from PyQt5 import QtCore, QtGui, QtWidgets

from PyQt5.QtWidgets import QDialog, QMainWindow, QApplication, QFileDialog, QGraphicsScene, QVBoxLayout, QGraphicsPixmapItem, QGraphicsView, QGraphicsRectItem, QGraphicsPathItem, QLabel, QProgressBar

from PyQt5.QtGui import QPixmap, QImage, QPainter, QColor, QPen, QPainterPath

from PyQt5.QtCore import Qt, QRectF, pyqtSignal, QPointF, QTimer, QThread

import sys

import os

import subprocess

import numpy as np

import matplotlib.pyplot as plt

import threading

from MainWindow import Ui\_MainWindow

import HSIpack

class Main(QMainWindow, Ui\_MainWindow):

settings = QtCore.QSettings("config.ini",

QtCore.QSettings.Format.IniFormat)

######----------------------------------------------------------------------------------------------------######

#####----------------------------------Parameters definition start here------------------------------------#####

####--------------------------------------------------------------------------------------------------------####

# ------------------------------------Tab1------------------------------------

fileNum = 0

fileName = "" # The filename of the HSI

selected\_directory = "" # The abs path of the raw spe file directory

# in Windows: C:\...\... while in linux C:/.../...

# Single file

rawSpeFile\_path = "" # The abs path of the raw spe file

rawHdrFile\_path = "" # The abs path of the raw hdr file

rawsSpeFile\_path = "" # The abs path of the raw spe files

rawsHdrFile\_path = "" # The abs path of the raw hdr files

BRFSpeFile\_path = "" # The abs path of the reference board spe file

BRFHdrFile\_path = "" # The abs path of the reference board hdr file

# Multiple files

rawfile\_paths = "" # The abs path of the raw spe files with batch processing

multiFlag = 0 # multiple files flag: 0 indicates not multiple while 1 indicates multiple

# Data recording for selection rectangular

raw\_scene = None

dealt\_scene = None

selecting = False

selection\_rect = None

selection\_start = None

selection\_end = None

BRF3\_pos\_range = [] # [BRF3%] [[3\_x0,3\_y0],[3\_x1,3\_y1]]

BRF30\_pos\_range = [] # [BRF30%] [[30\_x0,30\_y0],[30\_x1,30\_y1]]

# Data for single Hyperspectra image

raw\_HSI\_info = []

HSI\_lines = 0 # Default length value

HSI\_samples = 0 # Default width value

HSI\_channels = 300 # Default wavelength value

HSI = [[[]]] # 3-D HSI img

HSI\_wavelengths = [] # ranging from apporximately 400nm to 1000nm

# Data for reference board image

BRF\_HSI\_info = []

# select Refboard rect flag; 0 indicates not selected while 1 indicated selected

selectFlag\_1 = 0

selectFlag\_2 = 0

# rbg Image generated by the three bands of HSI

rgbImg = []

# ------------------------------------Tab2------------------------------------

# NDVI\_matrix

NDVI = []

# Threshold value by set at the Tab2

NDVI\_TH\_LOW = -1 # Threshold value of NDVI to seperate the plant from the background

NDVI\_TH\_HIGH = 1 # Threshold value of NDVI to seperate the plant from the background

ampl\_LowTH = 0 # Threshold value of amplititude of the hyperspectra to eliminate

ampl\_HighTH = 4095 # Threshold value of amplititude of the hyperspectra to eliminate

BRFfile\_paths = [] # ["3%BRF\_filename", "30%BRF\_filename"]

# The proportion is initially set as 1

cur\_proportion = 1

# class reflect

reflect = None

k = []

b = []

l1\_rgbimg\_path = "" # level 1 img rgb file path

l2\_rgbimg\_path = "" # level 2 img rgb file path

plant\_mask = [] # bool value mask for the plant: True implies a non-plant pixel while False implies a plant pixel

plantPixNum = 0 # the number of the plant pixels

# ------------------------------------Tab3---------------------------------

Hs\_Para = ""

Ptsths\_Para = ""

Ptsths\_Para\_Model = ""

pro\_data = None

Hs\_Para\_list = ["NDVI","OSAVI", "PSSRa","PSSRb", "PRI","MTVI2","SR", "DVI", "SIPI", "PSRI", "CRI1", "CRI2", "ARI1", "ARI2", "WBI"]

Ptsths\_Para\_list = ["SPAD","A1200", "N", "Ca", "Cb"]

###-------------------------------------------The End line---------------------------------------------------###

######----------------------------------------------------------------------------------------------------######

#####-------------------------------------\_init\_ Function start here---------------------------------------#####

####--------------------------------------------------------------------------------------------------------####

def \_\_init\_\_(self, QMainWindow):

QMainWindow.\_\_init\_\_(self)

Ui\_MainWindow.\_\_init\_\_(self)

self.setupUi(self)

self.previousPage = None

# ------------------------------------Tab1------------------------------------

# Part 1. Raw Data Processing

# Import the BRF HSI files

self.impBRFImgBtn.clicked.connect(self.importBRFImg)

# Mouse box selection for 3% board

self.selectBox3Btn.clicked.connect(lambda: self.selectBox("3"))

# Mouse box selection for 30% board

self.selectBox30Btn.clicked.connect(lambda: self.selectBox("30"))

# Get k and b of the reflectance equation

self.importRftCaliFileBtn.clicked.connect(self.importRftCaliFile)

self.RefCaliBtn.clicked.connect(self.RefCali)

# Import the single raw HSI file

self.impRawBtn.clicked.connect(self.importRaw)

self.impRawBtn.setGeometry(50, 50, 200, 30)

# Read the raw file

self.rgbGeneBtn.clicked.connect(lambda:self.getRgb("Gene"))

# show the raw file

self.rgbViewBtn.clicked.connect(lambda:self.getRgb("View"))

# Save the raw rgb file

self.rgbSaveBtn.clicked.connect(lambda:self.getRgb("Save"))

# Read the raw BRF file

self.BRFRawGeneBtn.clicked.connect(lambda:self.getBRFRgb("Gene"))

# show the raw BRF file

self.BRFRawViewBtn.clicked.connect(lambda:self.getBRFRgb("View"))

# Save the raw BRF file

self.BRFRawSaveBtn.clicked.connect(lambda:self.getBRFRgb("Save"))

# Show the hsi information

self.showHsiInfoBtn.clicked.connect(self.showHsiInfo)

# Draw the hyperspectra curve

self.HSCurveBtn.clicked.connect(self.HSCurveView)

# ------------------------------------Tab2------------------------------------

# Part 2. Data Pre-processing

self.NDVI\_TH\_HIGH = float(self.bgNdviHighDb.currentText())

self.NDVI\_TH\_LOW = float(self.bgNdviLowDb.currentText())

self.ampl\_LowTH = int(self.amplLowThDb.currentText())

self.ampl\_HighTH = int(self.amplHighThDb.currentText())

# Handle Selection Changed

self.bgNdviHighDb.currentIndexChanged.connect(lambda: self.getPreProcessPara(1))

self.bgNdviLowDb.currentIndexChanged.connect(lambda: self.getPreProcessPara(2))

self.amplLowThDb.currentIndexChanged.connect(lambda: self.getPreProcessPara(3))

self.amplHighThDb.currentIndexChanged.connect(lambda: self.getPreProcessPara(4))

# Level 1-2 pre-processing

self.RmBgGeneBtn.clicked.connect(lambda: self.RmBg("Gene"))

self.RmBgViewBtn.clicked.connect(lambda: self.RmBg("View"))

self.RmBgSaveBtn.clicked.connect(lambda: self.RmBg("Save"))

self.RmDbGeneBtn.clicked.connect(lambda: self.RmDb("Gene"))

self.RmDbViewBtn.clicked.connect(lambda: self.RmDb("View"))

self.RmDbSaveBtn.clicked.connect(lambda: self.RmDb("Save"))

self.RefGeneBtn.clicked.connect(lambda: self.getReflect("Gene"))

self.RefViewBtn.clicked.connect(lambda: self.getReflect("View"))

self.RefSaveBtn.clicked.connect(lambda: self.getReflect("Save"))

# Draw the reflectance curve

self.RFCurveBtn.clicked.connect(self.RFCurveView)

# ------------------------------------Tab3------------------------------------

# Get the current text in the drab bar

self.HS\_Para = self.hsParaDb.currentText()

self.Ptsths\_Para = self.ptsthsParaDb.currentText()

self.Ptsths\_Para\_Model = self.ptsthsParaModelDb.currentText()

# Get the changed text in the drab bar

self.hsParaDb.currentIndexChanged.connect(lambda: self.getProcessPara(1))

self.ptsthsParaDb.currentIndexChanged.connect(lambda: self.getProcessPara(2))

self.ptsthsParaModelDb.currentIndexChanged.connect(lambda: self.getProcessPara(3))

# HS Parameters

self.hsParaGeneBtn.clicked.connect(lambda: self.getHsPara("Gene"))

self.hsParaSaveBtn.clicked.connect(lambda: self.getHsPara("Save"))

self.hsParaViewBtn.clicked.connect(lambda: self.getHsPara("View"))

# Plant Phenotypic Parameters

self.ptsthsGeneBtn.clicked.connect(lambda: self.getPtsthsPara("Gene"))

self.ptsthsSaveBtn.clicked.connect(lambda: self.getPtsthsPara("Save"))

self.ptsthsViewBtn.clicked.connect(lambda: self.getPtsthsPara("View"))

self.AvgHsParaGeneBtn.clicked.connect(lambda: self.outputAvgHsParas("Gene",1))

self.AvgPtsthsParaGeneBtn.clicked.connect(lambda: self.outputAvgPtsthsParas("Gene",1))

# One-click processing for multiples file

# Import the multiples raw HSI files

self.impRawsBtn.clicked.connect(self.importRaws)

# Multiple raw datas generating

self.multiGeneBtn.clicked.connect(lambda:self.multiProcess("Gene"))

self.multiViewBtn.clicked.connect(lambda:self.multiProcess("View"))

######----------------------------------------------------------------------------------------------------######

#####-------------------------------------Helper Function start here---------------------------------------#####

####--------------------------------------------------------------------------------------------------------####

# -------------------------------------Tab1-------------------------------------

def importRaw(self):

file\_dialog = QFileDialog()

selected\_file, \_ = file\_dialog.getOpenFileName(QMainWindow(), '选择文件', '', '.spe(\*.spe\*)')

if selected\_file:

self.rawSpeFile\_path = selected\_file

self.rawSpeFile\_path = self.rawSpeFile\_path.replace("\\","/")

self.rawHSIPathlineEdit.setText(self.rawSpeFile\_path)

self.rawHdrFile\_path = self.rawSpeFile\_path.replace(".spe",".hdr")

self.fileNum = 1

# import the image that contains the Reference Board

def importBRFImg(self):

selected\_file, \_ = QFileDialog.getOpenFileName(QMainWindow(), '选择文件', '', '.spe(\*.spe\*)')

if selected\_file:

self.BRFSpeFile\_path = selected\_file

self.BRFSpeFile\_path = self.BRFSpeFile\_path.replace("\\","/")

self.BRFPathlineEdit.setText(self.BRFSpeFile\_path)

self.BRFHdrFile\_path = self.BRFSpeFile\_path.replace(".spe",".hdr")

if self.fileNum == 0: # Allow to upload multiple first

self.fileNum = 1

# import the amplititude along diferent wavelengths of 3% and 30% BRF

def importRftCaliFile(self):

file\_dialog = QFileDialog()

self.selected\_directory = file\_dialog.getExistingDirectory(self, "选择文件夹")

if self.selected\_directory:

BRFfile\_names = os.listdir(self.selected\_directory)

BRFfile\_names = [item.replace("\\","/") for item in BRFfile\_names]

self.selected\_directory = self.selected\_directory.replace("\\","/")

self.BRFCaliPathlineEdit.setText(self.selected\_directory)

self.BRFfile\_paths = [self.selected\_directory + "/" + item for item in BRFfile\_names]

def RefCali(self):

self.reflect = HSIpack.gr.Reflectance(self.HSI\_info, self.cur\_proportion, [self.BRF3\_pos\_range, self.BRF30\_pos\_range], self.BRFfile\_paths, [], [], self.plant\_mask, self.fileName)

# Get the k and b

self.k, self.b = self.reflect.getReflectEquation()

# Unlock the view and Save function

QtWidgets.QMessageBox.about(self, "", "反射板校准已就绪")

if self.fileNum > 1:

self.multiGeneBtn.setEnabled(True)

def getRgb(self, function):

match function:

case "Gene":

self.HSI\_info = HSIpack.rd.ReadData(self.rawHdrFile\_path,self.rawSpeFile\_path, 1)

self.HSI\_lines = self.HSI\_info[0]

self.HSI\_channels = self.HSI\_info[1]

self.HSI\_samples = self.HSI\_info[2]

self.HSI = self.HSI\_info[3]

self.HSI\_wavelengths = self.HSI\_info[4]

self.plant\_mask = np.zeros((self.HSI\_lines, self.HSI\_samples), dtype=bool)

self.cur\_proportion = 1

index1 = self.rawSpeFile\_path.rfind("/")

index2 = self.rawSpeFile\_path.find(".", index1)

self.fileName = self.rawSpeFile\_path[index1+1: index2]

# Week amplititude detection

if np.max(self.HSI) < 1000:

QtWidgets.QMessageBox.warning(self, "", "高光谱原始数据亮度值过低！请重新导入")

return

# Unlock the view and Save function

if self.fileNum == 1:

self.rgbSaveBtn.setEnabled(True)

self.rgbViewBtn.setEnabled(True)

self.showHsiInfoBtn.setEnabled(True)

QtWidgets.QMessageBox.about(self, "", "高光谱原始数据处理成功")

elif self.fileNum > 1:

self.multiViewBtn.setEnabled(True)

case "Save":

if self.rawSpeFile\_path != "":

if not os.path.exists("Outputs/figures/"+ self.fileName):

os.makedirs("Outputs/figures/"+ self.fileName)

if not os.path.exists("Outputs/results/" + self.fileName):

os.makedirs("Outputs/results/" + self.fileName)

self.rgbImg = HSIpack.rd.drawImg(self.HSI\_info)

self.rgbImg.save("Outputs/figures/" + self.fileName + "/raw.jpg")

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "高光谱可视化数据保存成功")

case "View":

if self.rawSpeFile\_path != "":

self.rawjpgFile\_path = "Outputs/figures/" + self.fileName + "/raw.jpg"

frame = QImage(self.rawjpgFile\_path)

pix = QPixmap.fromImage(frame)

item = QGraphicsPixmapItem(pix)

# the rgb scene in Tab1

self.raw\_scene = QGraphicsScene()

self.raw\_scene.addItem(item)

self.hsiRawView.setScene(self.raw\_scene)

# Make the graph self-adaptive to the canvas

self.hsiRawView.fitInView(self.raw\_scene.sceneRect(), Qt.KeepAspectRatio)

self.HSCurveBtn.setEnabled(True)

def getBRFRgb(self, function):

match function:

case "Gene":

self.HSI\_info = HSIpack.rd.ReadData(self.BRFHdrFile\_path,self.BRFSpeFile\_path, 1)

self.HSI\_lines = self.HSI\_info[0]

self.HSI\_channels = self.HSI\_info[1]

self.HSI\_samples = self.HSI\_info[2]

self.HSI = self.HSI\_info[3]

self.HSI\_wavelengths= self.HSI\_info[4]

self.plant\_mask = np.zeros((self.HSI\_lines, self.HSI\_samples), dtype=bool)

self.cur\_proportion = 1

index1 = self.rawSpeFile\_path.rfind("/")

index2 = self.rawSpeFile\_path.find(".", index1)

self.fileName = self.rawSpeFile\_path[index1+1: index2]

# Unlock the view and Save function

self.BRFRawViewBtn.setEnabled(True)

self.BRFRawSaveBtn.setEnabled(True)

self.showHsiInfoBtn.setEnabled(True)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "高光谱反射板处理成功")

case "Save":

if self.BRFSpeFile\_path != "":

self.rgbImg = HSIpack.rd.drawImg(self.HSI\_info)

# Handle multiple files saving

if not os.path.exists("Outputs/figures/BRF"):

os.makedirs("Outputs/figures/BRF")

if not os.path.exists("Outputs/results/BRF"):

os.makedirs("Outputs/results/BRF")

self.rgbImg.save("Outputs/figures/BRF/rawBRF.jpg")

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "高光谱反射板可视化保存成功")

case "View":

if self.BRFSpeFile\_path != "":

self.rawjpgFile\_path = "Outputs/figures/BRF/rawBRF.jpg"

frame = QImage(self.rawjpgFile\_path)

pix = QPixmap.fromImage(frame)

item = QGraphicsPixmapItem(pix)

# the rgb scene in Tab1

self.raw\_scene = QGraphicsScene()

self.raw\_scene.addItem(item)

self.hsiRawView.setScene(self.raw\_scene)

# Unlock

self.selectBox3Btn.setEnabled(True)

self.selectBox30Btn.setEnabled(True)

self.HSCurveBtn.setEnabled(True)

# Need to re select

def selectBox(self, brf\_flag):

if brf\_flag == "3" and self.selectFlag\_1 == 0:

self.view = hsiRawView(self.raw\_scene, brf\_flag)

self.selectFlag\_1 = 1

if brf\_flag == "30" and self.selectFlag\_2 == 0:

self.view = hsiRawView(self.raw\_scene, brf\_flag)

self.selectFlag\_2 = 1

#self.setCentralWidget(self.view)

self.view.show()

self.view.resize(600, 800)

#self.view.startSelection()

def showHsiInfo(self):

self.lenShowBtn.setText(str(self.HSI\_lines)+" pix")

self.widthShowBtn.setText(str(self.HSI\_samples)+" pix")

self.wlShowBtn.setText(str(self.HSI\_channels)+" bands")

self.wavesLayout = QVBoxLayout(self.wavesWidget)

text = "图像具体波段"

label = QLabel(text)

label.setStyleSheet("border: none; font: 12pt 'Agency FB';")

self.wavesLayout.addWidget(label)

for i in range(self.HSI\_channels):

text = "band " + str(i+1) + "------" + self.HSI\_wavelengths[i] + " nm"

label = QLabel(text)

label.setStyleSheet("border: none; font: 12pt 'Times New Roman';")

self.wavesLayout.addWidget(label)

self.WaveScrollArea.setWidgetResizable(True)

# ------------------------------------Tab2------------------------------------

def getPreProcessPara(self, index):

combo\_box = self.sender()

match index:

case 1:

self.NDVI\_TH\_HIGH = float(combo\_box.currentText())

case 2:

self.NDVI\_TH\_LOW = float(combo\_box.currentText())

case 3:

self.ampl\_LowTH = int(combo\_box.currentText())

case 4:

self.ampl\_HighTH = int(combo\_box.currentText()

# Remove the background by NDVI

def RmBg(self, function):

match function:

case "Gene":

pre\_data = HSIpack.pre.Preprocess(self.HSI\_info, self.NDVI\_TH\_LOW, self.NDVI\_TH\_HIGH, self.ampl\_LowTH, self.ampl\_HighTH, self.cur\_proportion, self.plant\_mask)

level1 = pre\_data.getLevel1()

self.HSI\_info = level1[0]

self.cur\_proportion = level1[2]

self.NDVI = level1[3]

level1\_mask = level1[4]

self.plant\_mask = self.plant\_mask | level1\_mask

self.plantPixNum = np.count\_nonzero(~self.plant\_mask)

# Unlock the view and Save function

self.RmBgViewBtn.setEnabled(True)

self.RmBgSaveBtn.setEnabled(True)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "去除非植物部分背景成功")

case "Save":

l1\_rgbimg = HSIpack.rd.drawImg(self.HSI\_info)

self.l1\_rgbimg\_path = "Outputs/figures/"+ self.fileName + "/preprocess/level1.jpg"

l1\_rgbimg.save(self.l1\_rgbimg\_path)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "可视化保存成功")

case "View":

frame = QImage(self.l1\_rgbimg\_path)

pix = QPixmap.fromImage(frame)

item = QGraphicsPixmapItem(pix)

# the rgb scene in Tab1

self.dealt\_scene = QGraphicsScene()

self.dealt\_scene.addItem(item)

self.hsidealtView.setScene(self.dealt\_scene)

# Make the graph self-adaptive to the canvas

#self.hsidealtView.fitInView(self.scene.sceneRect(), Qt.KeepAspectRatio)

'''

fig, ax = plt.subplots(figsize=(6, 8))

im = ax.imshow(self.NDVI, cmap='gray',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on NDVI", y=1.05)

fig.colorbar(im)

plt.show()

'''

# Remove the too bright and to dark img

def RmDb(self, function):

# To remove the shadow and the bright of the plot

match function:

case "Gene":

pre\_data = HSIpack.pre.Preprocess(self.HSI\_info, self.NDVI\_TH\_LOW, self.NDVI\_TH\_HIGH, self.ampl\_LowTH, self.ampl\_HighTH, self.cur\_proportion, self.plant\_mask)

level2 = pre\_data.getLevel2()

self.HSI\_info = level2[0]

self.cur\_proportion = level2[3]

self.plantPixNum = self.cur\_proportion \* (self.HSI\_lines \* self.HSI\_samples)

level2\_mask = level2[2]

self.plant\_mask = self.plant\_mask | level2\_mask

# Unlock the view and Save function

self.RmDbViewBtn.setEnabled(True)

self.RmDbSaveBtn.setEnabled(True)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "去除过暗过曝成功")

case "Save":

if not os.path.exists("Outputs/figures/"+ self.fileName + "/preprocess"):

os.makedirs("Outputs/figures/"+ self.fileName + "/preprocess")

l2\_rgbImg = HSIpack.rd.drawImg(self.HSI\_info)

self.l2\_rgbimg\_path = "Outputs/figures/"+ self.fileName + "/preprocess/level2.jpg"

l2\_rgbImg.save(self.l2\_rgbimg\_path)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "可视化保存成功")

case "View":

frame = QImage(self.l2\_rgbimg\_path)

pix = QPixmap.fromImage(frame)

item = QGraphicsPixmapItem(pix)

# the rgb scene in Tab1

self.dealt\_scene = QGraphicsScene()

self.dealt\_scene.addItem(item)

self.hsidealtView.setScene(self.dealt\_scene)

# Calculate the reflectance

def getReflect(self, function):

match function:

case "Gene":

self.reflect = HSIpack.gr.Reflectance(self.HSI\_info, self.cur\_proportion, [self.BRF3\_pos\_range, self.BRF30\_pos\_range], self.BRFfile\_paths, self.k, self.b, self.plant\_mask, self.fileName)

self.reflect.getReflect()

# Unlock the view and Save function

self.RefViewBtn.setEnabled(True)

self.RefSaveBtn.setEnabled(True)

self.RFCurveBtn.setEnabled(True)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "反射率校准处理成功")

case "Save":

self.reflect.visualizeReflect(1)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "植物平均反射率曲线保存成功")

case "View":

self.reflect.visualizeReflect(0)

def HSCurveView(self):

self.view = HSCurve(self.raw\_scene)

self.view.show()

def RFCurveView(self):

self.view = RFCurve(self.raw\_scene)

self.view.show()

# ----------------------------Tab3-----------------------------

def getProcessPara(self, index):

combo\_box = self.sender()

match index:

case 1:

self.Hs\_Para = combo\_box.currentText()

case 2:

self.Ptsths\_Para = combo\_box.currentText()

case 3:

self.Ptsths\_Para\_Model = combo\_box.currentText()

def getHsPara(self, function):

match function:

case "Gene":

# change the non-plant pixel reflectance to 0 at all bands

'''

self.reflect.ReflectMatrix = np.transpose(self.reflect.ReflectMatrix, (0, 2, 1)) # exChange the second and the third dimension

self.reflect.ReflectMatrix[~self.plant\_mask,:] = 0

self.reflect.ReflectMatrix = np.transpose(self.reflect.ReflectMatrix, (0, 2, 1))

'''

reflect\_info = [self.HSI\_lines, self.HSI\_channels, self.HSI\_samples, self.reflect.ReflectMatrix, self.HSI\_wavelengths, self.cur\_proportion]

self.HS\_Para = self.hsParaDb.currentText()

self.pro\_data = HSIpack.pro.Process(reflect\_info, self.Hs\_Para, self.Ptsths\_Para, self.Ptsths\_Para\_Model, self.plant\_mask, self.fileName)

self.pro\_data.calcHsParas()

# Unlock the view and Save function

self.hsParaSaveBtn.setEnabled(True)

self.hsParaViewBtn.setEnabled(True)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "光谱指数计算成功")

case "Save":

if not os.path.exists("Outputs/figures/" + self.fileName + "/process"):

os.makedirs("Outputs/figures/" + self.fileName + "/process")

self.pro\_data.draw\_pseudoColorImg("Save", 1)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "光谱指数计算结果保存成功")

case "View":

self.pro\_data.draw\_pseudoColorImg("View", 1)

def getPtsthsPara(self, function):

match function:

case "Gene":

reflect\_info = [self.HSI\_lines, self.HSI\_channels, self.HSI\_samples, self.reflect.ReflectMatrix, self.HSI\_wavelengths, self.cur\_proportion]

self.pro\_data = HSIpack.pro.Process(reflect\_info, self.Hs\_Para, self.Ptsths\_Para, self.Ptsths\_Para\_Model, self.plant\_mask, self.fileName)

for i in range(self.plantPixNum):

self.pro\_data.CalcPhenotypeParas(i)

print("one figure processed successfully")

self.ptsthsSaveBtn.setEnabled(True)

self.ptsthsViewBtn.setEnabled(True)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "光合表型参数计算成功")

'''

self.progress\_bar = QProgressBar(self)

self.progress\_bar.setGeometry(30, 40, 200, 25)

self.progress\_value = 0

self.progress\_bar.setValue(self.progress\_value)

self.timer = QTimer()

self.timer.timeout.connect(self.calcLoop)

self.timer.start(1000) # 每100毫秒更新一次进度条

'''

case "Save":

if not os.path.exists("Outputs/figures/" + self.fileName + "/process"):

os.makedirs("Outputs/figures/" + self.fileName + "/process")

self.pro\_data.draw\_pseudoColorImg("Save", 2)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "光合表型参数计算结果保存成功")

case "View":

self.pro\_data.draw\_pseudoColorImg("View", 2)

'''

def calcLoop(self):

count = 0

for i in range(self.plantPixNum):

self.pro\_data.CalcPhenotypeParas(i)

count += 1

if count == self.HSI\_samples:

print("count + 1")

self.progress\_value += 1

count = 0

self.progress\_bar.setValue(self.progress\_value)

if (self.progress\_value >=self.HSI\_lines):

self.timer.stop()

print(len(self.pro\_data.y\_pre))

# Unlock the view and Save function

self.ptsthsViewBtn.setEnabled(True)

self.ptsthsSaveBtn.setEnabled(True)

QtWidgets.QMessageBox.about(self, "", "光合表型参数计算成功")

#QApplication.processEvents() # 更新 UI，防止界面冻结

def closeEvent(self, event):

# 在窗口关闭时停止循环

#self.progress\_bar.setValue(self.HSI\_lines)

self.timer.stop()

event.accept()

'''

def outputAvgHsParas(self, function, idx):

match function:

case "Gene":

reflect\_info = [self.HSI\_lines, self.HSI\_channels, self.HSI\_samples, self.reflect.ReflectMatrix, self.HSI\_wavelengths, self.cur\_proportion]

self.pro\_data = HSIpack.pro.Process(reflect\_info, self.Hs\_Para, self.Ptsths\_Para, self.Ptsths\_Para\_Model, self.plant\_mask, self.fileName)

self.pro\_data.exportHsParas("Outputs/results/AvgHsPara.csv", idx)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "一键计算结果保存成功")

def outputAvgPtsthsParas(self, function, idx):

match function:

case "Gene":

reflect\_info = [self.HSI\_lines, self.HSI\_channels, self.HSI\_samples, self.reflect.ReflectMatrix, self.HSI\_wavelengths, self.cur\_proportion]

self.pro\_data = HSIpack.pro.Process(reflect\_info, self.Hs\_Para, self.Ptsths\_Para, self.Ptsths\_Para\_Model, self.plant\_mask, self.fileName)

self.pro\_data.exportPhenotypeParas("Outputs/results/AvgPtsthsPara.csv", idx)

if self.fileNum == 1:

QtWidgets.QMessageBox.about(self, "", "一键计算结果保存成功")

# one-cliked processing for multiple files

def importRaws(self):

rawfile\_names = []

file\_dialog = QFileDialog()

self.selected\_directory = file\_dialog.getExistingDirectory(self, "选择文件夹")

if self.selected\_directory:

rawfoleder\_names = os.listdir(self.selected\_directory)

for i in range(len(rawfoleder\_names)):

#rawfile\_names.append(item.replace("\\","/") for item in rawfile\_names if item.endswith(".spe")) # only show the .spe file

self.selected\_directory = self.selected\_directory.replace("\\","/")

rawfile\_names.append(rawfoleder\_names[i] + ".spe")

self.rawfile\_paths = [self.selected\_directory + "/" + rawfoleder\_names[i] + "/" + rawfoleder\_names[i] + ".spe" for i in range(len(rawfoleder\_names))]

self.fileNum = len(self.rawfile\_paths)

# Bug remain here to do

self.rawsLayout = QVBoxLayout(self.rawFilesWidget)

self.rawsLayout.setContentsMargins(0, 0, 0, 0) # Remove any margins

for i in range(len(self.rawfile\_paths)):

text = rawfile\_names[i]

label = QLabel(text)

label.setStyleSheet("border: none; font: 12pt 'Times New Roman';")

label.setCursor(QtCore.Qt.PointingHandCursor) # Set cursor to hand when hovering over the label

label.mousePressEvent = lambda event, label=label: self.labelClicked(label) # Connect the label's click event to the function

self.rawsLayout.addWidget(label)

self.rawsLayout.addStretch() # Add stretchable space at the end

self.RawFilesScrollArea.setWidgetResizable(True)

self.multiFlag = 1

def labelClicked(self, label):

self.raw = label.text()

self.rawSpeFile\_path = self.selected\_directory + "/" + self.raw

self.rawHdrFile\_path = self.rawSpeFile\_path.replace(".spe",".hdr")

# Refresh the widget background and Set the label background to the selected color

for child in self.rawFilesWidget.children():

if isinstance(child, QLabel):

child.setStyleSheet("background-color: None; font: 12pt 'Agency FB'; border: None;")

label.setStyleSheet("background-color: lightblue")

#self.rawFilesWidget.setStyleSheet("background-color: transparent;")

#Multiple files processing

def multiProcess(self, function):

match function:

# Multiple files result generation

case "Gene":

threadNum = len(self.rawfile\_paths)

# Show the progress dialog

'''

progress\_dialog = QtWidgets.QProgressDialog(self)

progress\_dialog.setModal(True)

progress\_dialog.setWindowTitle("Processing")

progress\_dialog.setLabelText("Processing files...")

progress\_dialog.setMinimum(0)

progress\_dialog.setMaximum(threadNum)

progress\_dialog.show()

'''

for i in range(threadNum):

# get the raw data

self.rawSpeFile\_path = self.rawfile\_paths[i]

self.rawSpeFile\_path = self.rawSpeFile\_path.replace("\\","/")

self.rawHSIPathlineEdit.setText(self.rawSpeFile\_path)

self.rawHdrFile\_path = self.rawSpeFile\_path.replace(".spe",".hdr")

# get the file name

self.getRgb("Gene")

self.getRgb("Save")

# get the preprocessed data

self.RmDb("Gene")

self.RmBg("Gene")

self.getReflect("Gene")

self.getReflect("Save")

# get the processed data

# Output the figure

for j in range(len(self.Hs\_Para\_list)):

self.Hs\_Para = self.Hs\_Para\_list[j]

self.getHsPara("Gene")

self.getHsPara("Save")

# Bugs remains here

'''

for j in range(len(self.Ptsths\_Para\_list)):

self.Ptsths\_Para = self.Ptsths\_Para\_list[j]

self.getPtsthsPara("Gene")

self.getPtsthsPara("Save")

'''

# Output the result

self.outputAvgPtsthsParas("Gene", i+1) # start from i = 1

self.outputAvgHsParas("Gene", i+1) # start from i = 1

# Data re-initiailization

self.cur\_proportion = 1

# Update the progress

#progress\_dialog.setValue(i + 1)

#QtWidgets.QApplication.processEvents() # Allow GUI updates

#progress\_dialog.close()

QtWidgets.QMessageBox.about(self, "", "一键完成")

# To show the multiple files result

case "View":

return

# ----------------------------Tab4-----------------------------

class hsiRawView(QGraphicsView):

def \_\_init\_\_(self, scene, brf\_flag):

super().\_\_init\_\_(scene)

self.setDragMode(QGraphicsView.RubberBandDrag)

self.selection\_rect = None

self.selecting = False

self.BRF\_flag = brf\_flag

def startSelection(self):

self.selecting = True

self.selection\_rect = QGraphicsRectItem()

if self.BRF\_flag == "3":

self.selection\_rect.setPen(Qt.blue)

if self.BRF\_flag == "30":

self.selection\_rect.setPen(Qt.red)

self.scene().addItem(self.selection\_rect)

def resetSelection(self):

if self.selection\_rect is not None:

print("YES")

self.scene().removeItem(self.selection\_rect)

self.selection\_rect = None

self.selecting = False

self.update()

def stopSelection(self):

if self.selection\_rect is not None:

selected\_items = self.scene().items(self.selection\_rect.rect(), Qt.IntersectsItemShape)

# print x and y

rect = self.selection\_rect.rect()

if self.BRF\_flag == "3":

BRF3\_x0 = int(rect.x())

BRF3\_y0 = int(rect.y())

BRF3\_x1 = int(BRF3\_x0 + rect.width())

BRF3\_y1 = int(BRF3\_y0 + rect.height())

md.BRF3\_pos\_range = [[BRF3\_x0,BRF3\_y0],[BRF3\_x1, BRF3\_y1]]

#print(md.BRF3\_pos\_range)

elif self.BRF\_flag == "30":

BRF30\_x0 = int(rect.x())

BRF30\_y0 = int(rect.y())

BRF30\_x1 = int(BRF30\_x0 + rect.width())

BRF30\_y1 = int(BRF30\_y0 + rect.height())

md.BRF30\_pos\_range = [[BRF30\_x0,BRF30\_y0],[BRF30\_x1, BRF30\_y1]]

#print(md.BRF30\_pos\_range)

#self.scene().removeItem(self.selection\_rect)

self.selection\_rect = None

self.selecting = False

def mousePressEvent(self, event):

if event.button() == Qt.LeftButton:

self.resetSelection()

self.startSelection()

pos\_in\_view = event.pos()

pos\_in\_scene = self.mapToScene(pos\_in\_view)

self.selection\_rect.setRect(QRectF(pos\_in\_scene, pos\_in\_scene))

self.scene().addItem(self.selection\_rect)

def mouseMoveEvent(self, event):

if self.selecting and self.selection\_rect is not None:

pos\_in\_view = event.pos()

pos\_in\_scene = self.mapToScene(pos\_in\_view)

rect = QRectF(self.selection\_rect.rect().topLeft(), pos\_in\_scene)

self.selection\_rect.setRect(rect.normalized())

def mouseReleaseEvent(self, event):

if event.button() == Qt.LeftButton and self.selecting:

self.stopSelection()

class HSCurve(QGraphicsView):

def \_\_init\_\_(self, scene):

super().\_\_init\_\_(scene)

self.setMouseTracking(True) # Turn on the mouse track

self.cursor\_pos = QPointF(0, 0)

self.crosshair\_item = QGraphicsPathItem()

self.crosshair\_item.setPen(QPen(Qt.blue))

self.scene().addItem(self.crosshair\_item)

def resizeEvent(self, event):

super().resizeEvent(event)

self.updateCrosshair()

def mouseMoveEvent(self, event):

self.cursor\_pos = self.mapToScene(event.pos())

#print(self.cursor\_pos)

self.updateCrosshair()

def updateCrosshair(self):

#view\_width = self.viewport().width()

#view\_height = self.viewport().height()

view\_width = md.HSI\_samples

view\_height = md.HSI\_lines

x = 0

y = 0

path = QPainterPath()

if self.cursor\_pos.x()>=0 and self.cursor\_pos.x()<=view\_width and self.cursor\_pos.y()>=0 and self.cursor\_pos.y()<=view\_height:

# Paint the cross cursor

path.moveTo(self.cursor\_pos.x(), 0)

path.lineTo(self.cursor\_pos.x(), view\_height)

path.moveTo(0, self.cursor\_pos.y())

path.lineTo(view\_width, self.cursor\_pos.y())

self.crosshair\_item.setPath(path)

def mousePressEvent(self, event):

if event.button() == Qt.LeftButton:

x = np.array(md.HSI\_wavelengths)

y = np.array(md.HSI[int(self.cursor\_pos.y()),:,int(self.cursor\_pos.x())])

plt.xlabel("Wavelength(nm)")

plt.ylabel("Hyperspectral Luminance")

plt.plot(x, y, c='g', label='Curve\_poly\_Fit')

plt.title("The Reflectance curve")

plt.show()

# delete the cross path after close the event

def closeEvent(self, event):

self.crosshair\_item.setPath(QPainterPath())

class RFCurve(QGraphicsView):

def \_\_init\_\_(self, scene):

super().\_\_init\_\_(scene)

self.setMouseTracking(True) # Turn on the mouse track

self.cursor\_pos = QPointF(0, 0)

self.crosshair\_item = QGraphicsPathItem()

self.crosshair\_item.setPen(QPen(Qt.blue))

self.scene().addItem(self.crosshair\_item)

def resizeEvent(self, event):

super().resizeEvent(event)

self.updateCrosshair()

def mouseMoveEvent(self, event):

self.cursor\_pos = self.mapToScene(event.pos())

#print(self.cursor\_pos)

self.updateCrosshair()

def updateCrosshair(self):

#view\_width = self.viewport().width()

#view\_height = self.viewport().height()

view\_width = md.HSI\_samples

view\_height = md.HSI\_lines

x = 0

y = 0

path = QPainterPath()

if self.cursor\_pos.x()>=0 and self.cursor\_pos.x()<=view\_width and self.cursor\_pos.y()>=0 and self.cursor\_pos.y()<=view\_height:

# Paint the cross cursor

path.moveTo(self.cursor\_pos.x(), 0)

path.lineTo(self.cursor\_pos.x(), view\_height)

path.moveTo(0, self.cursor\_pos.y())

path.lineTo(view\_width, self.cursor\_pos.y())

self.crosshair\_item.setPath(path)

def mousePressEvent(self, event):

if event.button() == Qt.LeftButton:

# Just show the HS with wavelength within 400nm - 990nm

x = np.array(md.HSI\_wavelengths[2:-22])

y = np.array(md.reflect.ReflectMatrix[int(self.cursor\_pos.y()),2:-22,int(self.cursor\_pos.x())])

plt.xlabel("Wavelength(nm)")

plt.ylabel("Reflectance")

plt.plot(x,y,c='lightcoral',label='Curve\_poly\_Fit')

plt.title("The Reflectance curve of the cross cursor point")

plt.show()

# delete the cross path after close the event

def closeEvent(self, event):

self.crosshair\_item.setPath(QPainterPath())

if \_\_name\_\_ == "\_\_main\_\_":

app = QApplication(sys.argv)

md = Main(QMainWindow)

md.show()

sys.exit(app.exec\_())

import numpy as np

import matplotlib.pyplot as plt

import csv

from scipy.interpolate import interp1d

class Reflectance:

HSI\_info = []

cur\_proportion = 1

BRF\_positionRange = [] # position range of the refer board

BRF\_files = "" # path location fo the 3% and 30% Refer board Cali files

AVG\_reflect = [] # 1-D average reflectance for the whole plot in different bands

ReflectMatrix = [] # 3-D reflectance matrix for the whole plot

k = [] # i.e. 300 k for 300 bands

b = [] # i.e. 300 b for 300 bands

plantMask = []

BRFfilename = "BRF"

def \_\_init\_\_(self, HSI\_info, cur\_proportion, BRF\_positionRange, brf\_files, k, b, plant\_mask, filename):

self.HSI\_info = HSI\_info

#print(self.HSI\_info)

self.cur\_proportion = cur\_proportion

self.BRF\_positionRange = BRF\_positionRange

self.BRF\_files = brf\_files

self.k = k

self.b = b

self.plantMask = plant\_mask

self.fileName = filename

#print(self.BRF\_positionRange)

def getReferAmplititudes(self, BRF\_flag):

channels = self.HSI\_info[1]

HSI = self.HSI\_info[3]

#positionRange i.e. (860,83),(890,120) for 3% Ref Board

Amplititudes = []

if BRF\_flag == "3":

RefHSI = HSI[self.BRF\_positionRange[0][0][1]:self.BRF\_positionRange[0][1][1],:,self.BRF\_positionRange[0][0][0]:self.BRF\_positionRange[0][1][0]]

elif BRF\_flag == "30":

RefHSI = HSI[self.BRF\_positionRange[1][0][1]:self.BRF\_positionRange[1][1][1],:,self.BRF\_positionRange[1][0][0]:self.BRF\_positionRange[1][1][0]]

for i in range(channels):

Amplititudes.append(np.array(RefHSI[:,i,:]).mean())

return Amplititudes

def writeRef(self, filename, wavelengths, RefAmplititudes1, RefAmplititudes2):

with open(filename, "w", newline="") as f:

writer = csv.writer(f)

writer.writerow(wavelengths)

writer.writerow(RefAmplititudes1)

writer.writerow(RefAmplititudes2)

def readRef(self, filename):

with open(filename,"r") as f:

raw = f.readlines()

str = ""

str = ','.join(raw)

str = str.replace("\n","").replace("波长,反射率(%)","").replace("波长","").replace("反射率%","").replace(" ","")

str = str.split(",")

result\_list = [x for x in str if x != '']

#print("Now is",result\_list)

waves = [float(x) for x in result\_list[0::2]]

reflect = [float(x) for x in result\_list[1::2]]

return waves,reflect

def mapRef(self, inputfile, outputfile, lineNum):

X\_map, Y\_map, orig\_X = [], [], []

Read\_info = self.readRef(outputfile)

waves = Read\_info[0]

k = 0

with open(inputfile, "r") as f1:

contents = f1.readlines()

RefWaves = [float(items) for items in contents[0].split(",")]

RefReflect = [float(items) for items in contents[lineNum].split(",")]

#print (contents[0])

#print("------\n",contents[1])

channels = self.HSI\_info[1]

num = len(waves)

#print(num)

#print("RefWaves is",RefWaves)

#print("waves is", waves)

min\_diff = float('inf')

min\_index = None

'''

for k in range(num):

for i in range(channels):

diff = abs(RefWaves[i] - waves[k])

if diff < min\_diff:

min\_diff = diff

min\_index = i

if min\_index is not None:

X\_map.append(RefReflect[min\_index])

Y\_map.append(Read\_info[1][k]/100)

'''

for i in range(channels):

for k in range(num):

if abs(RefWaves[i] - waves[k]) < 1:

X\_map.append(RefReflect[i])

#print(Read\_info[1])

Y\_map.append(Read\_info[1][k]/100)

orig\_X = RefReflect

return X\_map, Y\_map, orig\_X

def getReflectEquation(self):

self.getRefBoard()

channels = self.HSI\_info[1]

RefAmplititudes\_file ="Outputs/results/BRF/RefAmplititudes.csv"

#print("------------3Ref----------")

if self.BRF\_files[0][-6:] == "30.csv":

BRF\_30 = self.BRF\_files[0]

BRF\_3 = self.BRF\_files[1]

else:

BRF\_30 = self.BRF\_files[1]

BRF\_3 = self.BRF\_files[0]

refSample1= self.mapRef(RefAmplititudes\_file, BRF\_3, 1) # 30Ref

X1 = refSample1[2]

X1\_map = refSample1[0]

Y1\_map = refSample1[1]

refSample2 = self.mapRef(RefAmplititudes\_file, BRF\_30, 2)

X2 = refSample2[2]

X2\_map = refSample2[0]

Y2\_map = refSample2[1]

#Y1 = np.interp(X1, X1\_map, Y1\_map)

#Y2 = np.interp(X2, X2\_map, Y2\_map)

# get the vector of k and b

'''

Y1 = self.interpolate\_list(X1\_map, Y1\_map, X1)

Y2 = self.interpolate\_list(X2\_map, Y2\_map, X2)

print("-----------------X1-------------------")

print(X1)

print("-----------------Y1-------------------")

print(Y1)

print("-----------------X2-------------------")

print(X2)

print("-----------------Y2-------------------")

print(Y2)

'''

for i in range(len(X1\_map)):

self.k.append((Y2\_map[i] - Y1\_map[i]) / (X2\_map[i] - X1\_map[i]))

self.b.append(Y2\_map[i] - self.k[i]\*X2\_map[i])

#print(len(self.k))

# By Here the k and b is absolutely correct!!!

# Now length of k, b is 60

# we need 300 k,b

# For the rest of k,b we can use interpolation algorithm to get the rest of them

self.k = self.interpolate\_list(self.k, channels)

self.b = self.interpolate\_list(self.b, channels)

'''

print("-----------------k-------------------")

print(self.k)

print("-----------------b-------------------")

print(self.b)

'''

# Need to be checked here!!!

return self.k, self.b

def getReflectance(self):

HSI = self.HSI\_info[3]

lines = self.HSI\_info[0]

channels = self.HSI\_info[1]

samples = self.HSI\_info[2]

ReflectMatrix = np.zeros((lines, channels, samples))

#print(ReflectMatrix.shape)

for idx in range(channels):

ReflectMatrix[:, idx, :] = HSI[:, idx, :] \* self.k[idx] + self.b[idx]

# Ensure the values in the reflect matrix within [0,1]

ReflectMatrix = np.where(ReflectMatrix < 0, 0, ReflectMatrix)

ReflectMatrix = np.where(ReflectMatrix > 1, 1, ReflectMatrix)

#print(ReflectMatrix.shape)

print("ReflectMatrix is obtained.")

return ReflectMatrix

# interpolation algorithm to come up with the rest of k,b

def interpolate\_list(self, input\_list, output\_length):

# x-axis raw\_Data

x = np.linspace(400, 990, len(input\_list))

# use the interpolation function

f = interp1d(x, input\_list, kind='linear',fill\_value='extrapolate')

# make the output x

output\_x = np.linspace(394, 1034, output\_length)

# interpolate

output\_list = f(output\_x)

return output\_list

def getRefBoard(self):

wavelengths = self.HSI\_info[4]

#"------------For the 3% Ref Board------------")

three\_RefAmplititudes = self.getReferAmplititudes("3")

#"------------For the 30% Ref Board------------")

thirty\_RefAmplititudes = self.getReferAmplititudes("30")

RefAmplititudes\_file ="Outputs/results/BRF/RefAmplititudes.csv"

self.writeRef(RefAmplititudes\_file, wavelengths, three\_RefAmplititudes, thirty\_RefAmplititudes)

def getLeafAvgReflect(self):

Transposed\_ReflectMatrix = np.transpose(self.ReflectMatrix, (0, 2, 1))

plant\_ReflectMatrix = Transposed\_ReflectMatrix[~self.plantMask, :]

self.AVG\_reflect = plant\_ReflectMatrix.mean(axis=0)

def getReflect(self):

self.ReflectMatrix = self.getReflectance()

self.getLeafAvgReflect()

wavelengths = self.HSI\_info[4][2:-22]

FirstRow = wavelengths

ReflectRow = self.AVG\_reflect[2:-22]

with open("Outputs/results/" + self.fileName + "/ReflectCurve.csv","w",newline='') as f:

writer = csv.writer(f)

# Write the first row

writer.writerow(FirstRow)

# Write the remaining rows

writer.writerow(ReflectRow)

def visualizeReflect(self, saveFlag):

# Write the reflectvector into a local csv file

# Plotting

x = np.array(self.HSI\_info[4])[2:-22] # only use the data within 400nm to 990nm

y = np.array(self.AVG\_reflect)[2:-22] # only use the data within 400nm to 990nm

plt.xlabel("Wavelength(nm)")

plt.ylabel("Hyperspectral Luminance")

plt.xticks(range(400, 1000, 100))

plt.plot(x,y,c='orange',label='Curve\_poly\_Fit')

#plt.xticks(range(400, 1000, 100))

plt.title("The Average Reflectance Curve of the Plant Part")

if saveFlag ==1:

plt.savefig("Outputs/figures/" + self.fileName + "/preprocess/Reflectance\_curve\_new.jpg")

elif saveFlag ==0:

plt.show()

import numpy as np

import matplotlib.pyplot as plt

import csv

from scipy.interpolate import interp1d

class Reflectance:

HSI\_info = []

cur\_proportion = 1

BRF\_positionRange = [] # position range of the refer board

BRF\_files = "" # path location fo the 3% and 30% Refer board Cali files

AVG\_reflect = [] # 1-D average reflectance for the whole plot in different bands

ReflectMatrix = [] # 3-D reflectance matrix for the whole plot

k = [] # i.e. 300 k for 300 bands

b = [] # i.e. 300 b for 300 bands

plantMask = []

BRFfilename = "BRF"

def \_\_init\_\_(self, HSI\_info, cur\_proportion, BRF\_positionRange, brf\_files, k, b, plant\_mask, filename):

self.HSI\_info = HSI\_info

#print(self.HSI\_info)

self.cur\_proportion = cur\_proportion

self.BRF\_positionRange = BRF\_positionRange

self.BRF\_files = brf\_files

self.k = k

self.b = b

self.plantMask = plant\_mask

self.fileName = filename

#print(self.BRF\_positionRange)

def getReferAmplititudes(self, BRF\_flag):

channels = self.HSI\_info[1]

HSI = self.HSI\_info[3]

#positionRange i.e. (860,83),(890,120) for 3% Ref Board

Amplititudes = []

if BRF\_flag == "3":

RefHSI = HSI[self.BRF\_positionRange[0][0][1]:self.BRF\_positionRange[0][1][1],:,self.BRF\_positionRange[0][0][0]:self.BRF\_positionRange[0][1][0]]

elif BRF\_flag == "30":

RefHSI = HSI[self.BRF\_positionRange[1][0][1]:self.BRF\_positionRange[1][1][1],:,self.BRF\_positionRange[1][0][0]:self.BRF\_positionRange[1][1][0]]

for i in range(channels):

Amplititudes.append(np.array(RefHSI[:,i,:]).mean())

return Amplititudes

def writeRef(self, filename, wavelengths, RefAmplititudes1, RefAmplititudes2):

with open(filename, "w", newline="") as f:

writer = csv.writer(f)

writer.writerow(wavelengths)

writer.writerow(RefAmplititudes1)

writer.writerow(RefAmplititudes2)

def readRef(self, filename):

with open(filename,"r") as f:

raw = f.readlines()

str = ""

str = ','.join(raw)

str = str.replace("\n","").replace("波长,反射率(%)","").replace("波长","").replace("反射率%","").replace(" ","")

str = str.split(",")

result\_list = [x for x in str if x != '']

#print("Now is",result\_list)

waves = [float(x) for x in result\_list[0::2]]

reflect = [float(x) for x in result\_list[1::2]]

return waves,reflect

def mapRef(self, inputfile, outputfile, lineNum):

X\_map, Y\_map, orig\_X = [], [], []

Read\_info = self.readRef(outputfile)

waves = Read\_info[0]

k = 0

with open(inputfile, "r") as f1:

contents = f1.readlines()

RefWaves = [float(items) for items in contents[0].split(",")]

RefReflect = [float(items) for items in contents[lineNum].split(",")]

#print (contents[0])

#print("------\n",contents[1])

channels = self.HSI\_info[1]

num = len(waves)

#print(num)

#print("RefWaves is",RefWaves)

#print("waves is", waves)

min\_diff = float('inf')

min\_index = None

'''

for k in range(num):

for i in range(channels):

diff = abs(RefWaves[i] - waves[k])

if diff < min\_diff:

min\_diff = diff

min\_index = i

if min\_index is not None:

X\_map.append(RefReflect[min\_index])

Y\_map.append(Read\_info[1][k]/100)

'''

for i in range(channels):

for k in range(num):

if abs(RefWaves[i] - waves[k]) < 1:

X\_map.append(RefReflect[i])

#print(Read\_info[1])

Y\_map.append(Read\_info[1][k]/100)

orig\_X = RefReflect

return X\_map, Y\_map, orig\_X

def getReflectEquation(self):

self.getRefBoard()

channels = self.HSI\_info[1]

RefAmplititudes\_file ="Outputs/results/BRF/RefAmplititudes.csv"

#print("------------3Ref----------")

if self.BRF\_files[0][-6:] == "30.csv":

BRF\_30 = self.BRF\_files[0]

BRF\_3 = self.BRF\_files[1]

else:

BRF\_30 = self.BRF\_files[1]

BRF\_3 = self.BRF\_files[0]

refSample1= self.mapRef(RefAmplititudes\_file, BRF\_3, 1) # 30Ref

X1 = refSample1[2]

X1\_map = refSample1[0]

Y1\_map = refSample1[1]

refSample2 = self.mapRef(RefAmplititudes\_file, BRF\_30, 2)

X2 = refSample2[2]

X2\_map = refSample2[0]

Y2\_map = refSample2[1]

#Y1 = np.interp(X1, X1\_map, Y1\_map)

#Y2 = np.interp(X2, X2\_map, Y2\_map)

# get the vector of k and b

'''

Y1 = self.interpolate\_list(X1\_map, Y1\_map, X1)

Y2 = self.interpolate\_list(X2\_map, Y2\_map, X2)

print("-----------------X1-------------------")

print(X1)

print("-----------------Y1-------------------")

print(Y1)

print("-----------------X2-------------------")

print(X2)

print("-----------------Y2-------------------")

print(Y2)

'''

for i in range(len(X1\_map)):

self.k.append((Y2\_map[i] - Y1\_map[i]) / (X2\_map[i] - X1\_map[i]))

self.b.append(Y2\_map[i] - self.k[i]\*X2\_map[i])

#print(len(self.k))

# By Here the k and b is absolutely correct!!!

# Now length of k, b is 60

# we need 300 k,b

# For the rest of k,b we can use interpolation algorithm to get the rest of them

self.k = self.interpolate\_list(self.k, channels)

self.b = self.interpolate\_list(self.b, channels)

'''

print("-----------------k-------------------")

print(self.k)

print("-----------------b-------------------")

print(self.b)

'''

# Need to be checked here!!!

return self.k, self.b

def getReflectance(self):

HSI = self.HSI\_info[3]

lines = self.HSI\_info[0]

channels = self.HSI\_info[1]

samples = self.HSI\_info[2]

ReflectMatrix = np.zeros((lines, channels, samples))

#print(ReflectMatrix.shape)

for idx in range(channels):

ReflectMatrix[:, idx, :] = HSI[:, idx, :] \* self.k[idx] + self.b[idx]

# Ensure the values in the reflect matrix within [0,1]

ReflectMatrix = np.where(ReflectMatrix < 0, 0, ReflectMatrix)

ReflectMatrix = np.where(ReflectMatrix > 1, 1, ReflectMatrix)

#print(ReflectMatrix.shape)

print("ReflectMatrix is obtained.")

return ReflectMatrix

# interpolation algorithm to come up with the rest of k,b

def interpolate\_list(self, input\_list, output\_length):

# x-axis raw\_Data

x = np.linspace(400, 990, len(input\_list))

# use the interpolation function

f = interp1d(x, input\_list, kind='linear',fill\_value='extrapolate')

# make the output x

output\_x = np.linspace(394, 1034, output\_length)

# interpolate

output\_list = f(output\_x)

return output\_list

def getRefBoard(self):

wavelengths = self.HSI\_info[4]

#"------------For the 3% Ref Board------------")

three\_RefAmplititudes = self.getReferAmplititudes("3")

#"------------For the 30% Ref Board------------")

thirty\_RefAmplititudes = self.getReferAmplititudes("30")

RefAmplititudes\_file ="Outputs/results/BRF/RefAmplititudes.csv"

self.writeRef(RefAmplititudes\_file, wavelengths, three\_RefAmplititudes, thirty\_RefAmplititudes)

def getLeafAvgReflect(self):

Transposed\_ReflectMatrix = np.transpose(self.ReflectMatrix, (0, 2, 1))

plant\_ReflectMatrix = Transposed\_ReflectMatrix[~self.plantMask, :]

self.AVG\_reflect = plant\_ReflectMatrix.mean(axis=0)

def getReflect(self):

self.ReflectMatrix = self.getReflectance()

self.getLeafAvgReflect()

wavelengths = self.HSI\_info[4][2:-22]

FirstRow = wavelengths

ReflectRow = self.AVG\_reflect[2:-22]

with open("Outputs/results/" + self.fileName + "/ReflectCurve.csv","w",newline='') as f:

writer = csv.writer(f)

# Write the first row

writer.writerow(FirstRow)

# Write the remaining rows

writer.writerow(ReflectRow)

def visualizeReflect(self, saveFlag):

# Write the reflectvector into a local csv file

# Plotting

x = np.array(self.HSI\_info[4])[2:-22] # only use the data within 400nm to 990nm

y = np.array(self.AVG\_reflect)[2:-22] # only use the data within 400nm to 990nm

plt.xlabel("Wavelength(nm)")

plt.ylabel("Hyperspectral Luminance")

plt.xticks(range(400, 1000, 100))

plt.plot(x,y,c='orange',label='Curve\_poly\_Fit')

#plt.xticks(range(400, 1000, 100))

plt.title("The Average Reflectance Curve of the Plant Part")

if saveFlag ==1:

plt.savefig("Outputs/figures/" + self.fileName + "/preprocess/Reflectance\_curve\_new.jpg")

elif saveFlag ==0:

plt.show()

import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

import cv2

import colorsys

import csv

from sklearn.cluster import KMeans

#import ReadData

# Define the thershold of the background

lower\_hsv = np.array([20, 100, 60])

upper\_hsv = np.array([60, 255, 255])

def knn(img, iter, k):

print("-----------Start to get the HSV image-------------")

#img = get\_HSVimg(img)[0]

img\_row = img.shape[0]

img\_col = img.shape[1]

img = img.reshape(-1,3) # Rshape for reducing the loops

img\_new = np.column\_stack((img, np.ones(img\_row\*img\_col))) # Add a new column

# step 1. Randomly choose clustering center points withought the background point

#cluster\_orientation = np.random.choice(img\_row\*img\_col, k, replace=False) # k positions of index

cluster\_orientation = np.random.choice(np.where(np.any(img\_new != [0, 0, 0, 1], axis=1))[0], k, replace=False)

cluster\_center = img\_new[cluster\_orientation, :] # get the pixel value according to the cluster\_center position

# Iteration

distance = [ [] for j in range(k)] # [ [], [], [], [], []]create a list,each element represents a column vector ，which stores its distance to the center pixel j

for i in range(iter):

# step 2. Calculate all the value distances between the pixels and the cluster\_center

print("Iteration time: %d" % i)

for j in range(k):

distance[j] = np.sqrt(np.sum(np.square(img\_new - cluster\_center[j]), axis=1))

# step 3. Among all the centers points, find the closest one to the current pixel and update its pixel value same as the center

orientation\_min\_dist = np.argmin(np.array(distance), axis=0) # the min value in the column vector

img\_new[:, 3] = orientation\_min\_dist # return the 3th column of the labels

# step 4. Update the jth clustering center

for j in range(k):

one\_cluster = img\_new[img\_new[:, 3] == j] # find all the pixels with label j

# Remove the backgound pixels

#print(one\_cluster.shape)

cluster\_center[j] = np.mean(one\_cluster, axis=0) # Find the mean value of the pixels and update the center

#print(img\_new.shape[0],img\_new.shape[1])

return cluster\_center, img\_new

def get\_RGBimg(HSI\_info, band1, band2, band3):

samples = HSI\_info[2]

lines = HSI\_info[0]

channels = HSI\_info[1]

HSI = HSI\_info[3]

HSI\_npArray = np.array(HSI)

img = np.zeros((lines,samples,3))

# Show RGB img

#Window = int(channels/30) # Default value: 10

Window = 10

for k in range(2\*Window):

img[:,:,0] = HSI\_npArray[:,band1-Window+k,:]

img[:,:,1] = HSI\_npArray[:,band2-Window+k,:]

img[:,:,2] = HSI\_npArray[:,band3-Window+k,:]

return img

def get\_HSVimg(RGBimg):

#rgb\_image = RGBimg.convert('RGB')

rgb\_array = np.array(RGBimg)

rgb\_array = rgb\_array / 255.0

h, s, v = np.vectorize(colorsys.rgb\_to\_hsv)(rgb\_array[:, :, 0], rgb\_array[:, :, 1], rgb\_array[:, :, 2])

h = h \* 360

s = s \* 255

v = v \* 255

hsv\_array = np.dstack((h, s, v)).astype(np.uint8)

hsv\_image = Image.fromarray(hsv\_array, mode='HSV')

'''

# Perform k-means clustering

hsv\_flat = hsv\_array.reshape((-1, 3))

kmeans = KMeans(n\_clusters=num\_clusters, random\_state=0).fit(hsv\_flat)

labels = kmeans.labels\_

centers = kmeans.cluster\_centers\_

# Reshape labels back to the original image shape

labels = labels.reshape(hsv\_array.shape[:2])

'''

return hsv\_array, hsv\_image

def getknnlabels():

#img = cv2.imread("figures/wheat/pre\_processing/Level2\_img\_new.jpg")

img = cv2.imread("figures/wheat/test/test\_new.jpg")

print("---------------Start to do KNN method----------------")

knn\_center\_num = 3 # Clustering for 3 classes: 1) Useless info; 2) Normal blades info; 3) abnormal blades;

#iter\_time = 50 # normal value

iter\_time = 10 # test value

labels\_result = knn(img, iter\_time, knn\_center\_num)

print("---------------KNN Finish----------------")

labels\_centers = labels\_result[0]

labels\_vector = labels\_result[1]

img\_row = img.shape[0]

img\_col = img.shape[1]

labels\_img = labels\_vector[:,3].reshape(img\_row, img\_col)

plt.imshow(labels\_img)

plt.savefig("figures/wheat/Clustering/test/randomTry\_knn\_cluster\_n={0}.jpg".format(knn\_center\_num))

plt.show()

# Get the indices of all pixels labeled as "abnormal blades" (class 2)

PixelSum = img\_row \* img\_col

abnormal\_proportion = 1.0

for i in range(knn\_center\_num):

print("When labels\_img ==",i)

one\_indices = np.where(labels\_img == i)

# Convert the indices to (x,y) coordinates

relavent\_pixels = np.column\_stack((one\_indices[0], one\_indices[1]))

# Get the proportion of the relavent pixels by assuming the propotion of the abnormal blades is minimum

relavent\_proportion = len(relavent\_pixels) / PixelSum

if relavent\_proportion < abnormal\_proportion:

abnormal\_proportion = relavent\_proportion # Record the propotion of the abnormal blades

abnormal\_pixels = relavent\_pixels

print(relavent\_proportion)

print(abnormal\_pixels)

return abnormal\_pixels, abnormal\_proportion

def output(filename1, filename2):

HSI\_info = ReadData.Read()

wavelengths = HSI\_info[4]

lines = HSI\_info[0]

HSI\_img = HSI\_info[3]

channels= HSI\_info[1]

samples = HSI\_info[2]

PixelSum = lines \* samples

abnormal\_pixels = getknnlabels()[0]

abnormal\_HSI = HSI\_img[abnormal\_pixels[:, 0], : ,abnormal\_pixels[:, 1]]

#print(abnormal\_HSI)

abnormal\_mean\_HSI = np.mean(abnormal\_HSI, axies=(0, 2))

print(abnormal\_HSI)

print("------------Begin to write each row-------")

height, bands, width = abnormal\_HSI.shape

spectral\_data\_2d = np.reshape(abnormal\_HSI, (height \* width, bands))

bands\_info = np.reshape(wavelengths, (1, bands))

spectral\_data\_2d\_with\_bands = np.concatenate((bands\_info, spectral\_data\_2d))

np.savetxt(filename1, spectral\_data\_2d\_with\_bands, delimiter=',')

print("------------Finish writing each row-------")

print("------------Begin to write the mean row-------")

spectral\_data\_mean\_2d = np.reshape(abnormal\_mean\_HSI, (height \* width, bands))

spectral\_data\_mean\_2d\_with\_bands = np.concatenate((bands\_info, spectral\_data\_mean\_2d))

np.savetxt(filename1, spectral\_data\_mean\_2d\_with\_bands, delimiter=',')

print("------------Finish writing the mean row-------")

if \_\_name\_\_ == "\_\_main\_\_":

test = 3

match test:

case 1:

output("test/abnormal\_HSI.csv", "test/abnormal\_mean\_HSI.csv")

case 2:

#img = Image.open("figures/wheat/pre\_processing/Level2\_img\_new.jpg")

'''

img = Image.open("figures/wheat/test.jpg")

print(type(img))

'''

img = cv2.imread("figures/wheat/test/test\_new.jpg")

get\_HSVimg(img)

case 3:

getknnlabels()

import numpy as np

from PIL import Image

class Preprocess:

HSI\_info = [] # shape = (n, k, m)

HSI = []

lines = 0

channels = 0

samples = 0

wavelengths = []

PixelSum = 0

cur\_proportion = 1

######------------------------------ Level 1 Data -------------------------------######

NDVI\_TH\_LOW = -1

NDVI\_TH\_HIGH =1

NDVI = [] # shape = (n, 1, m)

band800 = 195

band670 = 134

######------------------------------ Level 2 Data -------------------------------######

ShadowTHValue = 100 # the set mean threshold Hyspectra value for the leaves in shadow

BrightTHValue = 800 # the set mean threshold Hyspectra value for the leaves in bright

set\_value = [0, 0, 0] # set the dealt value as 0 to represent remove the useless information

BG\_Counter = 0 # the counter of the background pixels

SD\_Counter = 0 # the counter of the shadow pixels

BT\_Counter = 0 # the counter of the bright pixels

DB\_Counter = 0 # the counter of the bright pixels + shadow pixels

plant\_mask = []

def \_\_init\_\_(self, hsiInfo, ndviThLow, ndviThHigh, ampl\_LowTH, ampl\_HighTH, proportion, plant\_mask):

self.NDVI\_TH\_LOW = ndviThLow

self.NDVI\_TH\_HIGH = ndviThHigh

self.HSI\_info = hsiInfo

self.ShadowTHValue = ampl\_LowTH

self.BrightTHValue = ampl\_HighTH

self.HSI = self.HSI\_info[3]

self.lines = self.HSI\_info[0]

self.channels = self.HSI\_info[1]

self.samples = self.HSI\_info[2]

self.wavelengths = self.HSI\_info[4]

self.PixelSum = self.lines \* self.samples

self.plant\_mask = plant\_mask

self.cur\_proportion = proportion

######-------------------------- Level 1 proprocessing ---------------------------######

###----------------------------- Remove the background -----------------------------###

def getPlantPos(self):

Plant\_pos = []

BG\_Counter = 0

HSI = np.array(self.HSI)

numerator = HSI[:, self.band800, :] - HSI[:, self.band670, :]

denominator = HSI[:, self.band800, :] + HSI[:, self.band670, :]

denominator[denominator == 0] = 1 # Avoid the denominator is zero, set it as 1

NDVI = numerator / denominator

NDVI[denominator == 0] = 0 # the denominator is zero, set it as 0

# Build Mask

level1\_1\_mask = NDVI <= self.NDVI\_TH\_LOW

level1\_2\_mask = NDVI >= self.NDVI\_TH\_HIGH

level1\_mask = level1\_1\_mask | level1\_2\_mask

# Generate the HSI

Transposed\_HSI = np.transpose(HSI, (0, 2, 1)) # exChange the second and the third dimension

Transposed\_HSI[level1\_mask,:] = 0

HSI = np.transpose(Transposed\_HSI, (0, 2, 1))

# Get the plants position

Plant\_pos = np.argwhere(~level1\_mask).tolist()

# Get the number of remove

self.BG\_Counter = np.count\_nonzero(level1\_mask)

self.HSI\_info = [self.lines, self.channels, self.samples, HSI, self.wavelengths]

return HSI, np.array(Plant\_pos), self.BG\_Counter, NDVI, level1\_mask

# Get the Level1 HSI by removing the background

def getLevel1(self):

### Level 1. Remove the background

Level\_1 = self.getPlantPos()

HSI\_1 = Level\_1[0]

self.BG\_Counter = Level\_1[2]

NDVI = Level\_1[3]

level1\_mask = Level\_1[4]

HSI\_info\_L1 = [self.lines, self.channels, self.samples, HSI\_1, self.wavelengths]

self.cur\_proportion = float((self.PixelSum - self.BG\_Counter)/self.PixelSum)

print("BG\_counter is", self.BG\_Counter)

print("cur\_proportion is", self.cur\_proportion)

return HSI\_info\_L1, self.BG\_Counter, self.cur\_proportion, NDVI, level1\_mask

######---------------------------- Level 2 proprocessing ----------------------------######

###----------------------------- Remove the dark and bright ---------------------------###

def calcAmplMean(self):

return self.HSI.mean(axis=1) / self.cur\_proportion

def RemoveDB(self, Remove\_type):

HSI = self.HSI

#print(HSI\_bandmean.mean())

#print(HSI\_bandmean)

HL\_position = [] # High lighted plant position

MaxAmplMatrix = np.max(HSI, axis=1)

if Remove\_type == "SD":

level2\_1\_mask = MaxAmplMatrix < self.ShadowTHValue

self.plant\_mask = self.plant\_mask | level2\_1\_mask

# Generate the HSI

Transposed\_HSI = np.transpose(HSI, (0, 2, 1)) # exChange the second and the third dimension

Transposed\_HSI[level2\_1\_mask,:] = 0

HSI = np.transpose(Transposed\_HSI, (0, 2, 1))

self.SD\_Counter = np.count\_nonzero(level2\_1\_mask)

self.cur\_proportion = float((self.PixelSum \* self.cur\_proportion - self.SD\_Counter) / self.PixelSum)

print("cur\_proportion is", self.cur\_proportion)

elif Remove\_type == "BT":

level2\_2\_mask = MaxAmplMatrix > self.BrightTHValue

self.plant\_mask = self.plant\_mask | level2\_2\_mask

# Generate the HSI

Transposed\_HSI = np.transpose(HSI, (0, 2, 1)) # exChange the second and the third dimension

Transposed\_HSI[level2\_2\_mask,:] = 0

HSI = np.transpose(Transposed\_HSI, (0, 2, 1))

self.BT\_Counter = np.count\_nonzero(level2\_2\_mask)

print("BTCounter is",self.BT\_Counter)

self.cur\_proportion = float((self.PixelSum \* self.cur\_proportion - self.BT\_Counter) / self.PixelSum)

print("cur\_proportion is", self.cur\_proportion)

self.DB\_Counter = self.SD\_Counter + self.BT\_Counter

return HSI, self.DB\_Counter, self.plant\_mask, self.cur\_proportion

# Get the Level2 HSI by removing the shadows

def getLevel2(self):

# Remove the blades in shadow

HSI\_2, DB\_Counter, self.plant\_mask, proportion\_2 = self.RemoveDB("SD")

# Remove the blades that are too bright

HSI\_2, DB\_Counter, self.plant\_mask, proportion\_2 = self.RemoveDB("BT")

HSI\_info\_L2 = [self.lines, self.channels, self.samples, HSI\_2, self.wavelengths]

return HSI\_info\_L2, DB\_Counter, self.plant\_mask, proportion\_2

import numpy as np

import warnings

import csv

import os

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.cross\_decomposition import PLSRegression

from sklearn.model\_selection import GridSearchCV

#from sklearn.model\_selection import train\_test\_split

class Process:

Reflect\_Info = []

hsPara = ""

phenotypePara = ""

ptsthsParaModel = ""

ReflectMatrix = []

ParaMatrix = []

cur\_proportion = 1

lines = 0

channels = 0

samples = 0

waveStart = 0

FirstRow = [] # The first row to write

pls = None # the PLSR model

filename = ""

### Need to remap the band intelligently

# map\_num = ("wavelengh" - 400) / ((waveEnd - waveStart) / channels)

map\_band = {"band430":16, "band445":22, "band500":47, "band510":51,"band531":62, "band550":70, "band570":80, "band635":110, "band670":126, "band680":131, "band700":139, "band705":143, "band750":164,"band780":178, "band800":188, "band900":235,"band970":268}

def \_\_init\_\_(self, reflectInfo, hsParaType, phenotypeParaType, phenotypeParaModelType, plant\_mask, filename):

self.Reflect\_Info = reflectInfo

self.hsPara = hsParaType

self.phenotypePara = phenotypeParaType

self.phenotypeParaModel = phenotypeParaModelType

self.ReflectMatrix = self.Reflect\_Info[3]

self.lines = self.Reflect\_Info[0]

self.channels = self.Reflect\_Info[1]

self.samples = self.Reflect\_Info[2]

self.cur\_proportion = self.Reflect\_Info[5]

self.waveStart = int(float(self.Reflect\_Info[4][0]))

self.plant\_mask = plant\_mask

self.ParaMatrix = np.zeros((self.lines, self.samples))

self.filename = filename

# Train the model

data = pd.read\_csv("model/LearningData/TrainData.csv")

#print("Dataset of Train Model loaded...")

train\_x = data.drop(['SPAD',"A1200", "N", "Ca", "Cb"],axis=1)

#train\_y = data[['SPAD',"A1200", "N", "Ca", "Cb"]].copy()

train\_y = data[[self.phenotypePara]]

train\_x = pd.DataFrame(train\_x, dtype='float32')

train\_y = pd.DataFrame(train\_y, dtype='float32')

# pls\_param\_grid = {'n\_components': list(range(10,20))}

# Train the data

pls\_param\_grid = {'n\_components':[10]}

warnings.filterwarnings('ignore', category=UserWarning)

self.pls = GridSearchCV(PLSRegression(), param\_grid=pls\_param\_grid,scoring='r2',cv=10)

self.pls.fit(train\_x, train\_y)

# Calculate the relative values the photosynthesis by the design formulas

def calcHsParas(self):

#print(self.hsPara)

match self.hsPara:

case "NDVI":

numerator = self.ReflectMatrix[:,self.map\_band["band800"],:] - self.ReflectMatrix[:,self.map\_band["band680"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band800"],:] + self.ReflectMatrix[:,self.map\_band["band680"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 0 # the denominator is zero, set it as 0

# All range in [-1, 1], while plant in [0.2, 0.8]

self.ParaMatrix[self.ParaMatrix < 0] = 0

self.ParaMatrix[self.ParaMatrix > 1] = 0

case "OSAVI":

numerator = (1+0.16) \* (self.ReflectMatrix[:,self.map\_band["band800"],:] - self.ReflectMatrix[:,self.map\_band["band670"],:])

denominator = self.ReflectMatrix[:,self.map\_band["band800"],:] + self.ReflectMatrix[:,self.map\_band["band670"],:]+ 0.16

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 0 # the denominator is zero, set it as 0

case "PRI":

numerator = self.ReflectMatrix[:,self.map\_band["band531"],:] - self.ReflectMatrix[:,self.map\_band["band570"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band531"],:] + self.ReflectMatrix[:,self.map\_band["band570"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = -0.2 # the denominator is zero, set it as 0

# All range in [-1, 1], while plant in [-0.2, 0.2]

self.ParaMatrix[self.ParaMatrix < -0.2] = -0.2

self.ParaMatrix[self.ParaMatrix > 0.2] = 0.2

case "MTVI2":

numerator = 1.5 \* (1.2 \* (self.ReflectMatrix[:,self.map\_band["band800"],:] - self.ReflectMatrix[:,self.map\_band["band550"],:]) - 2.5 \* (self.ReflectMatrix[:,self.map\_band["band670"],:] - self.ReflectMatrix[:,self.map\_band["band550"],:]))

denominator = np.sqrt(((2 \* self.ReflectMatrix[:,self.map\_band["band800"],:]+1)\*2 - (6\*self.ReflectMatrix[:,self.map\_band["band800"],:]-5\*np.sqrt(self.ReflectMatrix[:,self.map\_band["band670"],:]))-0.5))

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 0 # the denominator is zero, set it as 0

case "SR":

numerator = self.ReflectMatrix[:,self.map\_band["band800"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band680"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 2 # the denominator is zero, set it as 0

self.ParaMatrix[self.ParaMatrix < 2] = 2

self.ParaMatrix[self.ParaMatrix > 8] = 8

case "DVI":

var\_1 = self.ReflectMatrix[:,self.map\_band["band800"],:]

var\_2 = self.ReflectMatrix[:,self.map\_band["band680"],:]

self.ParaMatrix = var\_1 - var\_2

case "SIPI":

numerator = self.ReflectMatrix[:,self.map\_band["band800"],:] - self.ReflectMatrix[:,self.map\_band["band445"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band800"],:] + self.ReflectMatrix[:,self.map\_band["band680"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 0 # the denominator is zero, set it as 0

self.ParaMatrix[self.ParaMatrix < 0] = 0

self.ParaMatrix[self.ParaMatrix > 2] = 2

case "PSRI":

numerator = self.ReflectMatrix[:,self.map\_band["band680"],:] - self.ReflectMatrix[:,self.map\_band["band500"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band750"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = -1 # the denominator is zero, set it as 0

self.ParaMatrix[self.ParaMatrix < -1] = -1

self.ParaMatrix[self.ParaMatrix > 1] = 1

case "CRI1":

denominator\_1 = self.ReflectMatrix[:,self.map\_band["band510"],:]

denominator\_2 = self.ReflectMatrix[:,self.map\_band["band550"],:]

denominator\_1[denominator\_1 <= 0] = 1 # Avoid the denominator is zero, set it as 1

denominator\_2[denominator\_2 <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = 1/denominator\_1 - 1/denominator\_2

self.ParaMatrix[self.ParaMatrix < 0] = 0

self.ParaMatrix[self.ParaMatrix > 15] = 0

case "CRI2":

denominator\_1 = self.ReflectMatrix[:,self.map\_band["band510"],:]

denominator\_2 = self.ReflectMatrix[:,self.map\_band["band700"],:]

denominator\_1[denominator\_1 <= 0] = 1 # Avoid the denominator is zero, set it as 1

denominator\_2[denominator\_2 <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = 1/denominator\_1 - 1/denominator\_2

self.ParaMatrix[self.ParaMatrix < 0] = 0

self.ParaMatrix[self.ParaMatrix > 15] = 15

case "ARI1":

denominator\_1 = self.ReflectMatrix[:,self.map\_band["band550"],:]

denominator\_2 = self.ReflectMatrix[:,self.map\_band["band700"],:]

denominator\_1[denominator\_1 <= 0] = 1 # Avoid the denominator is zero, set it as 1

denominator\_2[denominator\_2 <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = 1/denominator\_1 - 1/denominator\_2

self.ParaMatrix[self.ParaMatrix < 0] = 0

self.ParaMatrix[self.ParaMatrix > 0.2] = 0.2

case "ARI2":

denominator\_1 = self.ReflectMatrix[:,self.map\_band["band550"],:]

denominator\_2 = self.ReflectMatrix[:,self.map\_band["band700"],:]

denominator\_1[denominator\_1 <= 0] = 1 # Avoid the denominator is zero, set it as 1

denominator\_2[denominator\_2 <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = self.ReflectMatrix[:,self.map\_band["band800"],:] \* (1/denominator\_1 - 1/denominator\_2)

self.ParaMatrix[self.ParaMatrix < 0] = 0

self.ParaMatrix[self.ParaMatrix > 0.2] = 0.2

case "WBI":

numerator = self.ReflectMatrix[:,self.map\_band["band900"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band970"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 0 # the denominator is zero, set it as 0

# plant in [0.8, 1.2]

self.ParaMatrix[self.ParaMatrix <= 0.8] = 0.8

self.ParaMatrix[self.ParaMatrix >= 1.2] = 1.2

# Bugs remain in PSSRa and PSSRb, which is caused by the raw data of band680/band635

# (reflectance is approximately zero and make the calculation result too large)

case "PSSRa":

numerator = self.ReflectMatrix[:,self.map\_band["band800"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band680"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 2 # the denominator is zero, set it as 0

self.ParaMatrix[self.ParaMatrix < 2] = 2

self.ParaMatrix[self.ParaMatrix > 8] = 8

case "PSSRb":

numerator = self.ReflectMatrix[:,self.map\_band["band800"],:]

denominator = self.ReflectMatrix[:,self.map\_band["band635"],:]

denominator[denominator <= 0] = 1 # Avoid the denominator is zero, set it as 1

self.ParaMatrix = numerator / denominator

self.ParaMatrix[denominator == 0] = 2 # the denominator is zero, set it as 0

self.ParaMatrix[self.ParaMatrix < 2] = 2

self.ParaMatrix[self.ParaMatrix > 8] = 8

# Self defined formular

case "user-defined":

print("ok")

'''

if np.any(self.ParaMatrix > 10):

print("Yes >10")

if np.any(self.ParaMatrix < -10):

print("Yes <10")

'''

#self.ParaMatrix[self.ParaMatrix < -1] = -1

#self.ParaMatrix[self.ParaMatrix > 1] = 1

#print(self.ParaMatrix)

def draw\_pseudoColorImg(self, op\_flag, para\_flag):

#print(self.ParaMatrix)

fig, ax = plt.subplots(figsize=(6, 8))

#print(self.hsPara)

masked\_array = np.ma.array(self.ParaMatrix, mask=self.plant\_mask)

# Make a bounding box to show the plant graph only

rows, cols = np.nonzero(~self.plant\_mask)

min\_row, max\_row = np.min(rows), np.max(rows)

min\_col, max\_col = np.min(cols), np.max(cols)

# Remain some blank for the outside of the graph

min\_col\_f = min\_col - int(1/8 \*(max\_col - min\_col))

max\_col\_f = max\_col + int(1/8 \*(max\_col - min\_col))

min\_row\_f = min\_row - int(1/8 \*(max\_row - min\_row))

max\_row\_f = max\_row + int(1/8 \*(max\_row - min\_row))

# False dectection for the plot bound

if min\_col\_f < 0:

min\_col\_f = 0

if min\_row\_f < 0:

min\_row\_f = 0

if max\_col\_f > self.samples:

max\_col\_f = self.samples

if max\_row\_f > self.lines:

max\_row\_f = self.lines

cropped\_image = masked\_array[min\_row\_f:max\_row\_f+1, min\_col\_f:max\_col\_f+1]

match para\_flag:

case 1:

match self.hsPara:

case "NDVI":

im = ax.imshow(cropped\_image, cmap='gray',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on NDVI", y=1.05)

case "OSAVI":

im = ax.imshow(cropped\_image, cmap='viridis',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on OSAVI", y=1.05)

case "PSSRa":

im = ax.imshow(cropped\_image, cmap='spring',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on PSSRa", y=1.05)

case "PSSRb":

im = ax.imshow(cropped\_image, cmap='summer',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on PSSRb", y=1.05)

case "PRI":

im = ax.imshow(cropped\_image, cmap='magma',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on PRI", y=1.05)

case "MTVI2":

im = ax.imshow(cropped\_image, cmap='hot',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on MTVI2", y=1.05)

case "SR":

im = ax.imshow(cropped\_image, cmap='gray',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on SR", y=1.05)

case "DVI":

im = ax.imshow(cropped\_image, cmap='viridis',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on Ca", y=1.05)

case "SIPI":

im = ax.imshow(cropped\_image, cmap='spring',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on SIPI", y=1.05)

case "PSRI":

im = ax.imshow(cropped\_image, cmap='summer',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on PSRI", y=1.05)

case "CRI1":

im = ax.imshow(cropped\_image, cmap='magma',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on CRI1", y=1.05)

case "CRI2":

im = ax.imshow(cropped\_image, cmap='hot',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on CRI2", y=1.05)

case "ARI1":

im = ax.imshow(cropped\_image, cmap='summer',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on ARI1", y=1.05)

case "ARI2":

im = ax.imshow(cropped\_image, cmap='magma',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on ARI2", y=1.05)

case "WBI":

im = ax.imshow(cropped\_image, cmap='hot',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on WBI", y=1.05)

cbar = fig.colorbar(im)

if op\_flag == "Save":

plt.savefig("Outputs/figures/" + self.filename + "/process/" + self.hsPara + ".jpg")

plt.close()

if op\_flag == "View": # Consider to just load the figure here!!!!!!!!!

plt.show()

case 2:

im = ax.imshow(cropped\_image, cmap='viridis',interpolation='nearest')

ax.set\_title("Pseudo\_Color Map of the Relative Values on SPAD", y=1.05)

cbar = fig.colorbar(im)

if op\_flag == "Save":

plt.savefig("Outputs/figures/" + self.filename + "/process/" + self.phenotypePara + ".jpg")

plt.close()

if op\_flag == "View": # Consider to just load the figure here!!!!!!!!!

plt.show()

# Machine learning prediction

def CalcPhenotypeParas(self, index):

# Fault Value Detection

# Get the remaining parameters by using the trained model to predict

if self.phenotypeParaModel == "PLSR":

# test\_x stores the raw data for one pixel; y\_pre stores the dealt results for all pixels

test\_x = self.ReflectMatrix[index//self.samples,6:-16,index%self.samples] # The data set of the train model only contains HS in parts of wavelength range

test\_x = pd.Series(test\_x, dtype='float32')

test\_x = test\_x.to\_frame().T

self.ParaMatrix[index//self.samples, index%self.samples] = self.pls.predict(test\_x)

# export file 1 to store the spectra data

def HyperspectraCurve(self, HSI\_info, proportion):

# Show the spectra curve

wavelengths = HSI\_info[4]

lines = HSI\_info[0]

channels= HSI\_info[1]

samples = HSI\_info[2]

HSI = HSI\_info[3]

remainRow = []

spec\_mean = self.calImgSpecMean(HSI,proportion)

x = [float(num) for num in wavelengths] # change str to float

y = np.array(spec\_mean)

plt.xlabel("Wavelength(nm)")

plt.ylabel("Hyperspectral Luminance")

plt.title("The Average Hyperspectral of the Light Blades")

plt.plot(x,y,c='lightcoral',label='Curve\_poly\_Fit')

plt.savefig("Results/Hyperspec\_curve.jpg")

remainRow = spec\_mean

#plt.show()

# Export the data of hyperspectra curve into the local .csv

FirstRow = wavelengths

curveFile = "Results/Hyperspectra\_Avg\_curve.csv"

with open(curveFile,"w",newline='') as f:

writer = csv.writer(f)

# Write the first row

writer.writerow(FirstRow)

# Write the remaining rows

writer.writerow(remainRow)

return curveFile

# export file to store the Parameters of Phenotype in terms of the single plot

def exportHsParas(self, filename,idx):

FirstRow = ["photoIdx","NDVI","OSAVI", "PSSRa","PSSRb", "PRI","MTVI2","SR", "DVI", "SIPI", "PSRI", "CRI1", "CRI2", "ARI1", "ARI2", "WBI"]

# if self.hsPara not in self.FirstRow:

# Export the results

# csv\_folder = os.path.dirname(os.path.abspath(filename))

writeFlag = "w"

if idx > 1:

writeFlag = "a"

with open(filename,writeFlag,newline='') as f:

writer = csv.writer(f)

if idx == 1:

writer.writerow(FirstRow)

meanHsPara = [] # initialize

meanHsPara.append(idx)

for j in range(len(FirstRow)-1):

self.hsPara = FirstRow[j+1]

self.calcHsParas()

non\_zero\_matrix = self.ParaMatrix[self.ParaMatrix != 0]

meanHsPara.append(np.mean(non\_zero\_matrix)) # bugs remain here

writer.writerow(meanHsPara)

# subprocess.run(['start', '', csv\_folder], shell=True)

def exportPhenotypeParas(self, filename, idx):

FirstRow = ["Idx","file","SPAD", "A1200", "N", "Ca", "Cb"]

# if self.hsPara not in self.FirstRow:

# Export the results

# csv\_folder = os.path.dirname(os.path.abspath(filename))

writeFlag = "w"

if idx > 1:

writeFlag = "a"

with open(filename, writeFlag, newline='') as f:

writer = csv.writer(f)

if idx == 1:

writer.writerow(FirstRow)

dataRow = [] # initialize

dataRow.append(idx)

dataRow.append(self.filename[:-4])

for j in range(len(FirstRow)-1):

if self.phenotypeParaModel == "PLSR":

data = pd.read\_csv("model/LearningData/TrainData.csv")

#print("Dataset of Train Model loaded...")

train\_x = data.drop(['SPAD',"A1200", "N", "Ca", "Cb"],axis=1)

#train\_y = data[['SPAD',"A1200", "N", "Ca", "Cb"]].copy()

match j:

case 0:

self.phenotypePara = "SPAD"

case 1:

self.phenotypePara = "A1200"

case 2:

self.phenotypePara = "N"

case 3:

self.phenotypePara = "Ca"

case 4:

self.phenotypePara = "Cb"

train\_y = data[[self.phenotypePara]]

train\_x = pd.DataFrame(train\_x, dtype='float32')

train\_y = pd.DataFrame(train\_y, dtype='float32')

# pls\_param\_grid = {'n\_components': list(range(10,20))}

# Train the data

pls\_param\_grid = {'n\_components':[10]}

warnings.filterwarnings('ignore', category=UserWarning)

pls = GridSearchCV(PLSRegression(), param\_grid=pls\_param\_grid,scoring='r2',cv=10)

pls.fit(train\_x, train\_y)

# test\_x stores the raw data for one pixel; y\_pre stores the dealt results for all pixels

expanded\_mask = np.expand\_dims(self.plant\_mask, axis=1)

expanded\_mask = np.repeat(expanded\_mask, self.channels, axis=1)

masked\_array = np.ma.array(self.ReflectMatrix, mask=expanded\_mask)

test\_x = np.mean(masked\_array[:, 6:-16, :],axis=(0,2)) # The data set of the train model only contains HS in parts of wavelength range

test\_x = pd.Series(test\_x, dtype='float32')

test\_x = test\_x.to\_frame().T

predict\_y = pls.predict(test\_x)

dataRow.append(predict\_y[0][0])

writer.writerow(dataRow)

# subprocess.run(['start', '', csv\_folder], shell=True)

import numpy as np

from PIL import Image

import re

import csv

# Test function

def Read():

HSI\_info = ReadData("M:/m-CTP\_DATA/2023.1.9/wheat/TASK2023-01-08-02-42/Hyperspectral/2023-01-08-06-01-59.hdr","M:/m-CTP\_DATA/2023.1.9/wheat/TASK2023-01-08-02-42/Hyperspectral/2023-01-08-06-01-59.spe", 1)

print("Successfully read the testPlant HSI")

return HSI\_info

def ReadRef():

HSI\_info = ReadData("M:/m-CTP\_DATA/2023.1.9/TeeSapling/wave.hdr",'M:/m-CTP\_DATA/2023.1.9/TeeSapling//2022-07-27-06-32.spe', 0)

print("Successfully read the refBoard HSI")

return HSI\_info

# Useful function

def ReadData(hdrfileName,spefileName, flag):

data = []

### Read .hdr file to store the infomation

with open(hdrfileName) as hdr\_file:

for num, line in enumerate(hdr\_file):

data.append(line.split(" "))

#print(data)

wavelengths= []

waveFlag = 0

for row in range(len(data)):

if data[row][0] == 'lines':

#print(data[row])

if (flag==0):

lines = int(195840000/300/480)

if (flag==1):

try:

lines = int(re.findall("\d{4}",data[row][2])[0])

except:

lines = int(re.findall("\d{3}",data[row][2])[0])

continue

if data[row][0] == 'samples':

#print(data[row])

samples = int(re.findall("\d{3}",data[row][2])[0])

continue

if data[row][0] == 'bands':

#print(data[row])

channels = int(re.findall("\d{3}",data[row][2])[0])

continue

if data[row][0] == "wavelength" and data[row][1] != "units":

'''

# Version for the previous CCD

str = data[row][2]

str = str.replace("{","")

str = str.replace("\n","")

wavelengths.append(str)

'''

waveFlag = 1

continue

if waveFlag == 1:

data\_row = data[row]

str = ''.join(data[row])

str = str.replace("\n","")

str = str.replace("}","")

wavelengths.append(str)

#print(wavelengths)

raw = ""

wavelengths = raw.join(wavelengths)

wavelengths = wavelengths.split(",")

### Read .spe file

# Load file and reshape

imgs = np.fromfile(spefileName, dtype=np.int16).reshape(lines,channels,samples)

#imgs = np.fromfile('M:/m-CTP\_DATA/2023.1.9/TeeSapling/2022-07-27-06-21.spe', dtype=np.int16).reshape(lines,channels,samples)

imgs = imgs.astype(np.int16) # change the value range from (0, 4095) to (0, 255)

return lines, channels, samples, imgs, wavelengths

def drawImg(HSI\_info):

lines = HSI\_info[0]

channels = HSI\_info[1]

samples = HSI\_info[2]

HSI = HSI\_info[3]

newimg = Image.new('RGB',(samples,lines))

for i in range(lines):

for j in range (samples):

newimg.putpixel((j,i),((int(HSI[i][105][j]\*256/4095)),(int(HSI[i][59][j]\*256/4096)),(int(HSI[i][34][j]\*256/4096))))

return(newimg)

from sklearn.cross\_decomposition import PLSRegression

from sklearn.model\_selection import GridSearchCV

import numpy as np

from sklearn.model\_selection import train\_test\_split

import pandas as pd

from sklearn.metrics import mean\_squared\_error, r2\_score,mean\_absolute\_error

data = pd.read\_csv("LearningData/TrainData.csv")

print("Dataset loaded...")

x = data.drop(['SPAD',"A1200", "N", "Ca", "Cb"],axis=1)

y1 = data[[ 'Ca', 'Cb']].copy()

x = pd.DataFrame(x, dtype='float32')

y1 = pd.DataFrame(y1, dtype='float32')

#train\_x, test\_x, train\_y, test\_y = train\_test\_split(x,y1,test\_size=0.2)

train\_x = x[:165]

train\_y = y1[:165]

# pls\_param\_grid = {'n\_components': list(range(10,20))}

pls\_param\_grid = {'n\_components':[10]}

pls = GridSearchCV(PLSRegression(), param\_grid=pls\_param\_grid,scoring='r2',cv=10)

#y\_pre = pls.predict(test\_x)

test\_x = x.tail(81) # u

#print(test\_x)

pls.fit(train\_x, train\_y)

y\_pre = pls.predict(test\_x)

new\_data = y\_pre

data.iloc[-81:,-2:] = new\_data1data.to\_csv('LearningData/NewTrainData.csv', mode='a', header=True, index=False)1'''

train\_r2=r2\_score(train\_y,y\_pre)

train\_mse = mean\_squared\_error(train\_y, y\_pre)

train\_RMSE=np.sqrt(train\_mse)

y\_test\_pre = pls.predict(test\_x)

test\_r2=r2\_score(test\_y, y\_test\_pre)

test\_mse = mean\_squared\_error(test\_y, y\_test\_pre)

test\_RMSE=np.sqrt(test\_mse)

mae = mean\_absolute\_error(test\_y, y\_test\_pre)

rpd = np.std(test\_y) / test\_RMSE

print("训练R2:",train\_r2)

print("训练RMSE:",train\_RMSE)

print("测试R2:",test\_r2)

print("测试RMSE:",test\_RMSE)

print("测试MSE:",test\_mse)

print("测试MAE = ",mae)

print("RPD =", rpd)

'''

# F