbacks=callbacks
    )
    plot_training_curve(os.path.join(log_dir,'train_log'))
    #model.save('ducknet_final.h5')
```

创建predict.py用于预测图像

load_and_preprocess_image函数用于加载单张图像, predict_single_image函数用于预测单张图像, predict_images_in_directory函数用于对整个目录的图像进行预测, load_trained_model函数用于加载训练好的模型

predict.py

```
import os
import numpy as np
import cv2
from PIL import Image
```

```python
from tqdm import tqdm
from ModelArchitecture.DUCK_Net import create_model


def load_and_preprocess_image(image_path, img_height, img_width):
    image = Image.open(image_path).convert('RGB')
    image = image.resize((img_height, img_width))
    image = np.array(image) / 255.0  # 归一化
    return np.expand_dims(image, axis=0)  # 增加批次维度
def predict_single_image(model, image_path, img_height, img_width):
    processed_image = load_and_preprocess_image(image_path, img_height, img_width)
    prediction = model.predict(processed_image)
    return prediction[0]
def predict_images_in_directory(model, images_dir, img_height, img_width, output_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    image_paths = [os.path.join(images_dir, fname) for fname in os.listdir(images_dir) if
                   fname.lower().endswith(('.png', '.jpg', '.jpeg'))]

    for image_path in tqdm(image_paths, desc="Processing images"):
        prediction = predict_single_image(model, image_path, img_height, img_width)

        prediction_mask = (prediction[..., 0] * 255).astype(np.uint8)
        prediction_mask = cv2.resize(prediction_mask, (img_width, img_height), interpolation=cv2.INTER_NEAREST)

        output_filename = os.path.basename(image_path)#.replace('.png', '_mask.png')
        output_path = os.path.join(output_dir, output_filename)
        cv2.imwrite(output_path, prediction_mask)
        print(f"Saved prediction to {output_path}")
def load_trained_model(model_path, img_height, img_width, input_channels, output_classes, starting_filters):
    model = create_model(img_height, img_width, input_channels, output_classes, starting_filters)
    model.load_weights(model_path)
    # model = tf.keras.models.load_model(model_path, custom_objects={'dice_metric_loss': dice_metric_loss})
    return model

if __name__ == '__main__':
    # 配置参数
```

```
    img_shape = [256, 256]
    input_channels = 3
    output_classes = 1
    starting_filters = 8
    # model_path = "logs/2025_04_20_20_21_03/best_epoch_model.h5"
    # images_dir = "data/BUSI-256/images/"
    # output_dir = "data/BUSI-256/predict/"
    model_path = "logs/2025_04_20_22_44_19/best_epoch_model.h5"
    images_dir='data/isic2018/test/images/'
    output_dir='data/isic2018/test/predict/'
    model = load_trained_model(model_path, img_shape[0], img_shape[1], input_channels, output_classes, starting_filters)
    predict_images_in_directory(model, images_dir, img_shape[0], img_shape[1], output_dir)
```

训练模型

BUSI-256数据集

在masksConvert.py中将masks_path改为训练集masks图像所在目录,运行,所有masks图像被转为灰度图

masksConvert.py

```
if __name__=='__main__':
    masks_path='../data/BUSI-256/masks/'
    #masks_path='../data/isic2018/train/masks/'
    images=os.listdir(masks_path)
```

在ImageLoader2D.py中修改正确的训练集路径

ImageLoader2D.py

```
'''
    修改训练数据集路径
'''
folder_path = "data/BUSI-256/"  # Add the path to your data directory
#folder_path="data/isic2018/train/"
```

在train.py中设置正确的训练参数,下图为使用显卡加速,图像尺寸为256*256,训练步长为2,训练100轮,8个过滤器,数据集类型为aiot

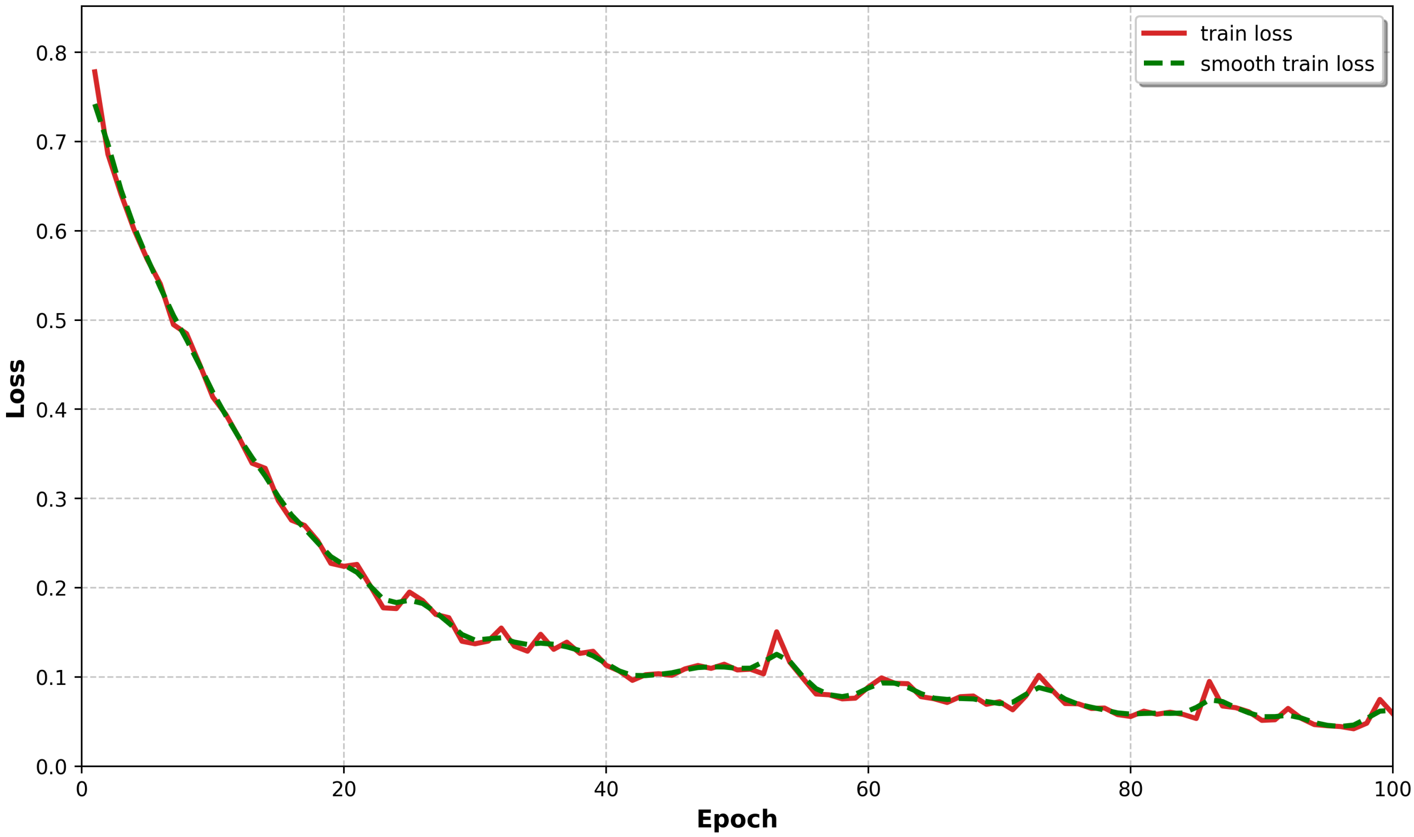
train.py

```
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
img_shape = [256, 256]
batch_size = 2
epochs = 100
start_filters = 8
dataset = 'aiot'
```

运行train.py开始训练,获得22个模型文件,其中20个为过程文件,1个为最佳验证损失率模型,一个为最新模型,train_log目录中train_log.txt保存每轮损失率信息,epoch_loss.png为训练损失率loss随轮次epoch变化的曲线图,log.txt为使用者自己创建保存的训练详细信息

代码块

```
D:\VISUAL STUDIO\CODE\PYCHARM\AIOT_DUCK_NET\LOGS\2025_04_20_20_21_03
│   best_epoch_model.h5
│   epoch_005.h5
│   epoch_010.h5
│   epoch_015.h5
│   epoch_020.h5
│   epoch_025.h5
│   epoch_030.h5
│   epoch_035.h5
│   epoch_040.h5
│   epoch_045.h5
│   epoch_050.h5
│   epoch_055.h5
│   epoch_060.h5
│   epoch_065.h5
│   epoch_070.h5
│   epoch_075.h5
│   epoch_080.h5
│   epoch_085.h5
│   epoch_090.h5
│   epoch_095.h5
│   epoch_100.h5
│   last_epoch_model.h5
│
└─train_log
        epoch_loss.png
        log.txt
        train_loss.txt
```



epoch-loss.png

当epoch=99时验证集损失率最小,此时最后一次保存最佳模型

虽然此时训练损失率不是最小,但验证损失率最小,判断模型性能不能只看训练损失率一个指标

代码块

```
Epoch 99/100: 100%|██████████| 252/252 [01:18<00:00,  3.19batch/s, loss=0.0747, accuracy=0.9871]
Total Loss: 0.075
Saving best model to best_epoch_model.h5
```

isic2018数据集

在masksConvert.py中修改isic训练集的目录

masksConvert.py

```
if __name__=='__main__':
    #masks_path='../data/BUSI-256/masks/'
    masks_path='../data/isic2018/train/masks/'
    images=os.listdir(masks_path)
```

在ImageLoader2D.py中修改isic训练集的目录

```
'''
    修改训练数据集路径
'''
#folder_path = "data/BUSI-256/"  # Add the path to your data directory
folder_path="data/isic2018/train/"
```

train.py中参数如下,运行

train.py

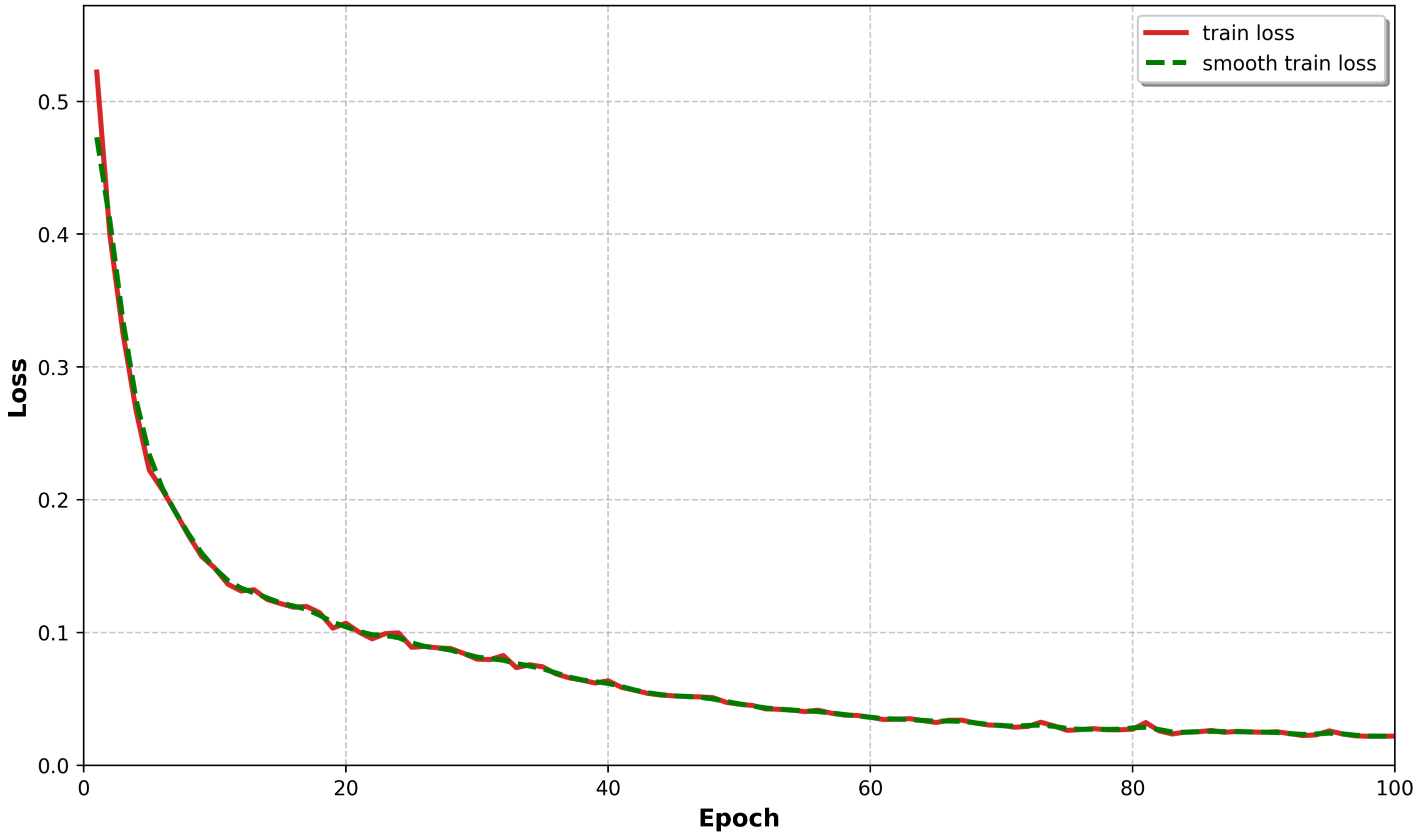
```
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
img_shape = [256, 256]
batch_size = 2
epochs = 100
start_filters = 8
dataset = 'aiot'
```

训练后的日志文件结构如下(log.txt为用户手动添加)

代码块

```
D:\VISUAL STUDIO\CODE\PYCHARM\AIOT_DUCK_NET\LOGS\2025_04_20_22_44_19
│  best_epoch_model.h5
│  epoch_005.h5
│  epoch_010.h5
│  epoch_015.h5
│  epoch_020.h5
│  epoch_025.h5
│  epoch_030.h5
│  epoch_035.h5
│  epoch_040.h5
│  epoch_045.h5
│  epoch_050.h5
│  epoch_055.h5
│  epoch_060.h5
│  epoch_065.h5
│  epoch_070.h5
│  epoch_075.h5
│  epoch_080.h5
│  epoch_085.h5
│  epoch_090.h5
│  epoch_095.h5
│  epoch_100.h5
│  last_epoch_model.h5
│
└─train_log
```

```
        epoch_loss.png
        log.txt
        train_loss.txt
```



epoch_loss.png

训练在epoch=74时取得最小的验证损失率,最后一次保存最佳模型

代码块

```
Epoch 74/100: 100%|██████████| 754/754 [02:48<00:00,  4.48batch/s, loss=0.0295, accuracy=0.9906]
Total Loss: 0.030
Saving best model to best_epoch_model.h5
```

预测图像

BUSI-256数据集

在predict.py中设置需要预测的图像目录,设置保存的预测图像目录,设置正确的模型路径,运行

predict.py

```
model_path = "logs/2025_04_20_20_21_03/best_epoch_model.h5"
images_dir = "data/BUSI-256/images/"
```

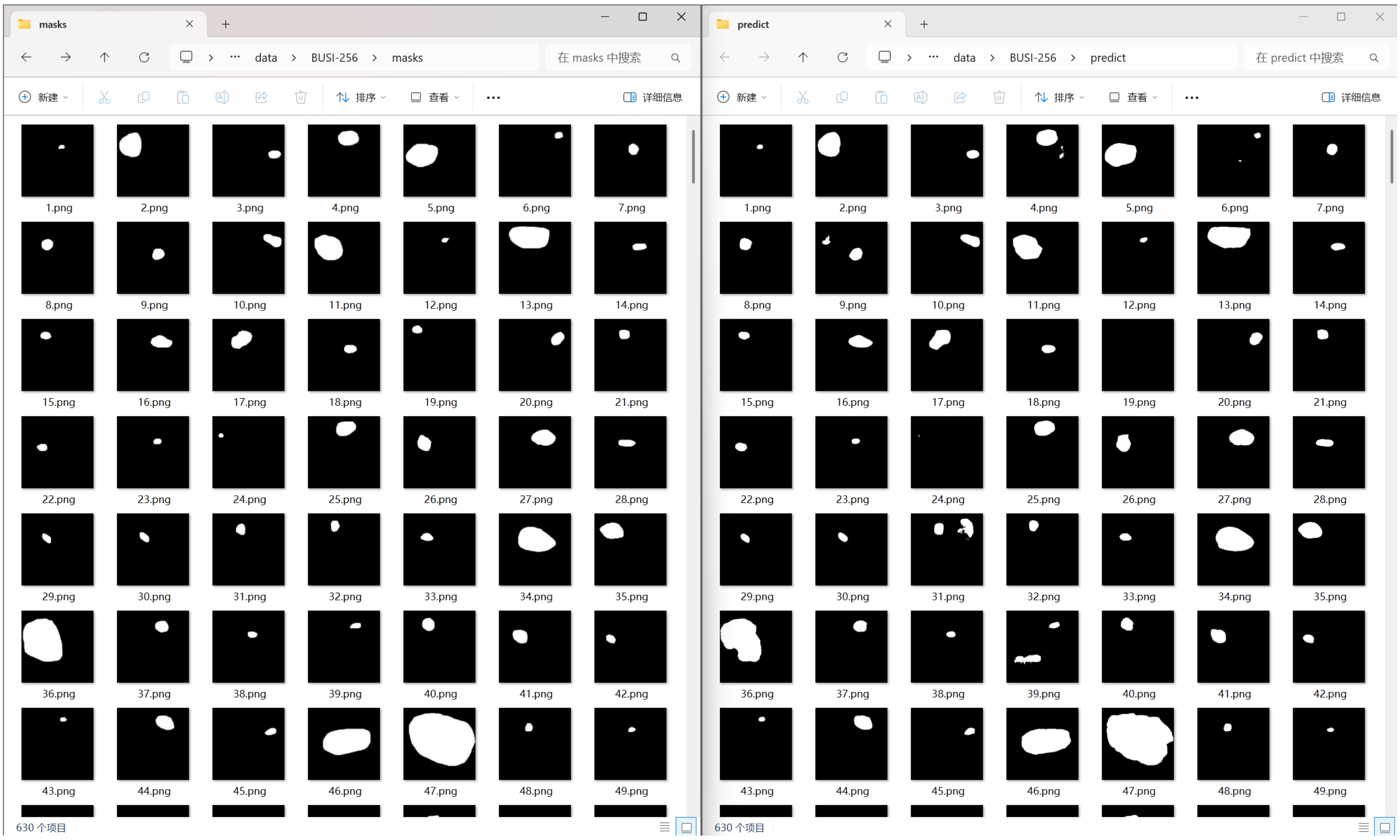
```
output_dir = "data/BUSI-256/predict/"
# model_path = "logs/2025_04_20_22_44_19/best_epoch_model.h5"
# images_dir='data/isic2018/test/images/'
# output_dir='data/isic2018/test/predict/'
```

预测文件已成功保存

代码块

```
D:\VISUAL STUDIO\CODE\PYCHARM\AIOT_DUCK_NET\DATA\BUSI-256
├─2025_04_20_20_21_03
│   └─train_log
├─images
├─masks
└─predict
```

在文件资源管理器中对比查看部分masks和predict文件,可以看到效果较为理想



isic2018数据集

在predict.py中选择isic2018测试集的路径,模型选择isic2018训练出的模型,运行

predict.py

```
# model_path = "logs/2025_04_20_20_21_03/best_epoch_model.h5"
# images_dir = "data/BUSI-256/images/"
```

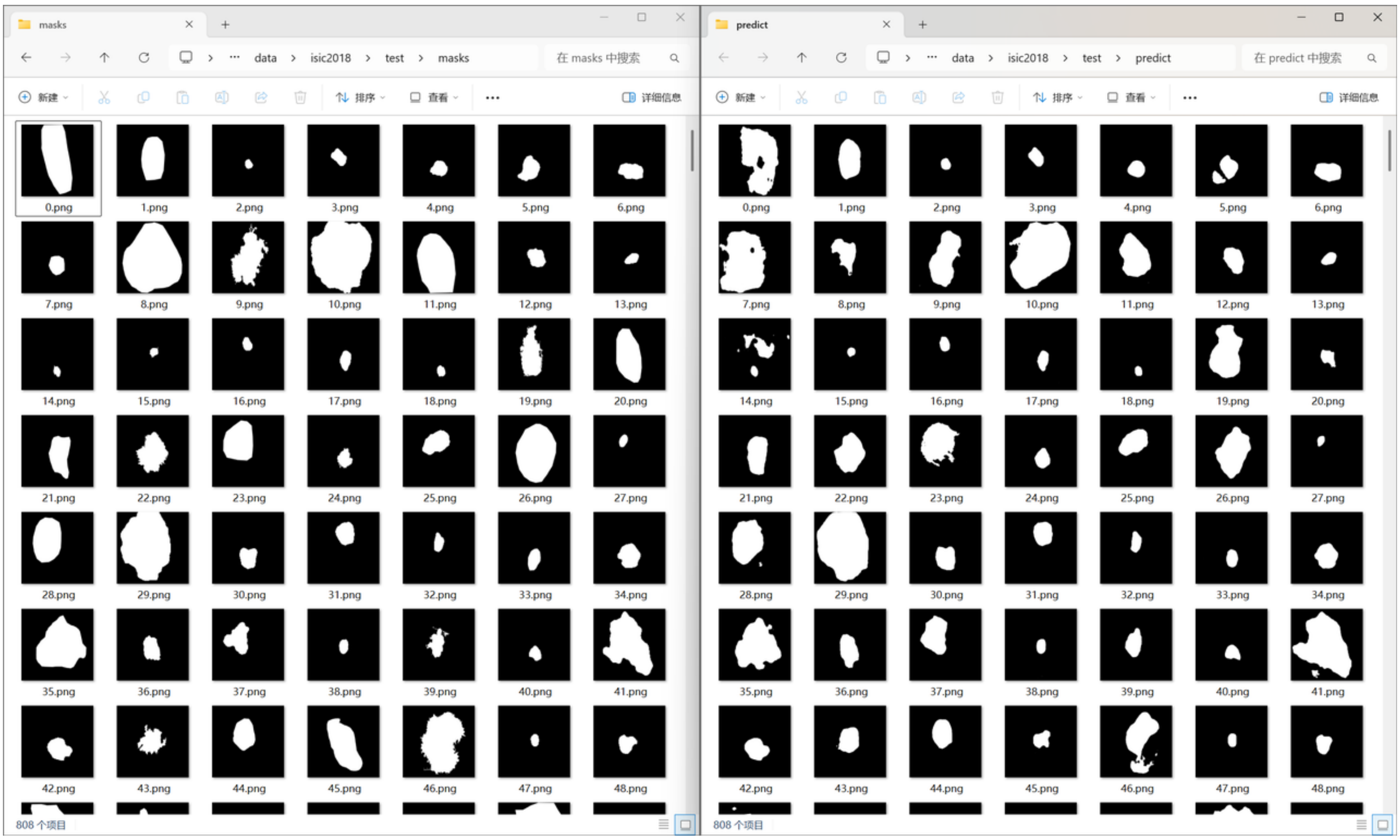
```
# output_dir = "data/BUSI-256/predict/"
model_path = "logs/2025_04_20_22_44_19/best_epoch_model.h5"
images_dir='data/isic2018/test/images/'
output_dir='data/isic2018/test/predict/'
```

预测文件结构

代码块

```
D:\VISUAL STUDIO\CODE\PYCHARM\AIOT_DUCK_NET\DATA\ISIC2018
├─2025_04_20_22_44_19
│   └─train_log
├─test
│   ├─images
│   ├─masks
│   └─predict
└─train
    ├─images
    └─masks
```

在文件资源管理器中打开isic2018测试集的masks和predict的图像对比查看



备注

更新了需求文档requirement.txt