数字图像处理期末大作业项目报告

华东师范大学软件工程学院密码与网络安全系 徐艺帆 10205101516

1.项目摘要

本项目为数字图像处理期末大作业。根据项目要求并进行需求分析后，我将本次项目分成了两个方向、七个模块：1-5为数字图像处理的基本功能，包括了：图像的基本操作、直方图统计与均衡、基本运算与空域运算、直线检测与边缘检测、图像频域计算；6-7为场景明确的基于经典深度学习模型的两大功能：风格迁移、目标检测与性别分类。

UI设计与后端设计上，主要基于PYQT5与QT Designer，并在部分前台窗口使用了诸如多线程、信息槽等技巧，保障程序能够流畅运行。软件测试方面，全程采取多轮黑盒测试与白盒测试，并在程序主要功能完成后尝试运用正交表进行自动化测试。总体来说，基本完成所需要求。

2.写在前面

本次项目布置较早、验收较晚，曹老师给了我们相当充足的时间进行自学相关知识和构思本次项目，因此准备时间也较为充裕。由于在2021年我跨年级选修了赵老师的机器学习，在2022年寒假学习了斯坦福大学的cs231n（计算机视觉），在数字图像处理和计算机视觉上有一定的理论基础和充足的兴趣，因此在项目前期（3-6月）的准备工作中，我的主要精力还是放在复现经典模型与炼丹上。本次项目给予了我一个非常好的实践机会，让我能够以软件工程的思维去衡量整个AI项目的开发流程。

项目运行的环境如下表所示：

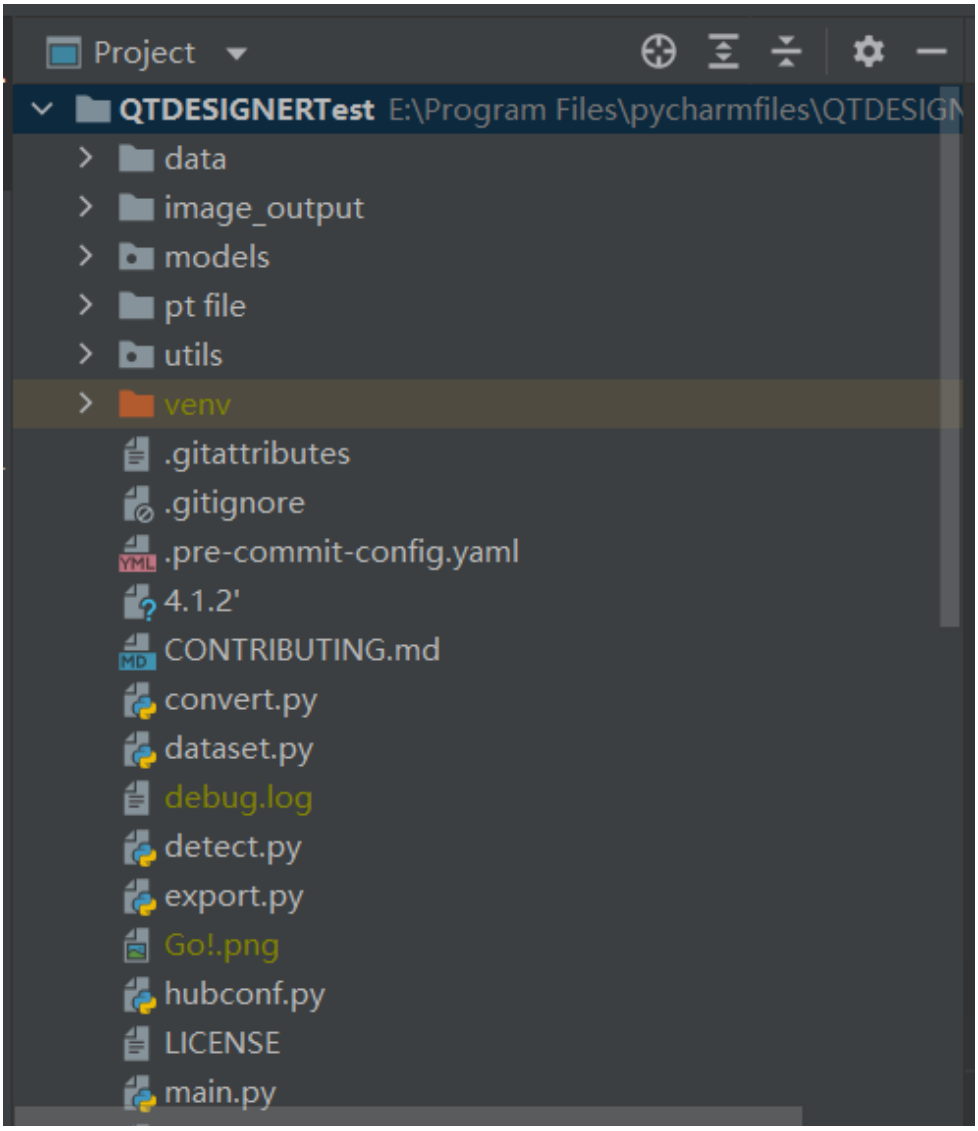
测试环境：
机型：联想拯救者R7000 2020

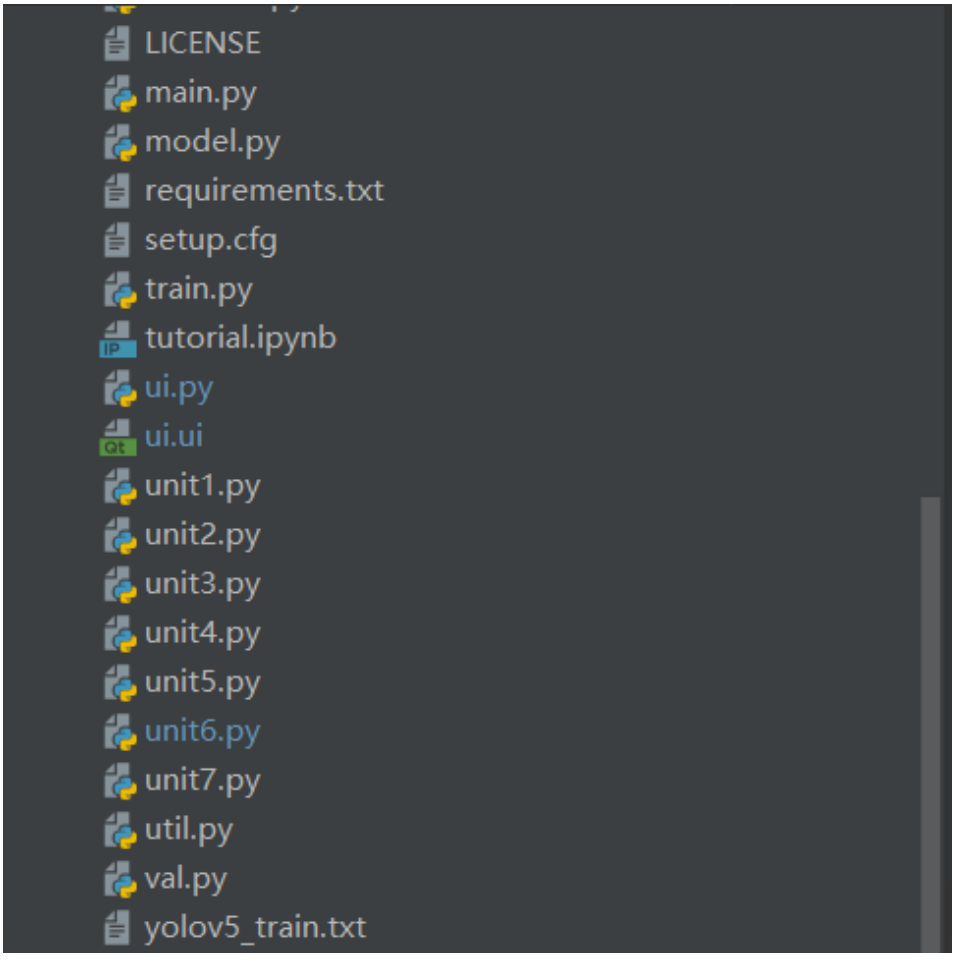
```
CPU: AMD R-4800H
GPU: RTX 2060(6G)
依赖:
# pip install -r requirements.txt

# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0
pyqt5
# Logging -------------------------------------
tensorboard>=2.4.1
# wandb

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Export --------------------------------------
# coremltools>=4.1  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.3.6  # ONNX simplifier
# scikit-learn==0.19.2  # CoreML quantization
# tensorflow>=2.4.1  # TFLite export
# tensorflowjs>=3.9.0  # TF.js export
# openvino-dev  # OpenVINO export

# Extras --------------------------------------
# albumentations>=1.0.3
# Cython  # for pycocotools https://github.com/cocodataset/cocoapi/issues/172
# pycocotools>=2.0  # COCO mAP
# roboflow
thop  # FLOPs computation
```

3.项目介绍

3.1.基础功能设计思路

项目文档设计结构:


Project
QTDESIGNERTest E:\Program Files\pycharmfiles\QTDESIGN
data
image_output
models
pt file
utils
venv
.gitattributes
.gitignore
.pre-commit-config.yaml
4.1.2'
CONTRIBUTING.md
convert.py
dataset.py
debug.log
detect.py
export.py
Go!.png
hubconf.py
LICENSE
main.py


LICENSE
main.py
model.py
requirements.txt
setup.cfg
train.py
tutorial.ipynb
ui.py
ui.ui
unit1.py
unit2.py
unit3.py
unit4.py
unit5.py
unit6.py
unit7.py
util.py
val.py
yolov5_train.txt

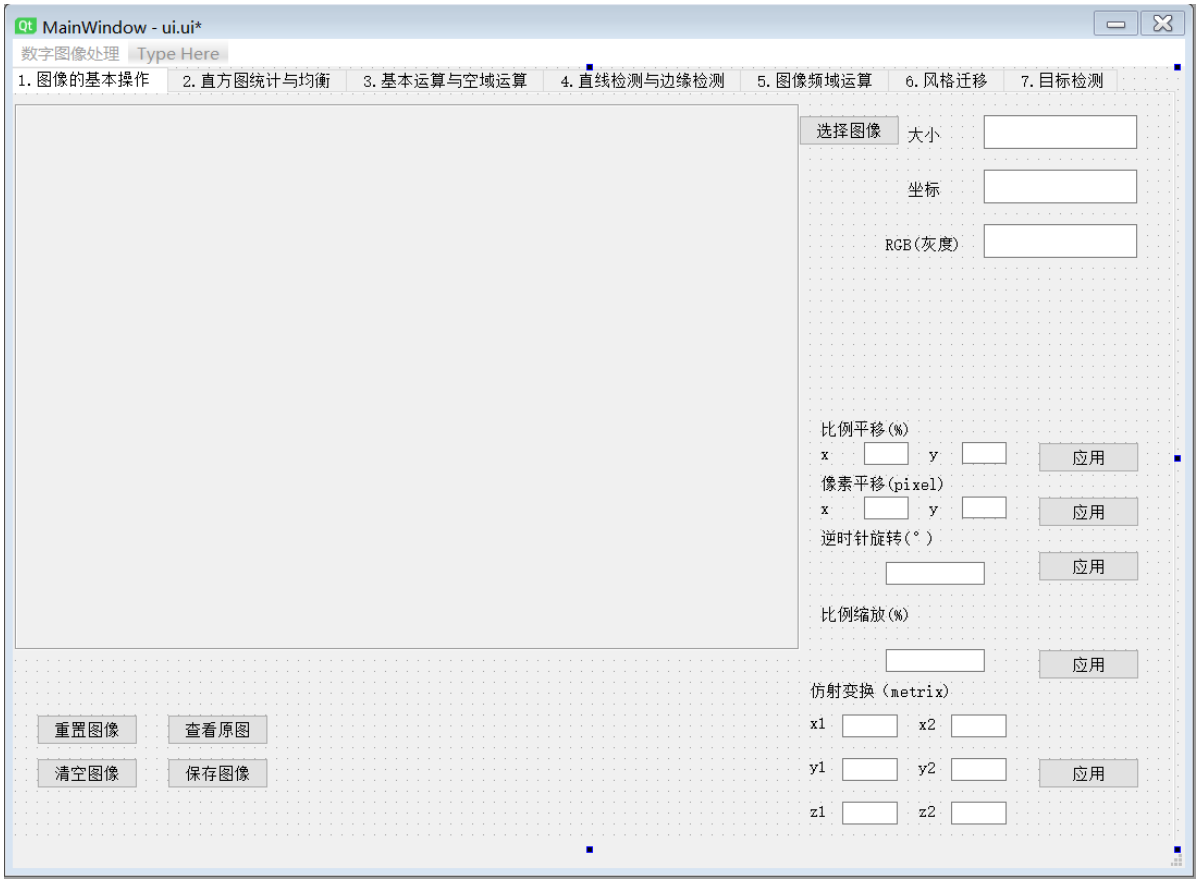
解释：

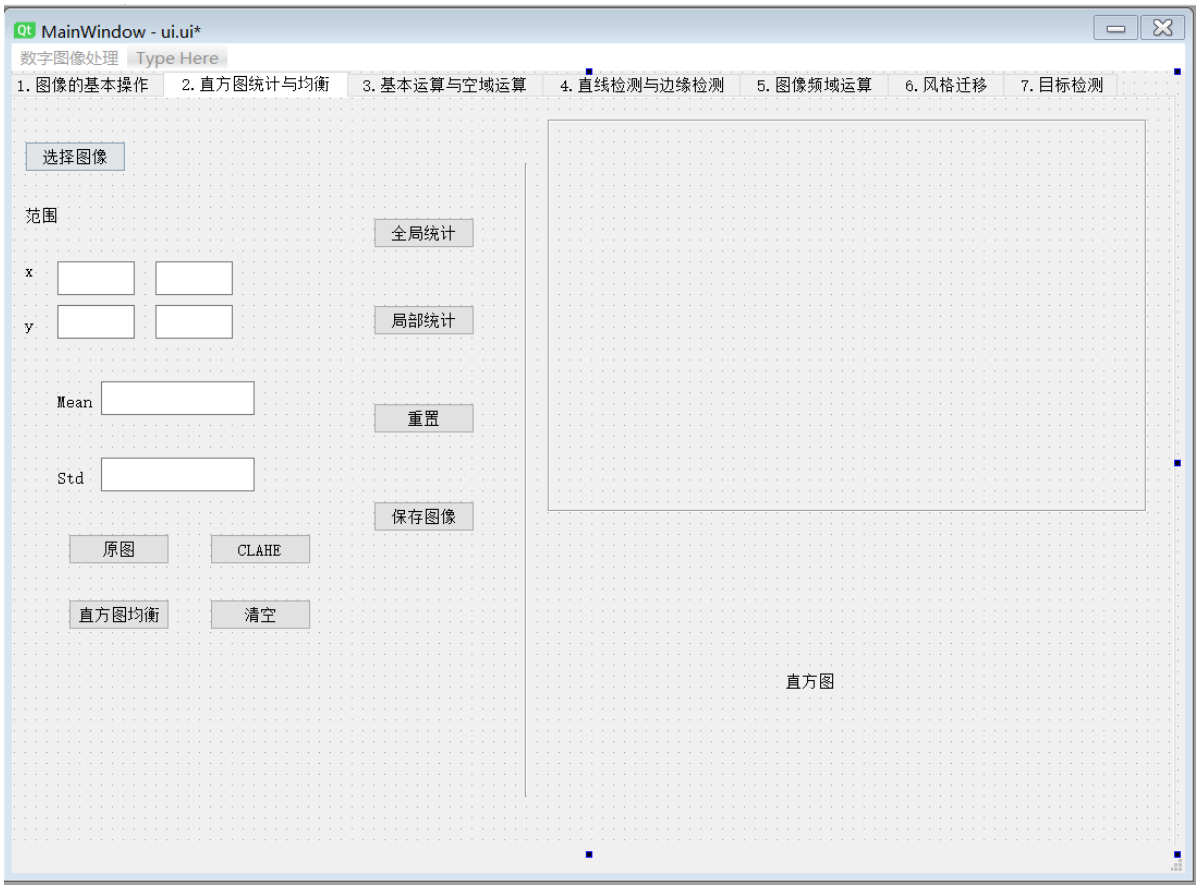
1. data文件夹：存放yolov5系列的各预训练数据集，已废弃(unit7)
2. image_output: 测试时用于存放unit7结果图片
3. models：存放yolov5各系列网络架构，已废弃(unit7)
4. pt file：存放已经训练好的预训练权重(unit7)
5. utils: unit7的网络架构中需要用到的函数工具包
6. coverts.py: unit7部分，在制作数据集时用到的label格式转换脚本，已废弃
7. datasets.py: unit7部分，在导入数据集时用于切分训练集、测试集、验证集，读取标签并进行标准化命名的脚本，已废弃
8. detect.py: unit7的预测功能主文件
9. export.py: unit7训练阶段的扩充训练方法，例如YOLOv5 ONNX，已废弃
10. hubconf.py: unit7训练阶段的hub文件，可以通过hub文件在线下载预训练权重
11. main.py: unit1-unit7槽函数绑定、主副线程初始化
12. model.py: unit6部分，引入VGG16以进行风格迁移
13. requirements.txt: 本项目依赖表
14. train.py: unit7训练文件，已废弃
15. ui.py: ui文件，由ui.ui转换
16. ui.ui: ui文件，用pyuic将其从.ui转换成.py
17. unit1.py-unit7.py: 各部分槽函数和线程初始化，主要内容
18. util.py: unit6的工具包，包括了tensor to PIL/numpy、PIL/numpy to tensor、 normal generalization等
19. val.py: unit7的验证文件，已废弃

3.1.1. UI设计：

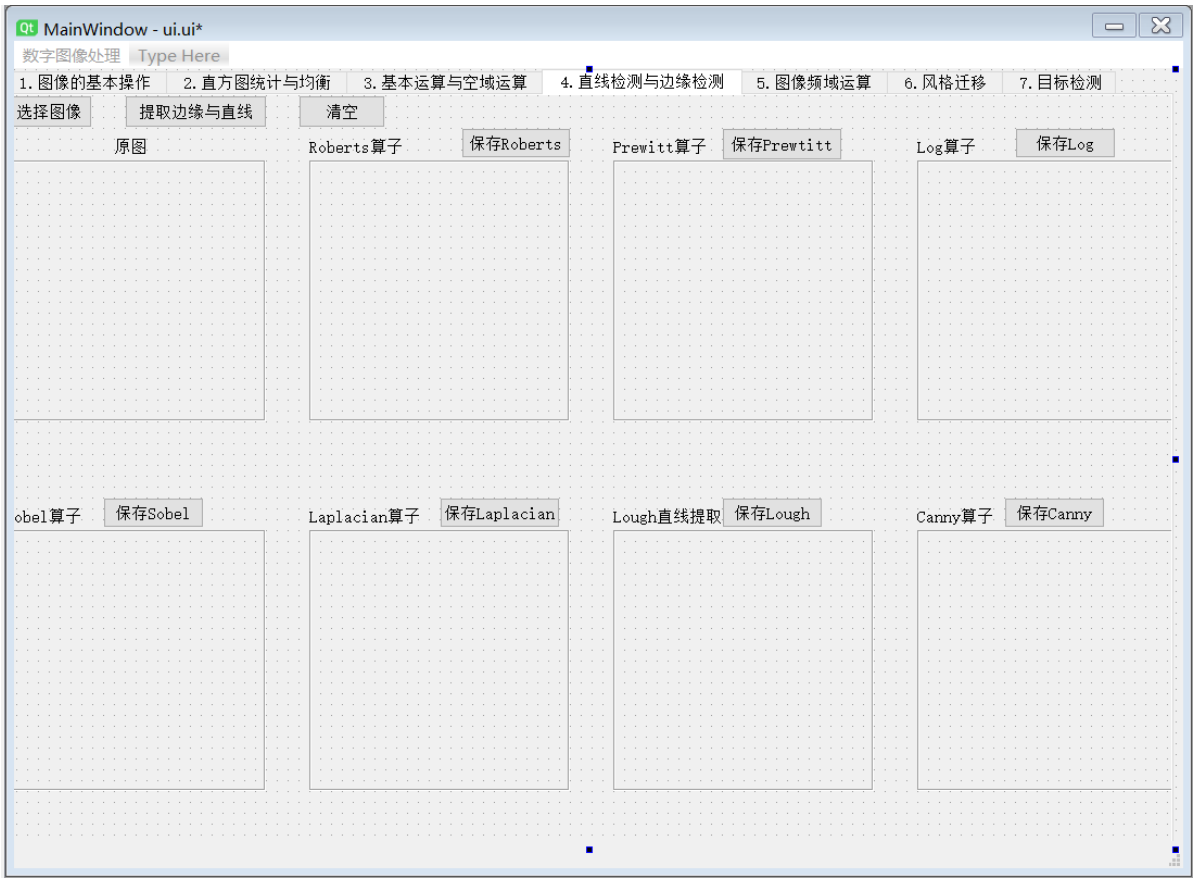
主要采取了PYQT5基本控件来设计UI界面，包括QMainWindow、QWedget、QTabWedget、QLabel、QTextBrowser、QpushButton、QLabel等...

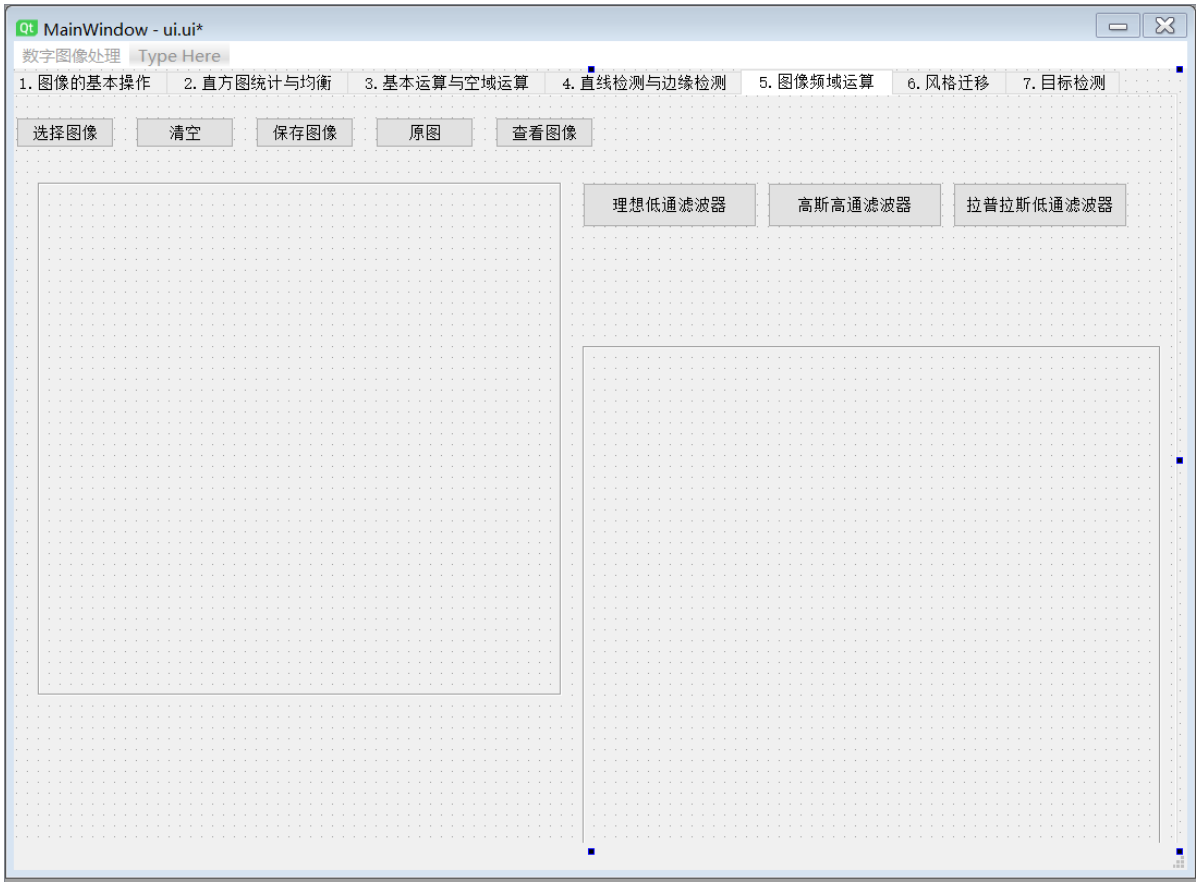
在处理部分界面时，由于程序执行时间较长，主线程常常会在Windows环境下无响应。因此采取了界面动态刷新和多线程的方法，一定程度上避免了这种问题的存在。具体的UI设计如下图所示：

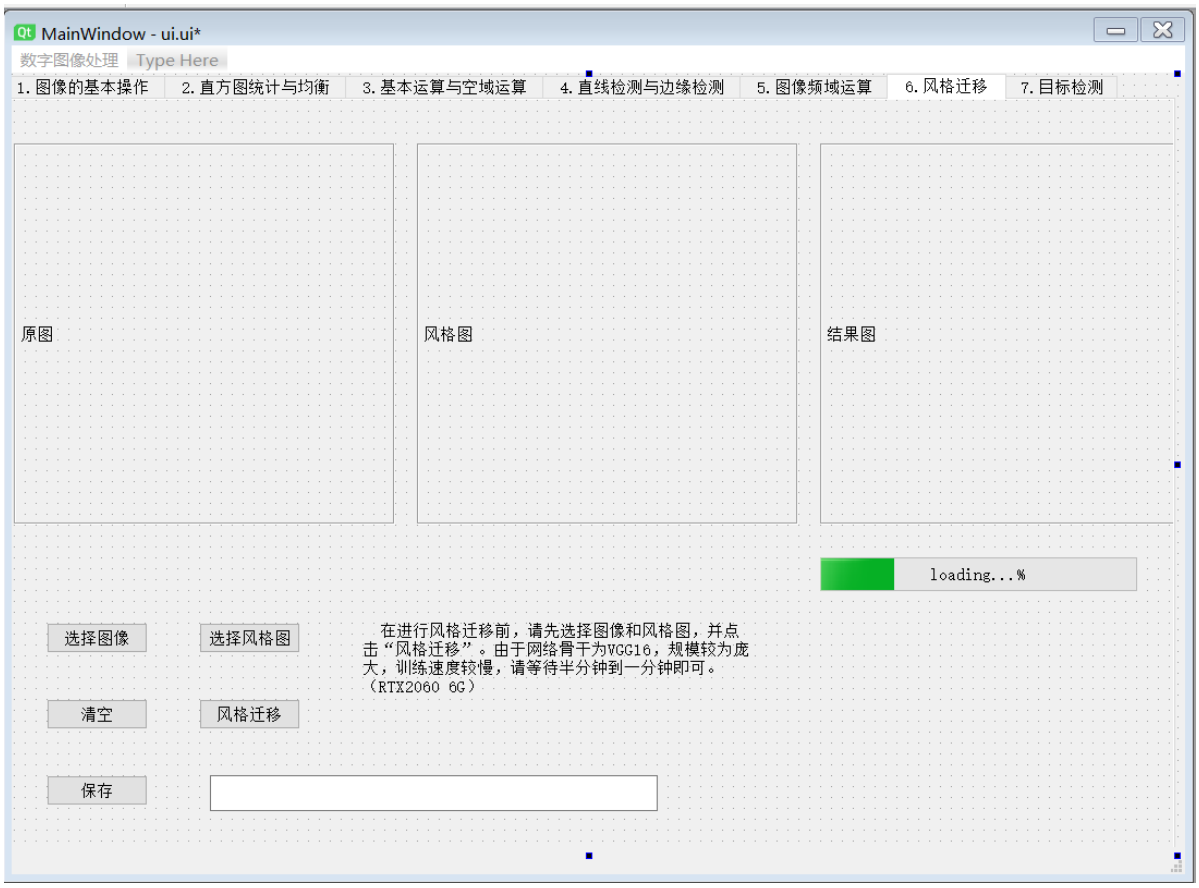

MainWindow - ui.ui*
数字图像处理
Type Here
1. 图像的基本操作
2. 直方图统计与均衡
3. 基本运算与空域运算
4. 直线检测与边缘检测
5. 图像频域运算
6. 风格迁移
7. 目标检测
选择图像
大小
坐标
RGB(灰度)
比例平移(%)
x
y
应用
像素平移(pixel)
x
y
应用
逆时针旋转(°)
应用
比例缩放(%)
应用
仿射变换（metrix）
x1
x2
y1
y2
z1
z2
应用
重置图像
查看原图
清空图像
保存图像

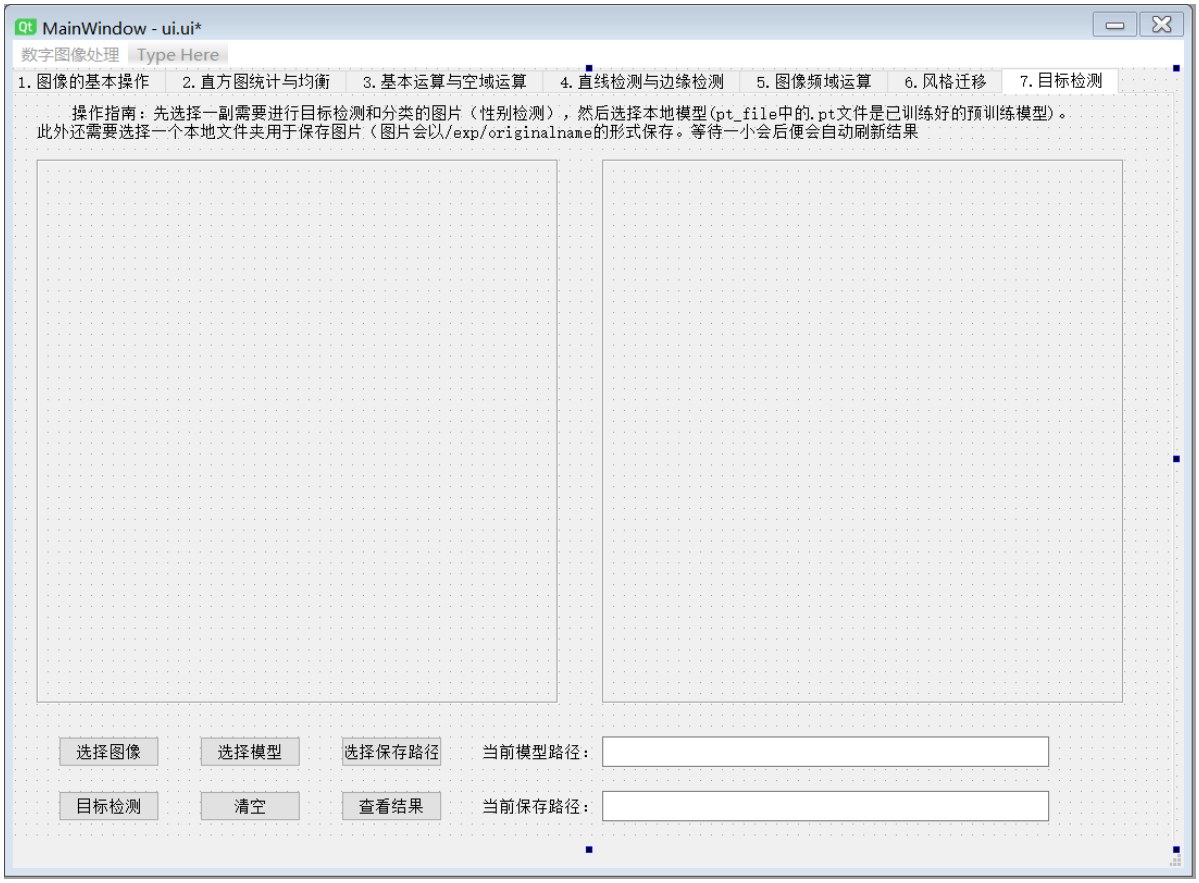

MainWindow - ui.ui*
数字图像处理
Type Here
1. 图像的基本操作
2. 直方图统计与均衡
3. 基本运算与空域运算
4. 直线检测与边缘检测
5. 图像频域运算
6. 风格迁移
7. 目标检测
选择图像
范围
x
y
Mean
Std
全局统计
局部统计
重置
保存图像
原图
CLAHE
直方图均衡
清空
直方图


MainWindow - ui.ui*
数字图像处理
Type Here
1. 图像的基本操作
2. 直方图统计与均衡
3. 基本运算与空域运算
4. 直线检测与边缘检测
5. 图像频域运算
6. 风格迁移
7. 目标检测
pic 1
pic 2
result
pic 3
result
算术运算
选择图像1
选择图像2
ADD
SUB
MULTI
DIVIDE
逻辑运算
与
或
非
清空
保存
查看
左侧部分包括了图片的算术运算和逻辑运算，其中算术运算包括了加减乘除四种运算，逻辑运算包括了与或非三种运算。对于其中的双目运算符，需要选择PIC1和PIC2合成出新的图片RESULT，对于单目运算符，会忽略PIC2，直接操作PIC。
右侧部分包括了图像结构元和算子、滤波的单目运算。运算前需要导入图像，并输入结构元初始化参数规格，并选择是否添加padding以及在原图还是在现有图片上操作。最后选择滤波按钮便可得到RESULT图片
选择图像
高斯噪声
椒盐噪声
结构元参数初始化
*
叠加至原图
腐蚀
开
膨胀
闭
拉普拉斯滤波
均值滤波
清空
中值滤波
高斯滤波
保存
双边滤波
查看
半径
色彩标准差
空间标准差


MainWindow - ui.ui*
数字图像处理
Type Here
1. 图像的基本操作
2. 直方图统计与均衡
3. 基本运算与空域运算
4. 直线检测与边缘检测
5. 图像频域运算
6. 风格迁移
7. 目标检测
选择图像
提取边缘与直线
清空
原图
Roberts算子
保存Roberts
Prewitt算子
保存Prewtitt
Log算子
保存Log
obel算子
保存Sobel
Laplacian算子
保存Laplacian
Lough直线提取
保存Lough
Canny算子
保存Canny


MainWindow - ui.ui*
数字图像处理
Type Here
1. 图像的基本操作
2. 直方图统计与均衡
3. 基本运算与空域运算
4. 直线检测与边缘检测
5. 图像频域运算
6. 风格迁移
7. 目标检测
选择图像
清空
保存图像
原图
查看图像
理想低通滤波器
高斯高通滤波器
拉普拉斯低通滤波器


MainWindow - ui.ui*
数字图像处理
Type Here
1. 图像的基本操作
2. 直方图统计与均衡
3. 基本运算与空域运算
4. 直线检测与边缘检测
5. 图像频域运算
6. 风格迁移
7. 目标检测
原图
风格图
结果图
loading...%
选择图像
选择风格图
在进行风格迁移前，请先选择图像和风格图，并点击“风格迁移”。由于网络骨干为VGG16，规模较为庞大，训练速度较慢，请等待半分钟到一分钟即可。
（RTX2060 6G）
清空
风格迁移
保存



3.1.2. 主体功能设计

1. unit1：图像的基本操作：主要包括了导入、导出图像、显示图像、显示图像坐标、灰度、图像缩放、图像平移和仿射变换功能。
2. unit2：直方图与均衡：主要包括了显示图像及其全局直方图、局部直方图，并提供了clahe、直方图均衡等接口，实现直方图及其基本功能。
3. unit3：基本运算和空域运算：包括两部分：左半边是图像的代数运算和逻辑运算；右半边是图像的空域运算，包括了添加噪声（椒盐噪声、高斯噪声）、形态学操作（开、闭，腐蚀、膨胀）、添加空域滤波（中值滤波、高斯滤波、双边滤波、均值滤波、拉普拉斯滤波）。左右两个模块相互独立。
4. unit4：边缘检测与直线检测：主要实现了6种边缘检测算子(log,laplacian,canny,prewitt,sobel)和1种直线检测算法(lough)。
5. unit5：图像频域操作：实现了三种频域滤波（理想低通，高斯高通，拉普拉斯低通）操作并展示其直方图。
6. unit6：风格迁移：基于论文《A Neural Algorithm of Artistic Style》[1]

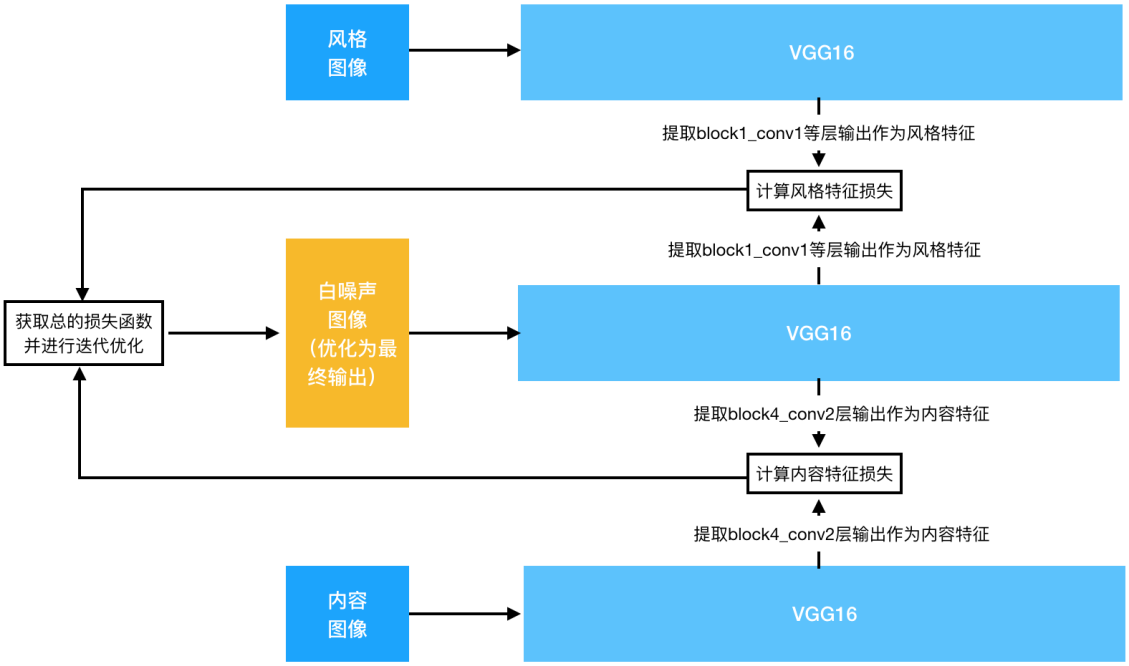
   ,学习并复现了基于VGG16的固定风格迁移网络
7. unit7：目标检测与性别分类：基于yolov5模型，在自己利用labelme制作的人像数据集（约300-350张）上训练出的预训练权重文件上进行离线检测[2]

3.2. 深度学习模块实践报告：

3.2.1.风格迁移网络报告：

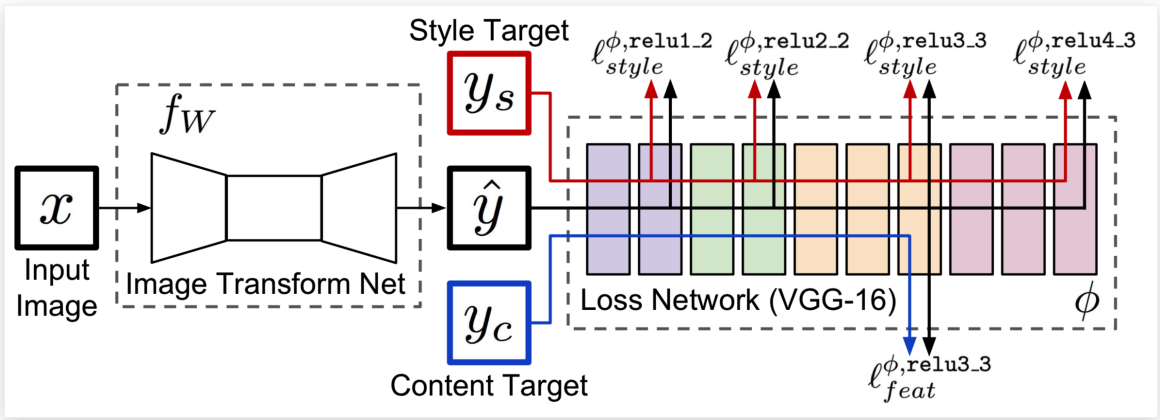
本次项目我阅读的论文是风格迁移开山之作《A Neural Algorithm of Artistic Style》。

在图像处理任务中最强大的一类深度神经网络被称为卷积神经网络。卷积神经网络由多层小型计算单元组成，这些计算单元以前馈方式分层处理视觉信息。论文中算法的本质是利用CNN抽象出输入图像的特征。将风格图输入CNN，将部分层输出为风格特征；将内容图像输入CNN，将部分层输出为内容特征。再优化一个随机图像（一般是全噪声图像），利用类似于GAN（生成对抗网络）的思想，使得其在该卷积神经网络的对应层输出不断接近之前得到的风格特征与内容特征。（事实上，GAN在随后的迁移学习中被广泛运用）



正文中介绍的结果是在VGG19网络的基础上生成的。在VGG论文中，作者使用了19层VGG-Network的16个卷积层和5个池化层提供的特征空间，取代了之前神经网络中常见的全连接层，有利于优化在全连接神经网络中常出现的过拟合、参数爆炸与反向传播困难的情况。对于图像合成，用average pooling代替max-pooling操作可以改善反向传播梯度流。

对于一个神经网络Q,定义其给定的输入图像X。内容特征方面，我们的目的是获取其在 relu1_2, relu2_2, relu3_3, relu4_3 中输出的特征图：



这里采取了修改后的VGG16：（也是原作的思路）

```
    def __init__(self, features):
        super(VGG, self).__init__()
        self.features = features
        self.layer_name_mapping = {
            '3': "relu1_2",
            '8': "relu2_2",
            '15': "relu3_3",
            '22': "relu4_3"
        }
        for p in self.parameters():
            p.requires_grad = False

    def forward(self, x):
        outs = []
        for name, module in self.features._modules.items():
            x = module(x)
            if name in self.layer_name_mapping:
                outs.append(x)
        return outs
```

定义损失函数为生成图像与目标图像对应层输出的特征图的均方损失MSE：

$L_{content} = \frac{1}{2}\sum_{i,j}(F_{ij}^{l} - Q_{i,j}^{l})^2$ 其中F为噪声图像产生的内容特征输出，P为输入图像产生的内容特征输出，l为层
也即等价于损失:

$$\frac{\partial L}{\partial F^{l}} = \begin{cases} \left(\mathrm{F}_{ij}^{l} - \mathrm{Q}_{ij}^{l}\right)_{\mathrm{ij}} & \text{if } \mathrm{F}_{\mathrm{ij}}^{l} > 0 \\ 0 & \text{if } \mathrm{F}_{\mathrm{ij}}^{l} < 0 \end{cases}$$

对于风格而言，网络的每一层中的响应特征值，建立了一种风格表示。该风格表示计算不同层响应特征值之间的相关性，这些特征相关性由Gram矩阵，也就是向量化特征的内积构成。也即
$G = F * F^{T}, G_{ij} = \sum_{k} F_{ik}F_{jk}$。其风格损失函数应为：

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{ij}(G_{ij} - A_{ij})^2$$

$$loss = \sum_{l} E_l$$

接下来就定义损失函数并进行梯度优化即可。主体代码如下所示：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from PIL import Image
import matplotlib.pyplot as plt

import torchvision.transforms as transforms
import torchvision.models as models
from PyQt5 import QtGui
from PyQt5.QtWidgets import *
import cv2
import numpy as np
```

```
from PyQt5 import QtWidgets, QtCore
import sys
from PyQt5.QtCore import *
import time

import util
from util import *
from model import *

#images = np.ndarray(())

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

#class Runthread(QtCore.QThread):
    #  通过类成员对象定义信号对象
   # signal = pyqtSignal(int)
  #  filename2 = ''
  #  def __init__(self):
     #   super(Runthread, self).__init__()

  #  def __del__(self):
     #   self.wait()

  #  def setParam(self, file1, file2):
       # self.filename1 = file1
       # self.filename2 = file2




def init(self):
    self.unit6_img1 = np.ndarray(())
    self.unit6_img2 = np.ndarray(())
    self.unit6_result = np.ndarray(())
    self.unit6_img1_channel = 3
    self.unit6_img2_channel = 3
    self.unit6_result_channel = 3
    self.filepath1 =''
    self.filepath2 =''
    self.ui.progressBar.setValue(0)
    self.ui.progressBar.setMaximum(100)
    self.ui.textBrowser_8.setText('您的设备有cuda，可以运行'if
torch.cuda.is_available() else "您的设备没有cuda，不建议运行")

def img_load1(self):#导入原图
    fileName, tmp = QFileDialog.getOpenFileName(self, '打开图像', 'Image', '*.png
*.jpg *.bmp *.jpeg')
    if fileName == '':
        return
    self.unit6_img1 = np.ndarray(())
    self.filepath1 = fileName
    self.unit6_img1 = cv2.imread(fileName, -1)
    if len(self.unit6_img1.shape) == 3:
        self.unit6_img1_channel = 3
        if self.unit6_img1.shape[2] == 4:
            self.unit6_img1 = cv2.cvtColor(self.unit6_img1, cv2.COLOR_BGRA2BGR)
    else:
        msg_box = QMessageBox(QMessageBox.Warning, "不是彩图", '请选择彩图进行风格迁移
')
```

```
            msg_box.exec_()
            init(self)
            return
        print(self.unit6_img1.shape)
        img_refresh(self)

    def img_load2(self):#导入风格图
        fileName, tmp = QFileDialog.getOpenFileName(self, '打开图像', 'Image', '*.png
    *.jpg *.bmp *.jpeg')
        if fileName == '':
            return
        self.unit6_img2 = np.ndarray(())
        self.filepath2 = fileName
        self.unit6_img2 = cv2.imread(fileName, -1)
        if len(self.unit6_img2.shape) != 3:
            msg_box = QMessageBox(QMessageBox.Warning, "不是彩图", '请选择彩图进行风格迁移
    ')
            msg_box.exec_()
            init(self)
            return
        else:
            self.unit6_img2_channel = 3
            if self.unit6_img2.shape[2] == 4:
                self.unit6_img2 = cv2.cvtColor(self.unit6_img2, cv2.COLOR_BGRA2BGR)
        print(self.unit6_img1.shape)
        img_refresh(self)

    def img_refresh(self):#图像刷新
        array = \
            [self.unit6_img1,
             self.unit6_img2,
             self.unit6_result]

        array2 = [self.ui.label_44,
                  self.ui.label_45,
                  self.ui.label_47
                 ]

        channel = [self.unit6_img1_channel,
                   self.unit6_img2_channel,
                   self.unit6_result_channel]
        height = 350
        weight = 350
        for index in range(len(array)):
            M = np.float32([[1, 0, 0], [0, 1, 0]])
            if array[index].size <= 1:
                array2[index].setPixmap(QtGui.QPixmap(''))
                continue
            print(array[index].shape)
            index_h = array[index].shape[0]
            index_w = array[index].shape[1]
            if index_h / index_w == height / weight:
                img = array[index].tobytes()
                if channel[index] == 1:
                    image = QtGui.QImage(img, index_w, index_h, index_w *
    channel[index], QtGui.QImage.Format_Grayscale8)
                    pix = QtGui.QPixmap.fromImage(image)
                    scale_pix = pix.scaled(weight, height)
                    array2[index].setPixmap(scale_pix)
```

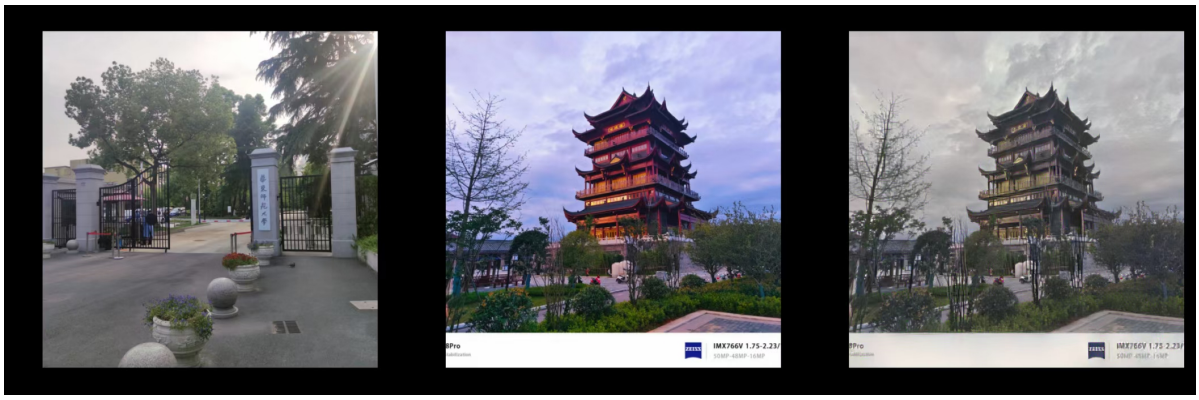
```
                continue
            elif channel[index] == 3:
                image = QtGui.QImage(img, index_w, index_h, index_w *
channel[index], QtGui.QImage.Format_BGR888)
                pix = QtGui.QPixmap.fromImage(image)
                scale_pix = pix.scaled(weight, height)
                array2[index].setPixmap(scale_pix)
                continue
        elif index_h / index_w > height / weight:
            h_ = index_h
            w_ = int(index_h * weight / height + 0.5)
            M[0, 2] += (w_ - index_w) / 2
            M[1, 2] += (h_ - index_h) / 2
        else:
            h_ = int(index_w * height / weight + 0.5)
            w_ = index_w
            M[0, 2] += (w_ - index_w) / 2
            M[1, 2] += (h_ - index_h) / 2
        img = cv2.warpAffine(array[index], M, (w_, h_))
        data = img.tobytes()
        if channel[index] == 1:
            image = QtGui.QImage(data, w_, h_, w_ * channel[index],
QtGui.QImage.Format_Grayscale8)
            pix = QtGui.QPixmap.fromImage(image)
            scale_pix = pix.scaled(weight, height)
            array2[index].setPixmap(scale_pix)
            continue
        else:
            image = QtGui.QImage(data, w_, h_, w_ * channel[index],
QtGui.QImage.Format_BGR888)
            pix = QtGui.QPixmap.fromImage(image)
            scale_pix = pix.scaled(weight, height)
            array2[index].setPixmap(scale_pix)
            continue
    return

def gram_matrix(y):#计算Gramma矩阵，为tensor类型
    (b, ch, h, w) = y.size()
    features = y.view(b, ch, w * h)
    features_t = features.transpose(1, 2)
    gram = features.bmm(features_t) / (ch * h * w)
    return gram

def style_transfer(self):#风格迁移
    if self.unit6_img1.size>1 or self.unit6_img2.size>1:
        style_img = read_image(self.filepath2, target_width=512).to(device)#读入图
像，做归一化、张量化，调整大小
        print(torch.cuda.is_available())
        content_img = read_image(self.filepath1, target_width=512).to(device)
        print(style_img.shape)
        print(content_img.shape)
        vgg16 = models.vgg16(pretrained=True)#导入VGG
        vgg16 = VGG(vgg16.features[:23]).to(device).eval()#导入VGG对应层输出的接口函数
        style_features = vgg16(style_img)#导出风格特征
        content_features = vgg16(content_img)#导出内容特征
        style_grams = [gram_matrix(x) for x in style_features]#计算gramma矩阵，实际上
就是计算内积
        input_img = content_img.clone()
        optimizer = optim.LBFGS([input_img.requires_grad_()])#优化器
```

```
        style_weight = 1e6
        content_weight = 1
        run = [0]
        while run[0] <= 300:
            QApplication.processEvents()
            self.ui.progressBar.setValue(int(run[0]/3))#迭代过程中更新进度条
            # if(run[0]%3==0):
            # self.signal.emit(int(run[0]/3))
            def f():
                optimizer.zero_grad()
                features = vgg16(input_img)
                content_loss = F.mse_loss(features[2], content_features[2]) *
content_weight#计算均方损失，按照之前写的公式计算
                style_loss = 0
                grams = [gram_matrix(x) for x in features]
                for a, b in zip(grams, style_grams):
                    style_loss += F.mse_loss(a, b) * style_weight#累加一下总损失
                loss = style_loss + content_loss
                if run[0] % 50 == 0:
                    print('Step {}: Style Loss: {:4f} Content Loss: {:4f}'.format(
                        run[0], style_loss.item(), content_loss.item()))
                run[0] += 1
                loss.backward()
                return loss

            optimizer.step(f)
        self.unit6_result = util.recover_image(input_img)
        img_refresh(self)
        print("Train over!")
      #  thread = Runthread()
      #  thread.setParam(self.filepath1, self.filepath2)
       # print("Yes0")
       # try:
      #   thread.signal.connect(self.progressBar_refresh)
      #  except:
       #   print("Yes1")
      #  thread.start()
      #  print("Yes2")
      #  thread.wait()
     #   print("Yes3")
    else:
        msg_box = QMessageBox(QMessageBox.Warning, '无需清空', '没有图片')
        msg_box.exec_()


#def progressBar_refresh(self, msg):
  #  self.ui.progressBar.setValue(int(msg))



def img_save(self):#保存结果
    if self.unit6_result.size > 1:
        fileName, tmp = QFileDialog.getSaveFileName(self, '保存图像', 'Image',
'*.png *.jpg *.bmp *.jpeg')
        if fileName == '':
            return
        cv2.imwrite(fileName, self.unit6_result)
        msg_box = QMessageBox(QMessageBox.Information, '成功', '图像保存成功,保存路径
为: ' + fileName)
```

```
            msg_box.exec_()
        else:
            msg_box = QMessageBox(QMessageBox.Warning, '提示', '没有生成图像')
            msg_box.exec_()



def img_clear(self):#图像清除
    if self.unit6_img1.size > 1 or self.unit6_img2.size > 1:
        init(self)
        img_refresh(self)

    else:
        msg_box = QMessageBox(QMessageBox.Warning, '无需清空', '没有图片')
        msg_box.exec_()
```

毋庸置疑的，在实践中，需要先对图像（由PIL lib中的Image.open(src)读入）进行归一化并转化成tensor（torch），再转化为cuda进行训练。实测对于300*400*的风格图，*1080*720的待迁移图而言，其在性能较强的计算卡上训练训练时间大约在30s-50s（epoch=300）（GTX 2060 6G），在性能较弱的计算卡（GTX1650）上需要计算1-1.5分钟左右。如果用cpu训练，则需要超过3分钟时间。因此此部分不推荐没有配置cuda的用户使用。其他内容受限于篇幅，不在展开。

如下两张图是我的实验结果展示：



3.2.2.yolo报告：

yolo系列作为one stage的开山之作，其运行速度快、泛化能力强。相比RCNN系列先选出候选框，再识别对象的方法，yolo系列选择了端到端的方式，将检测问题变成了回归问题，拥有非常高的准确性。考虑到yolo系列的内容较多，这里只简单介绍一下其训练过程：

（这里我只看了yolov1-v3的论文，实践部分使用的是yolov5）

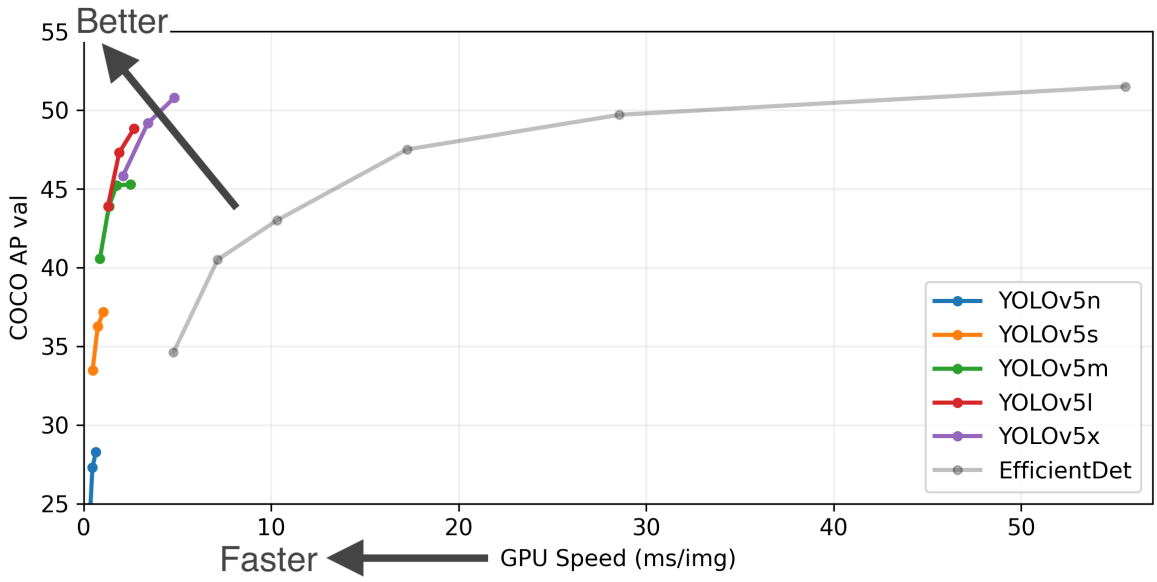
yolov1的训练过程大致如下：

1. 将图片resize，调整成$448*448$，分割成$7*7$的Cells
2. 和传统神经网络架构类似，用CNN提供感受野，切割图片寻找特征，FC负责分类（交叉熵）
3. 在李飞飞教授创建的经典数据集ImageNet上面运行，由于有20个label，加上两个bounding box for each cells（包含了x，y，h，w）以及两个bbox的IOU置信度，组成一个30维的结果向量。其损失函数如下：

$$L=\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(x_i-\hat{x}_i\right)^2+\left(y_i-\hat{y}_i\right)^2+\lambda_{coord}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2$$

$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_I)^2+\sum_{i=0}^{s^2}\mathbb{1}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2$$

这里我选择了最新的yolov5系列。yolov5有多个版本，包括yolov5s，yolov5m……其各版本性能差异如下图所示：



由于我的设备是RTX2060（6G），并已安装好cuda11.1。在该项目中，还需要提前安装Pytorch和labelme，后者在制作数据集时使用。

1.制作数据集：

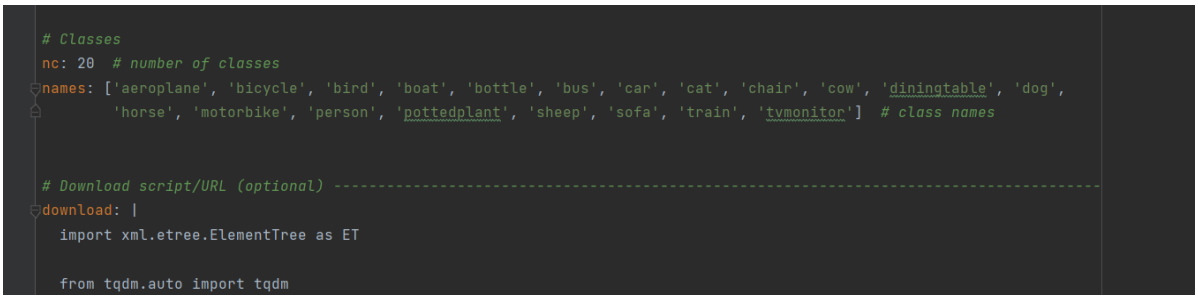
由于我的想法是制作一个效果还不错的性别分类数据集，但手上拥有的数据集较少，因此我的目标主要放在了小型规模的数据集上，并以此希望在泛化条件下拥有还过得去的准确率。这里我选择了我高中的人像数据集和部分补充图片，共计304张。切割成训练集：测试集=4：1的比例，并使用labelme手动打上标签：

名称	修改日期	类型	大小
1650702317432	23/4/2022 下午9:53	JSON File	215 KB
1650702317444	23/4/2022 下午9:55	JSON File	152 KB
1650702317453	23/4/2022 下午9:55	JSON File	159 KB
1650702317464	23/4/2022 下午10:14	JSON File	154 KB
1650702317476	23/4/2022 下午9:55	JSON File	222 KB
1650702317486	23/4/2022 下午9:55	JSON File	212 KB
1650702317497	23/4/2022 下午9:56	JSON File	181 KB
1650702317509	23/4/2022 下午9:56	JSON File	176 KB
1650702317519	23/4/2022 下午9:56	JSON File	167 KB
1650702317531	23/4/2022 下午9:57	JSON File	176 KB
1650702317543	23/4/2022 下午9:57	JSON File	180 KB
1650702317558	23/4/2022 下午9:57	JSON File	169 KB
1650702317575	23/4/2022 下午9:57	JSON File	161 KB
1650702317588	23/4/2022 下午9:58	JSON File	140 KB
1650702317601	23/4/2022 下午9:58	JSON File	149 KB
1650702317615	23/4/2022 下午9:58	JSON File	148 KB
1650702317630	23/4/2022 下午9:58	JSON File	165 KB
1650702317645	23/4/2022 下午9:58	JSON File	155 KB
1650702317659	23/4/2022 下午9:59	JSON File	111 KB
1650702317676	23/4/2022 下午9:59	JSON File	126 KB
1650702317689	23/4/2022 下午9:59	JSON File	115 KB
1650702317699	23/4/2022 下午9:59	JSON File	141 KB
1650702317709	23/4/2022 下午9:59	JSON File	198 KB

由于labelme标注的标签格式是json格式，而项目需要的是yolo的txt格式，因此我使用了labelme2txt.py脚本，将其转化为txt格式。转化后生成文件如下：

名称	修改日期	类型	大小
1650702317432	24/4/2022 下午10:41	文本文档	1 KB
1650702317444	24/4/2022 下午10:41	文本文档	1 KB
1650702317453	24/4/2022 下午10:41	文本文档	1 KB
1650702317464	24/4/2022 下午10:41	文本文档	1 KB
1650702317476	24/4/2022 下午10:41	文本文档	1 KB
1650702317486	24/4/2022 下午10:41	文本文档	1 KB
1650702317497	24/4/2022 下午10:41	文本文档	1 KB
1650702317509	24/4/2022 下午10:41	文本文档	1 KB
1650702317519	24/4/2022 下午10:41	文本文档	1 KB
1650702317531	24/4/2022 下午10:41	文本文档	1 KB
1650702317543	24/4/2022 下午10:41	文本文档	1 KB
1650702317558	24/4/2022 下午10:41	文本文档	1 KB
1650702317575	24/4/2022 下午10:41	文本文档	1 KB
1650702317588	24/4/2022 下午10:41	文本文档	1 KB
1650702317601	24/4/2022 下午10:41	文本文档	1 KB
1650702317615	24/4/2022 下午10:41	文本文档	1 KB
1650702317630	24/4/2022 下午10:41	文本文档	1 KB
1650702317645	24/4/2022 下午10:41	文本文档	1 KB
1650702317659	24/4/2022 下午10:41	文本文档	1 KB
1650702317676	24/4/2022 下午10:41	文本文档	1 KB
1650702317689	24/4/2022 下午10:41	文本文档	1 KB
1650702317699	24/4/2022 下午10:41	文本文档	1 KB
1650702317709	24/4/2022 下午10:41	文本文档	1 KB

接下来需要修改clone下来的yolov5系列内所有类别配置文件，这里我使用的是yolov5s，因此复制一份yolov5.yaml和voc.yaml，并修改其内部的类别参数。



```yaml
# Classes
nc: 20  # number of classes
names: ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair', 'cow', 'diningtable', 'dog',
        'horse', 'motorbike', 'person', 'pottedplant', 'sheep', 'sofa', 'train', 'tvmonitor']  # class names


# Download script/URL (optional) ---------------------------------------------------------------------------------------
download: |
  import xml.etree.ElementTree as ET

  from tqdm.auto import tqdm
```

此后，基本数据和参数就配置好了。接下来我写了一个脚本dataset.py用于切割数据集：

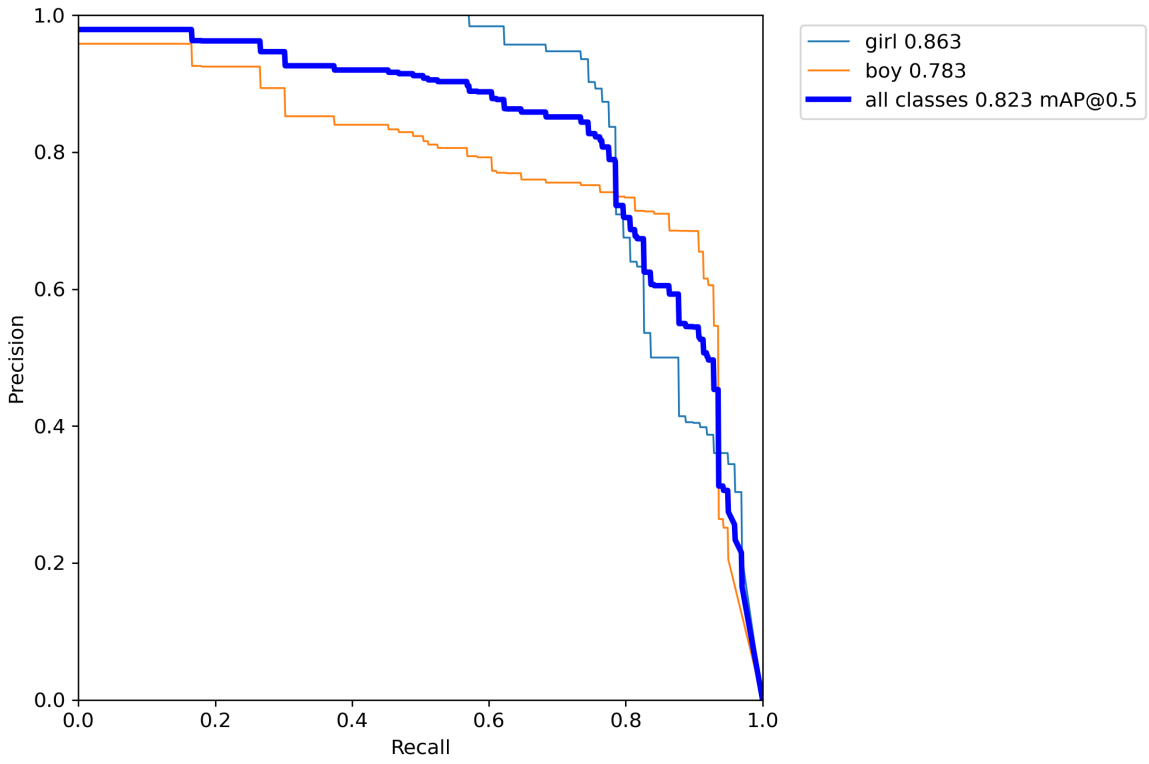
```python
import os
from shutil import copy
import random


def mkfile(file):
    if not os.path.exists(file):
        os.makedirs(file)
#os.path.exists(file)检测file路径是否有文件或者目录，如果没有，就在file路径创建一个空文件夹

def testjpg(image):
    if (image[-3:] == 'jpg'):
        s_image = image[:-4]
    else:
        s_image = image[:-5]
    return s_image
file_path = 'mydata'
list1 = ['images', 'labels']
for cla in list1:
    mkfile('./mydata/'+ cla + '/train')
    mkfile('./mydata/'+ cla + '/val')

split_rate = 0.2 #划分比例
path1 = file_path + '/Yoloimages/'
path2 = file_path + '/Yololabels/'
images = os.listdir(path1)
num = len(images)
eval_index = random.sample(images, k=int(num * split_rate)) #随机截取images中k个元素组成新列表
for index, image in enumerate(images):#index是默认变量，代表自增索引
        if image in eval_index:
            image_path = path1 + image
            s_image = testjpg(image)
            text_path = path2 + s_image + '.txt'
            new_path1 = 'mydata/images/val/'
            new_path2 = 'mydata/labels/val/'
            copy(image_path, new_path1)
            copy(text_path, new_path2) #从老路径copy到新路径
        else:
            image_path = path1 + image
            s_image = testjpg(image)
            text_path = path2 + s_image + '.txt'
            new_path1 = 'mydata/images/train/'
            new_path2 = 'mydata/labels/train/'
            copy(image_path, new_path1)
```

```
            copy(text_path, new_path2)
print('Processing files over')
```

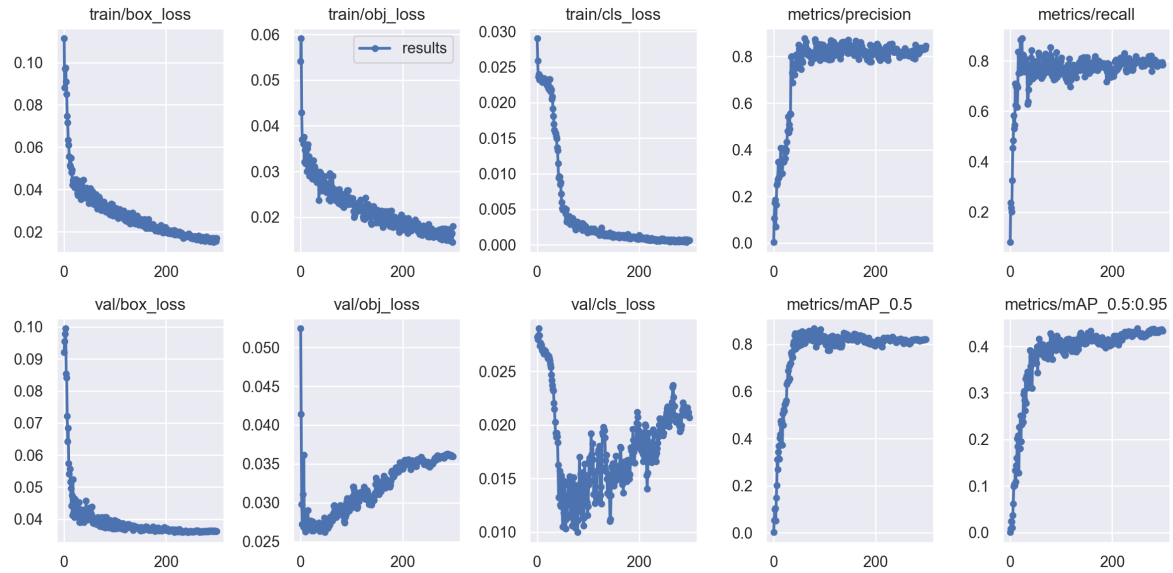
2.训练阶段：

```
def parse_opt(known=False):
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='models/myWeight/yolov5s-test.yaml', help='model.yaml path')
    parser.add_argument('--data', type=str, default='mydata/mydata.yaml', help='dataset.yaml path')
    parser.add_argument('--hyp', type=str, default='data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=120)
    parser.add_argument('--batch-size', type=int, default=20, help='total batch size for all GPUs, -1 for autobatch')
    parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--noval', action='store_true', help='only validate final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
    parser.add_argument('--noplots', action='store_true', help='save no plot files')
    parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--workers', type=int, default=0, help='max dataloader workers (per RANK in DDP mode)')
    parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
```

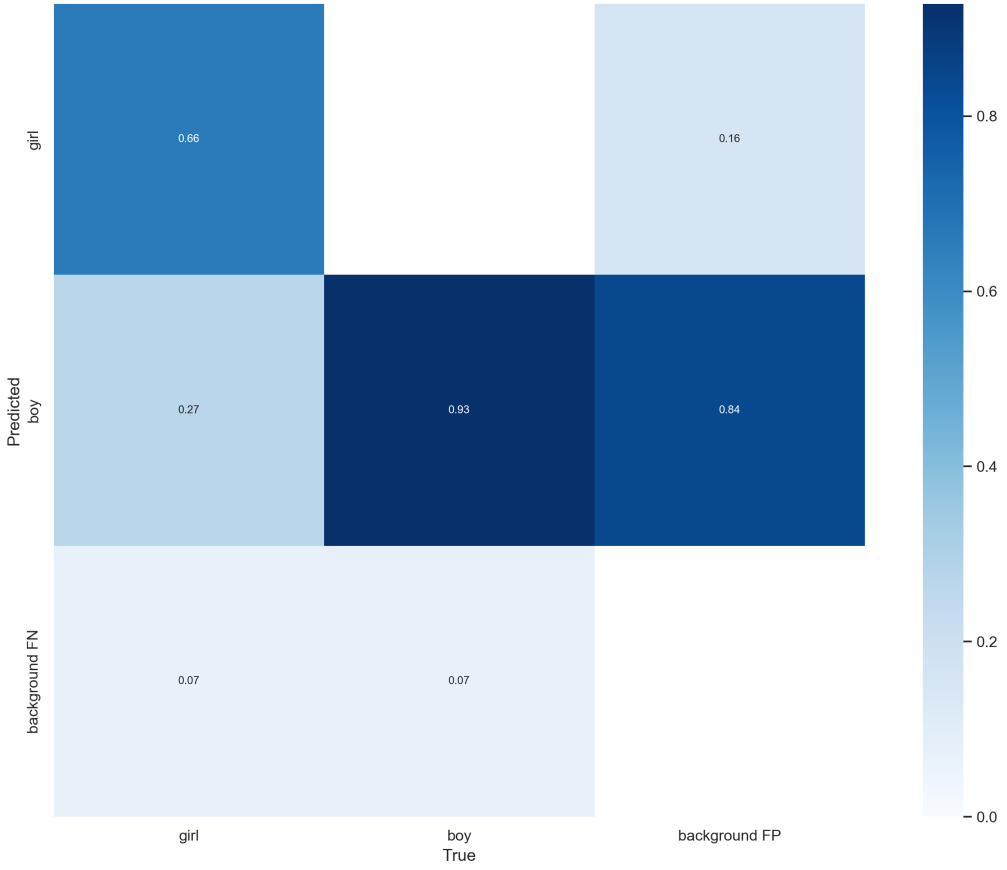
只需要调整参数、调整数据集即可！经过测试，在worker=1，batchsize=20的情况下能够不超显存（6G），训练300轮大约需要5-6小时时间。经过多次训练、调参、数据集做图像处理后，得到的最优训练情况如下：（以下皆为tensorboard记录）



PR曲线显示，整体的召回率与精确率在0.7-0.8附近。



训练轮数表明，大约在100轮左右，网络已经基本达到过拟合，这里我们最终选用的最优预训练权重也大致是训练了100轮的结果。



混淆矩阵如图所示。

接下来，在已经得到预训练权重的情况下，我将预测模块移植进我们的数字图像处理项目中，实现了最终的功能：

代码主体如下所示：

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

from PIL import Image
import matplotlib.pyplot as plt

import torchvision.transforms as transforms
import torchvision.models as models
from PyQt5 import QtGui
from PyQt5.QtWidgets import *
import cv2
import numpy as np
from PyQt5 import QtWidgets, QtCore
import sys
from PyQt5.QtCore import *
import time

import utils
from utils import *
from models import *
import detect
import utils.general
from pathlib import Path

def init(self):
    self.unit7_img = np.ndarray(())
    self.unit7_img_channel = 1
    self.unit7_result = np.ndarray(())
    self.unit7_result_channel = 1
```

```
    self.unit7_filepath = ''
    self.unit7_imgpath = ''
    self.unit7_savepath = ''
    self.unit7_suffix = ''
    self.ui.textBrowser_6.setText('')
    self.ui.textBrowser_7.setText('')


def img_load(self):#图像导入
    fileName, tmp = QFileDialog.getOpenFileName(self, '打开图像', 'Image', '*.png
*.jpg *.bmp *.jpeg')
    if fileName == '':
        return
    self.unit7_img = np.ndarray(())
    self.unit7_img_channel = 1
    self.unit7_result = np.ndarray(())
    self.unit7_result_channel = 1
    self.unit7_img = cv2.imread(fileName, -1)
    self.unit7_suffix = fileName.split('/')[-1]
    print(self.unit7_suffix)
    self.unit7_imgpath = fileName
    if self.unit7_img.size <= 1:
        return
    if len(self.unit7_img.shape) == 3:
        self.unit7_img_channel = 3
        if self.unit7_img.shape[2] == 4:
            self.unit7_img = cv2.cvtColor(self.unit7_img, cv2.COLOR_BGRA2BGR)
    print(self.unit7_img.shape)
    img_refresh(self)

def img_refresh(self):#图像刷新
    array = \
        [self.unit7_img,
         self.unit7_result]

    array2 = [self.ui.label_54,
              self.ui.label_55]

    channel = [self.unit7_img_channel,
               self.unit7_result_channel]
    height = 480
    weight = 500
    for index in range(len(array)):
        M = np.float32([[1, 0, 0], [0, 1, 0]])
        if array[index].size <= 1:
            array2[index].setPixmap(QtGui.QPixmap(''))
            continue
        print(array[index].shape)
        index_h = array[index].shape[0]
        index_w = array[index].shape[1]
        if index_h / index_w == height / weight:
            img = array[index].tobytes()
            if channel[index] == 1:
                image = QtGui.QImage(img, index_w, index_h, index_w *
channel[index], QtGui.QImage.Format_Grayscale8)
                pix = QtGui.QPixmap.fromImage(image)
                scale_pix = pix.scaled(weight, height)
                array2[index].setPixmap(scale_pix)
                continue
```

```
            elif channel[index] == 3:
                image = QtGui.QImage(img, index_w, index_h, index_w *
channel[index], QtGui.QImage.Format_BGR888)
                pix = QtGui.QPixmap.fromImage(image)
                scale_pix = pix.scaled(weight, height)
                array2[index].setPixmap(scale_pix)
                continue
        elif index_h / index_w > height / weight:
            h_ = index_h
            w_ = int(index_h * weight / height + 0.5)
            M[0, 2] += (w_ - index_w) / 2
            M[1, 2] += (h_ - index_h) / 2
        else:
            h_ = int(index_w * height / weight + 0.5)
            w_ = index_w
            M[0, 2] += (w_ - index_w) / 2
            M[1, 2] += (h_ - index_h) / 2
        img = cv2.warpAffine(array[index], M, (w_, h_))
        data = img.tobytes()
        if channel[index] == 1:
            image = QtGui.QImage(data, w_, h_, w_ * channel[index],
QtGui.QImage.Format_Grayscale8)
            pix = QtGui.QPixmap.fromImage(image)
            scale_pix = pix.scaled(weight, height)
            array2[index].setPixmap(scale_pix)
            continue
        else:
            image = QtGui.QImage(data, w_, h_, w_ * channel[index],
QtGui.QImage.Format_BGR888)
            pix = QtGui.QPixmap.fromImage(image)
            scale_pix = pix.scaled(weight, height)
            array2[index].setPixmap(scale_pix)
            continue
    return


def result_save(self):#图像保存
    fileName= QFileDialog.getExistingDirectory(self, '保存图像')
    if fileName == '':
            return
    self.unit7_savepath = fileName
    self.ui.textBrowser_7.setText(fileName.split('/')[-2]+'/'+fileName.split('/')
[-1])
    msg_box = QMessageBox(QMessageBox.Information, '成功', '选择路径成功,保存路径为：'
+ fileName)
    msg_box.exec_()

def clear(self):
    if self.unit7_img.size > 1:
        init(self)
        img_refresh(self)
    else:
        msg_box = QMessageBox(QMessageBox.Warning, '无需清空', '没有图片')
        msg_box.exec_()


def result_show(self):#图像查看
    if self.unit7_result.size > 1:
        cv2.imshow('Original pic', self.unit7_result)
```

```
        cv2.waitKey(0)
    else:
        msg_box = QMessageBox(QMessageBox.Warning, '没有图像', '没有生成图像')
        msg_box.exec_()


def object_detection(self):#调用detect.py，进行目标检测
    if self.unit7_filepath !='' and self.unit7_img.size>1 and
self.unit7_savepath!='':
        modelpath = self.unit7_filepath
        imgpath = self.unit7_imgpath
        savepath = self.unit7_savepath
        detect.main(imgpath, modelpath,savepath)
        name ='exp'
        z = utils.general.increment_path_num(Path(savepath) / name,
exist_ok=False)
        num = str(z) if z!=1 else ''
        path = savepath +'/exp'+ num+'/'+self.unit7_suffix
        print(path)
        self.unit7_result = cv2.imread(path, -1)
        if self.unit7_result.size >1:
           if len(self.unit7_result.shape) == 3:
               self.unit7_result_channel = 3
               if self.unit7_result.shape[2] == 4:
                   self.unit7_result = cv2.cvtColor(self.unit7_result,
cv2.COLOR_BGRA2BGR)
           print(self.unit7_result.shape)
           img_refresh(self)
        else:
            msg_box = QMessageBox(QMessageBox.Warning, 'error', 'error1')
            msg_box.exec_()
    else:
        msg_box = QMessageBox(QMessageBox.Warning, '没有导入模型或图片', '请导入模型和图
片后再进行尝试')
        msg_box.exec_()




def model_load(self):#导入预训练权重
    fileName, tmp = QFileDialog.getOpenFileName(self, '选择模型路径', 'Model',
'*.pt')
    if fileName == '':
        return
    self.unit7_filepath = fileName
    self.ui.textBrowser_6.setText(fileName.split('/')[-2]+'/'+fileName.split('/')
[-1])
    print(self.unit7_filepath)
    if self.unit7_filepath =='':
        return
    else:
        msg_box = QMessageBox(QMessageBox.Information, '已检测到模型', '模型导入成功')
        msg_box.exec_()
```

4.结语与思考

经过本次长达三个月的项目搭建，我增进了许多的理论和实践知识。理论知识主要为传统数字图像处理功能和计算机视觉方面，前者源自于曹老师的课程，后者则源自于自己课外学习的课程和论文；实践知识则更加丰富，从PYQT的开发到PYINSTALLER打包、从conda虚拟环境的配置到GIT仓库高级指令的运用，从需求分析到软件测试，这些实践经历无不增强了我的软件工程开发意识和能力。受限于时间与能力问题，有许多功能未必完美，例如在风格迁移方面，由于网络训练时间长达半分钟以上，直接导致无法实时处理视频，那么如何优化这个问题？男女性别差异极小，光靠yolo传统架构无法直接one-stage高精确的完成目标检测与性别判断两个任务，能否修改架构，将目标检测任务交予yolo，而将目标切割出来的图像交给诸如Alexnet这样的分类神经网络？由于是单人小组，项目还有许多可以扩展的内容还未完成，这也说明了我还有许多方面需要学习与进步。

参考文献：

1. http://arxiv.org/abs/1508.06576
2. https://github.com/ultralytics/yolov5