基于 YOLO V3 发动机表面缺陷检测的研究

摘 要

本文研究了 YOLO v3 在发动机表面缺陷检测中的表现。作为深度学习检测缺陷的启发，本文首先研究了实际生产中缺陷检测的方式及其优缺点，然后研究了深度学习在理论上发展，并详细探讨了将最先进的算法应用在发动机缺陷检测上的具体办法。在分析了基础的 YOLO 网络结构和算法实现原理之后，本文继续分析了发动机表面缺陷的具体特征，总结出了这个实际生产问题中缺陷目标与一般目标检测算法中目标的巨大区别，并针对性地提出了解决方案以 YOLO 算法最大化地适配到缺陷检测中。接下来，本文讨论了定制表面缺陷数据集的方法，为了将收集好的，有限的原始图像数据处理成可以用于 YOLO 网络训练的足量的 YOLO 专用数据集，本文对原始数据采取了数据增强，数据标注和数据集划分等操作。然后本文进行了实验，在特定的编程环境下，将原始的 YOLO v3 训练算法加以参数上的调整，最终调试出了最适用于缺陷数据集的训练模型，并且实现了实际检测测试集图片的高准确度。最后，利用优化过的模型，以及相对应的检测程序，实验将所有的运行环境打包起来一起封装一个使用的 windows 应用程序中，实现了缺陷检测的简单化，便利化。

综上，基于 YOLO v3 的发动机表面缺陷算法最终成功地训练出了检测模型，不仅证明 YOLO v3 算法的确可以应用到缺陷检测中，还证明了该算法实现的检测精度高，性能好，可以普及使用。

关键词：YOLO v3 算法，发动机，缺陷检测，深度学习

装 订 线

Research on Surface Defect Detection of Engine Based on YOLO V3

ABSTRACT

This paper studies the performance of YOLO v3 in the detection of engine surface defection. As the inspiration for deep learning defect detection, this paper first studies the method of defect detection in actual production and its advantages and disadvantages, then studies the theoretical development of deep learning, and discusses in detail the application of the most advanced algorithms in engine defect detection specific methods. After analyzing the basic YOLO network structure and algorithm implementation principles, this article continues to analyze the specific characteristics of engine surface defects. Summing up the huge difference between the defect target in this actual production problem and the target in the general target detection algorithm, this paper comes out a solution to maximize the adaptation of YOLO algorithm to defect detection. Next, this paper discusses the method of customizing the surface defect data set. In order to process the collected but limited original image data set into a sufficient amount of YOLO dedicated data set that can be used for YOLO network training, this article takes data enhancement on the original data , as well as data labeling and data set partitioning. Then this paper conducts experiments to adjust the parameters of the original YOLO v3 training algorithm under a specific programming environment, and finally debugs the training model that is most suitable for the defect data set, and realizes the high precision of the actual detection of test set. Finally, using the optimized model and the corresponding detection program, the experiment packags all the operating environments together and packags it into a windows application, which realizs the simplification and convenience of defect detection.

In summary, the engine surface defect algorithm based on YOLO v3 has been successfully put into practical use. It not only proves that the YOLO v3 algorithm can indeed be applied to defect detection, but also proves that the algorithm achieves high detection accuracy, good performance, and can be widely used.

Key words: YOLO v3 algorithm, engine, defect detection, deep learning

装 订 线

目 录

装 订 线

装

订

线

1 绪论

1.1 研究背景和意义

发动机是汽车和其他大型动力机械的动力发生装置，可以说是机械的心脏，如果它的表面因为种种原因（生产缺陷，使用磨损等）产生缺陷，则会带来诸多不良隐患，造成严重后果。即使是划痕，铸造沙眼这样的看似微小的瑕疵，也会影响发动机缸体整体的强度；由此还会造成缸体结构变形，使零件之间的连接松动。发动机表面的缺陷不仅会干扰使用者的主观判断，还会破坏发动机设计时期望的功能完整性。

发动机缸体通常采用铸造的方式生产，大型铸件的生产过程有很多繁琐的环节，每一步都需要精密的操作。首先是金属融化，要使用冶炼设备加工毛胚金属，然后采用压射、吸入或者浇注的方式把液态熔融的金属注入铸体中，接下来经过压制和冷凝等过程塑造成型，最后是施以打磨等后处理将其加工到设计好的固定尺寸。打磨过程，现阶段主要采用人工打磨的方式，人工打磨对技工技术水平要求极高，如果出现微小的失误，比如打磨不足或者打磨过度，会直接导致零件的凸起或是凹陷。除了工人技术因素需要重点考虑，金属自身的质量和设备的精度同样需要严格的把关：前者的不足会使内部结构塌陷损坏，导致暗孔的生成，后者的不足会降低模具的精度直接影响成品质量。除此之外，铸体相互碰撞产生的裂纹，熔融液飞溅产生的沙眼等同样会对发动机表面质量造成不良影响。上述的任何一个环节出现纰漏，就会导致表面含有缺陷的发动机被生产出来。

现今主流的发动机表面缺陷检测可分为人工检测和自动化检测两种。在当今这个自动化生产逐渐成为趋势的新时代，人工检测因其耗费人力成本高，对专业人员资质要求严苛，肉眼观测个体判断存在差距难以统一标准，工人难以适应越来越快流水线速度等因素越来越多地被先进流水线所淘汰。传统的无损检测大致有超声波检测、X 光检测、红外线检测、漏磁检测、涡流检测等，这些方法以缺陷的物理特征为判断的标准，由此让检测条件变得十分局限，检测结果与实际可能有较大差距，像涡流检测这类检测方法还要求检测设备与被测件直接接触，这无疑增加铸件表面被损坏的概率。

近年来，计算机机器视觉技术逐渐变得成熟，主流研究中的目标识别技术被越来越多地应用在了工程领域，针对如发动机缸体这类的大型铸件的生产过程中也有 Gabor 变换检测法和 PCB 检测法这样的方法。它们的核心思想都是提取表面图像的灰度差异特征后对这些特征进行比对，然后通过分析相似度或者判断有无核心特征的方法作出结果预测。另外一些的检测方法还借鉴了机器学习的部分思想，设计了以特征提取器和分类器两部分为核心的算法，在检测思路上有极大的突破。但这类新方法依然有不足之处，其一，特征提取器的设计上极大依赖专业人员的个人经验，并且难以考虑到所有细节，由此在真实的测试环境下易受到外界环境不可预知的干扰；其二，设计特征提取器和定义缺陷特征需要人工操作步骤多，不能保证特征精度，在此基础上设计出的算法程序鲁棒性差。

1.2 国内外研究现状

1.2.1 缺陷检测研究现状

目标的各种物理性质是最常运用到传统的发动机表面缺陷检测的，方法设计的基本思想是尽量的零接触，在不意外伤害表面材质和内部结构的情况下对可能的缺陷进行识别。具体到电、磁、声、光四个主要的方向，传统检测方法有涡流检测法、漏磁检测法、超声波检测法、红外线检测法，另外还有振动分析检测法等等。针对缺陷检测技术的研发和应用，国外开始较早，最早的应用实践甚至可以追溯到 1970 年前后。1971 年，英国钢铁公司就和 London Metropolitan 大学展开过合作，共同开发了钢板表面缺陷检测系统[1]，当时他们选择的是激光扫描技术，即以氦氖激光扫描器作为光源，应用光线反射的原理透过旋转的棱镜将激光照射到钢板表面。为了将光信号转换成电信号输入计算机分析，还采用了光电倍增管的技术，当然这也不可避免地造成了本就不太精确的信息的进一步损失，然而，这项研究在当年依然获得了不小的认可。另外有美国高格斯公司也曾独立研发过缺陷检测系统，号称“智能检视”，原理依然是通过光照射来提取物体表面的图像及其特征信息，对特征进行进一步分析来诊断缺陷[2]。类似的依靠光信息的检测案例还有不少，他们共同的缺陷是人工操作的过于依赖和预处理的过于复杂，虽然理想状态下检测精确率很高，但是在实际应用中由于各种环境影响和人为影响，精确率都不理想。

计算机视觉技术作为计算机深度学习的前沿技术，在纯理论不断精进的同时，在类似于缺陷检测这样的实际应用领域同样开始大放异彩，它的高检测速度、便于实现这些特点被研究人员所青睐。具体来说，缺陷检测利用计算机视觉技术大致分为了两个大的步骤，一为提取特征，二为分类所提取的特征。这里的特征不仅指材质特征和颜色特征这样的二维特征，也可以是结构特征这样的三维特征在图像上的表现。所谓分类即是将提取的各类特征作为参考对缺陷类别进行判别。目前主流的检测方法有三种：

1）投影检测法。其核心思想是剖析出目标的本质特征，然后依照物体特征的相关性建模，再将特征映射到另一个设计的空间。在这种思想指导下，B Yousef 等提出了主成分分析法(PCA)[3]。另外有姚忠伟等人针对大尺寸缺陷以及细小缺陷的二值图像[4]，研发出了利用计算机视觉的 PCB 缺陷检测新技术，检测精确率十分可观。

2）滤波器检测法。其核心思想是通过滤化处理，让滤波器处理图像。导入滤波器的图像是已经进行过频域变换后的原始图像。这样的好处是相关性低的特征以及背景噪点会被滤去，使算法表现出更高的鲁棒性。

3）学习检测法。其核心思想顾名思义，希望通过对样本的学习自动总结出缺陷特征，避开充满变数且费时费力的人工提取特征的环节。最著名的算法即是深度神经网络（DNNs）[5-6]，包括本文研究的 YOLO 算法，还有 Faster r-CNN 等目标识别算法都建立在它的基础之上，无论是精度还是性能都是在现有算法里首屈一指的。

最近一两年中，学习检测法逐渐成了缺陷检测的新潮流。将卷积神经网络可以对模型进行优化这一长处和实际检测中海量的新数据输入相结合，深度学习的缺陷检测性能逐渐强大起来。相较于 CPU 设备，当今研究人员更加青睐处理能力强大的 GPU 设备，在先进硬件的帮助下，基于深度学习的缺陷检测已经应用到了各种工程领域。

1.2.2 深度学习研究现状

上世纪六十年代，生物学家在研究猫视神经系统时，发现在某些领域猫的大脑皮层非常敏感，之后他们便提出了 Receptive Field[7]，即感受域的概念。建立在他们研究的基础之上，日本学者福岛在八十年代仿造了大脑皮层视觉系统的结构研究出了 Neocognition 网络[8]，这就是神经网路的前身。深度学习是机器学习领域中最杰出的成果，尤以卷积神经网络[9]最为出名。第一个真正意义上的卷积神经网络模型是 L Yann 等人提出的 Le Net[10-11]，他们最先引入了 pooling layer 和 fully connected layer 的概念，从此池化和全连接成为构建高性能卷积网络难以绕开的优化方法。而深度学习这一概念，则是由 Geoffrey Hinton 教授与其团队[12]在 2006 年提出，他们首先推出基于神经网络的降维学习的研究，他们的这一指导性思想一直被广大学者沿用至今。

2009 年, Image Net 数据集由 F Lee 与其团队[13]推出，并于第二年发起了 ILSVRC 计算机视觉识别与分类挑战赛。Alex Net[14]获得了 2009 年 ILSVRC 的冠军，令人震惊的是，这个网络的 TOP-5 误差率仅为 15.4%，世界各地的计算机行业从业者都为这优秀的数据所吸引，由此深度学习成为了研究新风向直至今日。诸如脸书、谷歌、微软、IBM 这些计算机行业领头人，都成立了专门研究深度学习技术的实验室。谷歌公司 AI 团队在 2014 年[15]提出了 Goog Le Net——一个全新的网络结构设计，它在 Image Net 数据集的 Top-5 错误率仅为 6.67%。1x1 的卷积方式是他们的网络的特色，这样的卷积扩大感受域。Oxford 的计算机视觉研究团队在同年推出了 VGG-Net[16]的深层网络结构。进行特征提取和压缩操作的是尺寸仅 3x3 的卷积核，并由此进行了最大池化操作，使这个网络加深到了 19 层，在该数据集 Top-5 准确率也达到了到 7.5%。VGG-Net 网络结构最值得称道的是他的可塑性之强。他同时也让大家明白了，更细致的卷积网络设计可以明显提高卷积神经网络的识别效率。2015 年底 K He 的团队提出了将深度残差网络的概念[17-18]，Short Cut 连接是 Res Net 别出心裁的连接方式。这个连接的优点是，应用了它的卷积神经网络可以加深更多的层数而不用担心梯度消失，由此卷积神经网络的最高层数达到了 152 层，它的识别性能无人能及。

1.3 本文主要研究方向

发动机表面缺陷通常有多种的形状和结构，另外缺陷的种类和大小完全随机产生，位置的分布也少有规律。本文提出一种基于 YOLO v3[19]的发动机表面缺陷检测法，由此可以发挥出卷积神经网络自动学习更新的优势，提取出各类缺陷的特征，然后在 YOLO 的统一网络的帮助下定位目标并识别缺陷。为此，我们需要研究发动机表面的缺陷究竟有哪些特征，由此剖析出检测任务的特点并顺藤摸瓜设计出优秀的解决方案；要产出功能完全，性能良好的算法，对缺陷的检测与识别准确率达到可以接受的水平；最后对算法做出改进，研究 YOLO 算法中哪些关键的参数会对训练出的模型产生重要的精度影响；最后搭建原型系统平台，将上述完成的算法封装制作应用程式，提高检测程序的可使用性。

主要研究内容和步骤如下：

1）发动机表面缺陷图像进行数据扩充和标注以适配 YOLO v3 网络

实验阶段，收集到的发动机缺陷图像是有限的，而且与训练 YOLO v3 网络需要的输入的标准输入图像有区别。如果是一般的 YOLO v3 算法的实验，可以直接使用 COCO 数据集或是 VOC

2007 数据集，但是缺陷检测是一种特殊的目标检测，目前不存在公开的高质量数据集，所以需要在实验之前利用收集到的图片，自己制作足量的，合乎要求的数据集。为此，实验需要详细了解 YOLO v3 算法中网络模型的优化流程，了解一张标准的图片被算法学习的过程，被算法检测的过程，进而建立发动机表面缺陷数据集。

2）训练一个针对发动机表面缺陷的 YOLO v3 检测模型

制做好可供 YOLO v3 训练算法学习的数据集以后，实验开始训练针对发动机表面缺陷的 YOLO v3 检测模型。虽然标准的 YOLO 模型已经可以检测出 VOC 2007 数据集中的 20 种目标，但这些目标都跟发动机表面缺陷相距甚远，它们通常是汽车、人物等生活中的目标。可以想见，要让模型有能力检测出表面缺陷，需要进行大量针对性的训练。为此，实验中会研究训练过程中的损失函数变化，因为损失函数是决定一个模型是否拟合的重要参数。除了损失函数以外，实验还会研究其他一些用于定量分析模型性能的指标，如召回率，精确度等，并重点研究为了提升这些指标，实验中应该如何调试算法中的两个重要参数：epochs 和 batch_size。最终得出结论，在发动机表面缺陷检测问题中设置怎样的参数可以使模型最优。

3）搭建发动机表面缺陷检测的原型系统平台

在训练出了最优秀的模型以后，实验希望使检测过程更加直观。比起在程序编辑框中输入代码使程序运作来检测图片，实验希望实现检测图片的可视化操作，以此证明 YOLO v3 算法确实可以运用到实际生产中，而不是仅在实验室中，仅由编程人员才能使用。为此，实验中也会使用可视化的平台搭建工具，实现“站在使用者的角度”，将冗长复杂的检测代码封装成“即插即用”的 windows 应用程序。原型平台将力求简洁明了，并保持原程序的强大功能。最后，实验还会制作与该原型平台对应的简明的操作指南。

1.4 本章小结

本章先阐明了发动机表面缺陷检测的重要性，以及在目前的技术下，要实现自动化实时检测的技术上的局限。接下来，结合缺陷检测研究的历史发展和现状，详细解释了学习检测法是目前的研究潮流；而其中，深度学习又是学习检测法中最具有研究价值的，因为由深度学习的历史发展和研究现状可知，深度学习，不仅识别准确率在同类检测方法中最为可观，而且作为一种可以进化的算法，深度学习的模型可以不断被优化，这非常符合当今工业生产中越来越复杂的生产环境。接下来，本章简明地介绍了 YOLO v3 算法，这是深度学习领域中的一种最新颖的算法。由于 YOLO v3 算法最开始为生活中的目标设计，如汽车，行人等，为了将 YOLO v3 算法更好地适配到发动机表面缺陷检测这一生产中的问题，本章设计了三大步骤，首先收集到的发动机表面缺陷图像作为训练材料训练出可以普遍检测缺陷的模型，然后改变具体参数优化模型准确率，最后封装检测模型到普遍使用的应用程序中。

2 卷积神经网络基本理论及缺陷问题分析

近年来，卷积神经网络的研究成为深度学习研究中的热点，应用在机器视觉方面的程序设计在不同类别的目标检测中都取得了一定的成果，但是缺陷检测任务由于其独特的难点，使相关算法在缺陷检测中表现一般。YOLO 算法独特的统一网络实现对模型端对端的优化，本文认为这个特点非常适合缺陷检测任务。本章会详细介绍卷积网络神经，和在此基础上使用的 YOLO 算法，另外，本章还会研究发动机表面缺陷检测的难点，缺陷目标与一般目标的区别，主要是尺寸小，个体差距大和易受干扰等，和由此给算法编写带来的困难。

2.1 卷积神经网络基础理论

Neural Networks，即神经网络,是大量的基本单元逻辑严密地相互交错而成的，人工神经元就是这个基本单元，见图 2.1。这个功能强大的网络在设计时极大地参考了真实的猫的神经元。生物的神经系统是由突触释放神经递质，将“兴奋”给下一个神经元；同时，神经元可以和多个接受兴奋的其他单位（其他神经元或者细胞）相连，如此构成信息传递网络。同样的，神经网络 CNN 的运作即在一个“神经元”接受数个“兴奋”x_i 的输入以后，会在神经元内部接受兴奋阈值（threshold）的判断并加以偏差（bias），即如果处理后的输入值可以达到兴奋阈值，就会激活这个“神经元”并输出一一映射的结果。从神经网络的整体来看，神经元层层排列，复数个的神经元会组成一个完整的层（layer）。网络的输入层首先接受信号，并输出结果至下一层，如此反复，多次处理以后，最后一层输出最后的结果。

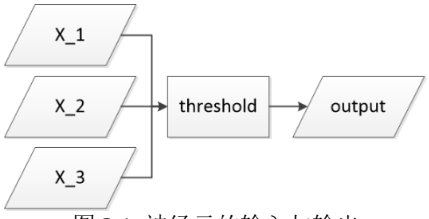


图 2.1 神经元的输入与输出

卷积神经网络 CNN，是最基础的神经网络经过广大学者的优化改进进化来的，它的设计还借鉴生物的视觉感知（Visual perception）的概念，即它的运作方式更接近生物从眼睛接受视觉信息后传递到大脑的过程。这种思想我们直接借鉴生物学中的定义，感受野（receptive field）。跟普通的多层神经网络所有网络以全连接的形式相连，见图 2.2，有所不同的是，卷积神经网络当中，任何一个神经元，有且仅有跟上一层某些特定神经元的连接。处理输入的时候，某一特定层中全部神经元遵守同样的加权规则，处理接受的所有信息。如此，第 N 层的每个神经元都提取有第 N-1 层的部分特征信息，那么第 N 层就可以总结出第 N-1 层信息的特征。这样的结构好处是，特征被层层提取，最细微的特征也可以在某个神经元储存好，训练时避免遗漏。

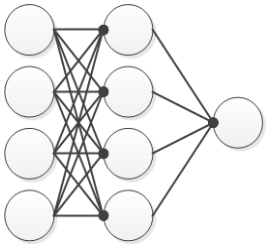


图 2.2 CNN 局部连接示意图

经典卷积神经网络模型结构主要包含：input、convolution layer(卷积层)、pooling layer(池化层)、Full connected layer(全连接层)、output 。CNN 必以 input 为始以 output 为末，这两层是位置已经固定的，卷积层和池化层以及一些其他的激活层（如 ReLU 激活），另外通常也会有一层池化层在着网络的输出层之前，见图 2.3。激活层的位置一般在卷积层和池化层之间，网络的一个特征提取层就是由复数次的连续卷积层与池化层组成的。网络会通过数次的特征提取和特征压缩，详细分析特征信息并输入到全连接层，最终的结果将由输出层给出。

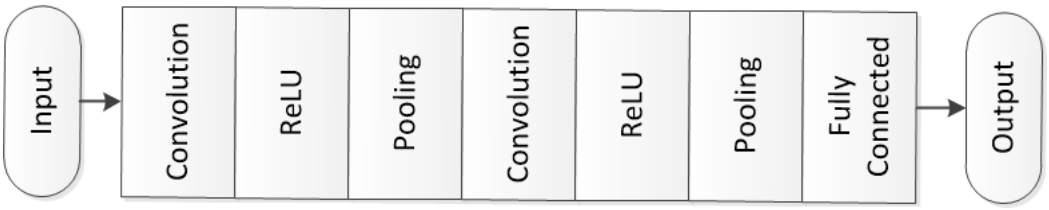


图 2.3 卷积神经网络结构示意图

1）卷积层（Convolution layer）

卷积层可以提取图像的特征，执行这个操作的工具叫做卷积核(filter)。数个卷积核会同时存在于一个卷积层中。卷积核各自拥有不同的权重。主要参数：边缘补足(padding)，步幅(strides)，尺寸(size)构成一个卷积核，图片经过一次卷积操作以后尺寸会发生变化：

$$h_{out}=\left(h_{in}+2p+h_{filter}\right)/s_y+1 \tag{2.1}$$

$$w_{out}=\left(w_{in}+2p+w_{filter}\right)/s_x+1 \tag{2.2}$$

其中: h_{in} 和 w_{in}是输入图像的高度和宽度，p 是像素在边沿的补充量，h_{filter}和 w_{filter}是卷积核的高度和宽度，s_y和s_x是卷积核在纵坐标和横坐标的步幅。

一张普通的 RGB 格式图片有 3 个通道，把它输入神经网络以后就会自动分割成 3 个特征图。网络会使用数个卷积核对所有 3 个通道进行卷积操作，由此可以确保不漏过任何通道中的特征。一般化的，当输入图像有 a 个通道数，我们的卷积层中有 b 个卷积核，那么对于每一幅图像，a*b 次卷积操作将被在每一个卷积层被执行。在一个卷积层中，原始的输入首先经过卷积，然后加上偏置 bias，如果把激活层也作为卷积层中的一部分，就会将激活函数运用于加上偏置以后的结果，结果如下：

$$M_j=f\left(M_{j-1}\otimes W_j+bi_j\right) \tag{2.3}$$

其中，M_j是经过 j 层卷积的输出特征矩阵,M_{j-1} 是第 j 层的输入特征图，即第 j-1 层的特征图，“⊗”即卷积运算，W_j是卷积核的权值向量，bi_j为偏置。

2）池化层（Pooling layer）

池化层，由于其往下采样的特点，又叫降采样层或下池化层，一般出现在卷积层之后，用于

压缩图像的特征，由多个下采样特征映射[40]组成。池化层有三个功能：第一，它对特征图进行过滤，保留主要特征;第二，可以压缩特征矩阵的大小，减小网络参数;最后，扩大图像的感受野。下采样层中的过滤器通常为 2×2 的矩阵，采用的步长也和尺寸一致。池化时，网络计算并选择每个抽样的特征。池化之后的特征矩阵数量保持不变，但是特征矩阵的尺寸会减少到原来的 1/4。第 i 层池化操作可以总结为如下公式：

$$M_i=p(M_{i-1}) \tag{2.4}$$

其中 p(x)为池化函数，最常使用函数有两种，分别是平均池化和最大池化（average pooling & maximize pooling）[22-23]。平均池化更加适用于特征图平整，局部差异可接受的图像；最大池化对于特征“尖锐”的图像较为敏感。

3）全连接层（Fully Connected layers，FC）

通常情况下，全连接层在一张 CNN 网络的倒数第二层，即输出层之前使用，神经元发布形式采用分层分布。第一层用于特征展开，功能是把最后一个池化层的输出信息，通过映射转换为一维向量。例如，如果原始输入是一个 2×2×256 的特征矩阵，那么在全连接层中，一个 1×1024 的一维向量会被输出。一般的，第二层是隐含层，每个神经元接收前一层每一个神经元的输入，矩阵运算后，后一层所有神经元再接受所有的计算结果。矩阵向量积的公式为：

$$M_{out}=WM_{in} \tag{2.5}$$

其中M_{in}和M_{out}是隐藏层的输入和输出，W 是权重矩阵。从 Conv.层和 Pooling 层提取的特征信息可以有全连接层映射至样本标记空间，最后输出相应的概率。

4）输出层（Output layer）

输出层是卷积神经网络的最后一层，功能主要是分类，所以也叫分类层，概率输出会在这一层通过之前网络获得的特征转换得到。两种用于转换的函数分别是 Logitics 离散分布函数分类器和 Softmax 指数分布函数分类器。前者为 1/0 输出，通常用于定性判断；后者会输出概率值，通常用于定量预测。

2.2 YOLO 算法基础理论

人类的视觉系统可以快速处理看到的信息内容，基本上一眼就可以认出视野里的目标是什么，它们的位置，尺寸，颜色，形状等。这个视觉系统在快速处理的基础上还可以达到相当高的准确率，由此，人类获得了执行复杂任务的技能，比如我们可以边看电视边打电话，在不遗漏电视内容的同时正常与电话对面的人沟通。最近流行的 R-CNN 算法采用的是“滑窗法”，“滑窗法”的目标检测算法核心比较移动，它将检测问题转化为了影象分类问题。YOLO 将目标检测这一任务转换成了回归问题，图像的每一个像素预测的边界框的坐标和目标的类别。YOLO 算法的终极目标即是，你只需看一次图像（YOLO）就可以预测出目标物体的类别和位置。[20]

2.2.1 YOLO 算法基本思想

YOLO（YOU ONLY LOOK ONCE）算法之前，目标物体的检测工作都是如上文提到 R-CNN 中的分类器（classifier）来进行的。YOLO 网络作为一个统一（unified）的网络，在输入一张完整的图像后可以输出标记好的位置信息和类别信息的概率。正因为整个检测网络是统一的，所以我们可以进行端到端（end-to-end）[20]地网络模型优化。

YOLO 网络结构由 24 个卷积层构建，在输出层之前是两个全连接层，见图 2.4。输入的图像设定为 448×448×3 的 RGB 图像，3 即为三通道。因此，其他格式或者尺寸的图片在运行网络会提前处理成标准格式。网络的输出结果为一个维度确定的张量。

YOLO 的核心思想是将图像分为 448×448 的网格，每个网格通过算法预测出 b 个边界框（bounding box）同时计算出网格中有无目标的概率以及是哪一个类型的目标分别是多少。当我们一共设置 c 个类别的目标时，则每一个网格会输出 b×5+c 的向量。

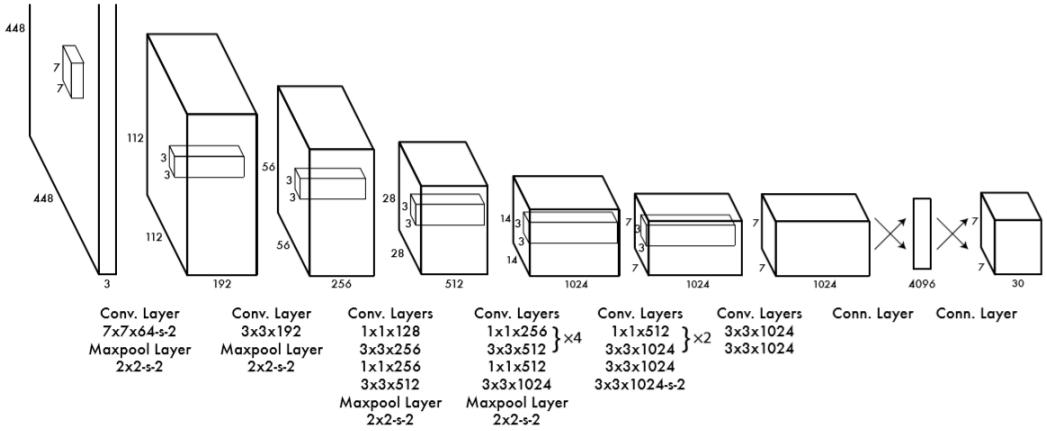


图 2.4 YOLO 网络经典框架

举个例子，一张 448×448×3 的图像输入系统，一个单元格预测 2 个边界框，且有 20 个类别的目标，依然见图 2.4。训练样本将会如下：对 448×448 个网格，每个网格输出一个（20+2×5）=30 维的向量。边界框的参数分别为中心点坐标：2 个参数，尺寸：2 个参数，边界框中存在目标的概率：1 个参数，为 0 或 1，代表判断存在或不存在——共计 5 个。综上，该 30 维向量可以表示为：

$$y=(pr\_1,cx\_1,cy\_1,h\_1,w\_1,pr\_2,cx\_2,cy\_2,h\_2,w\_2,c1,c2,c3,\ldots,c20) \quad (2.6)$$

其中 c1 分别表示“类别 1”目标在该网格中存在的概率，依此类推。所有类别的概率相加应当为 1。如果图像里有复数个目标，YOLO 会分别取出所有目标的中心点坐标，定位到包含了中心点的网格，然后定义该网格将“负责”预测这个目标。

简而言之，YOLO 算法将整张图片分为 S×S 个网格，每一个网格各自卷积，而非采用 Faster R-CNN 中“滑窗法”依次卷积[21]，最后输出 S×S 个 N 维向量对应于 S×S 的每一个网格的预测目标类概率和目标坐标。

2.2.2 损失函数

损失函数（loss function），在 YOLO 中使用均方差优化损失函数的模型参数，也就是网络输出的 S×S×N 特征矩阵与真实图像的同尺寸特征矩阵的均方误差。损失函数由输出数据于真是标定数据之间的三项误差构成：中心坐标(coord)误差、IoU 误差和分类（class）误差。

$$Loss=Error_{coord}+Error_{iou}+Error_{class} \quad (2.7)$$

1) 坐标误差

坐标误差由两部分组成，分别是对预测的中心坐标做损失和对预测边界框的宽高做损失。在具体计算误差之前，我们要先了解这个单元格对于自己输出的预测如何做定性判断。这里会有两

种情况，也就是前文提到的非一即零的“边界框中存在任何一个目标的概率”。具体来说：如果网格中存在目标，则认为它的 b 个边界框的预测是有价值的；反之，无论得出的是什么预测，都不应该采纳。我们定义：

$$I_{i,j}^{obj}=\begin{cases}1, & \text{当网格 i 有目标中心，第 j 个边界框预测有效}\\ 0, & \text{网格单元 } i \text{ 中不存在目标}\end{cases} \tag{2.8}$$

首先，计算对于中心坐标的误差，就是简单的坐标误差，我们认为可以直接套用均方差公式。其次，考虑边界框宽高误差时，我们不能简单的套用均方差公式，这是由于大边界框在出现误差是可能会极大地影响函数值，这对于小边框出现的误差是“不公平”的，因此应该削弱尺寸的绝对值过大带来不良影响，在这里我们采用给宽高开平方根的手段。

至此我们得出了坐标误差公式：

$$\begin{aligned}Error_{coord}=&\lambda_{noobj}\sum_{i=0}^{s\times s}\sum_{j}^{b}I_{i,j}^{obj}(x)*\left((x_i-x_{0i})^2+(y_i-y_{0i})^2\right)\\&+\sum_{i=0}^{s\times s}\sum_{j}^{b}I_{i,j}^{obj}(x)*\left(\sqrt{h_i}-\sqrt{h_{0i}}\right)^2+\left(\sqrt{w_i}-\sqrt{w_{0i}}\right)^2\end{aligned} \tag{2.9}$$

其中x_i，y_i，h_i，w_i分别第 i 格预测的边界框位置信息，x_{0i}，y_{0i}，h_{0i}，w_{0i}则是真实数据，$\lambda_{noobj}=5$是中心坐标误差的权重。

2）IoU 误差

IoU，即交并比。这里要计算的实际上是预测边界框与真实边界框的 IoU 和网格自己预测出的置信度的误差。通俗地讲，要让模型自己“学会”自己的判断到底有多可信，这对于提高程序的稳定性非常主要。同样的，$1_i^{obj}(x)$将作为第 i 个网格对自己的先行判断加入误差计算；与之前不同的是，第 i 网格不存在目标时，它预测的置信度依然有参考价值，只是不可与存在目标的网格加以同一权重。

$$Error_{iou}=\sum_{i=0}^{s\times s}\sum_{j}^{b}I_{i,j}^{obj}(x)*\left(p_i-p_{0i}\right)^2+\lambda_{noobj}\sum_{i=0}^{s\times s}\sum_{j}^{b}I_{i,j}^{noobj}(x)*\left(p_i-p_{0i}\right)^2 \tag{2.10}$$

其中$1_{i,j}^{noobj}(x)=1-1_{i,j}^{obj}(x)$，在 YOLO 设计者设计里$\lambda_{noobj}=0.5$的权重是合适的，另外$p_i$即第 i 格预测置信度，$p_{0i}$即第 i 格预测边界框于真实边界框的 IoU。

3）分类误差

与坐标误差类似，如果第 i 格不存在目标，则不惩罚分类误差。

$$Error_{class}=\sum_{i=0}^{s\times s}I_i^{obj}(x)*\sum_{c}^{class}((c_i-c_{0i})^2) \tag{2.11}$$

其中，c_i第 i 格存在 c 类目标的概率。

2.2.3 非极大值抑制

在使用目标检测时，不仅要对目标进行检测，而且要找到目标的准确位置。如果实际边界框与算法中指定的边界框不完全匹配，计算 IoU(Intersection-over-Union)的值就非常的重要，这是定量分析对象定位算法的精度的重要办法，即，IoU 可以确定两个边界框重叠的程度。交并合函数的功能就是计算预测出的边界框与真是标注的边界框的交（面积之比）。一般情况下，如果 IoU

≥0.5，就将这个计算机检测任务定义为成功。当预测出的边界框和真实的边界框完全重叠时，则IoU=1，也就是交集与并集相等。虽然 IoU≥0.5 时结果是可以接受的，但是按照实际应用情况可以适度调高。因此，它是一种合适的评估定位精度的方法，只需要正确地计量检测成功和识别准确目标的个数就可以确定的定位是否准确。

当某个目标的中心落在某个网格里面时，该网格就负责检测这个特定的目标。S×S 网格中的每一个都会预测 b 个边界框，每一个边界框都是 5 个值来定义。这样最基本预测单位就成了一个个的网格。

目标检测中存在的一个问题是，一个对象可能会被算法多次检测，这样的后果是算法很会浪费很多时间，极大地降低了程序性能。非最大抑制确保了算法只检测每个对象一次。假设我们的任务是在这一图中检测目标 A 和目标 B。目标 A 只有一个中心点,所以对象 A 会被一个网格所“负责”,所以理想状态是目标 A 只被一个网格预测。但在实际运行中,我们分别对每个格子运行预测模型，可能会有多个格子都预测了目标 A，这无疑带来了很大的性能浪费。

非最大值抑制就是避免多格子预测一个目标的算法，达到的最佳效果是每个目标只被检测一次，而不是一个对象被多个格子“争抢”。在测试非最大值抑制的时候，对于每个边界框都应套用下列公式来明确是否能够负责（responsible）该目标：

$$\begin{aligned} &confidence * Pr(class_i|Object) \\ &= Pr(Object) * IoU_{truth}^{pred} * Pr(class_i|Object) \\ &= IoU_{truth}^{pred} * Pr(class_i) \end{aligned} \tag{2.12}$$

其中，confidence 即该单元格存在目标时的置信度，显然，当单元格内不存在目标的时候，无论真实与预测的交并比是多少，都应当按零处理，这就是$\Pr(\text{Object}) * IoU_{truth}^{pred}$的含义。另外$\Pr(class_i|\text{Object})$也就是在单元格已经存在物体的情况下，存在的是第 i 类目标的概率。由此这个置信度得分既包含了单元格中各类别目标存在的概率，也包含了此单元格每个边界框自己预测的信息。

这个算法的实现流程为：首先从所有的检测中挑选出置信度得分最高的边界框及其所在的单元格，我们称这个边界框为“理想边界框”。非最大值抑制就是遍历其他的边界框，如果发现与“理想边界框”的 IoU 很高，即高度重叠的那些，就抑制它们的输出。举个例子，如果有个置信度得分约为 0.5 的边界框和置信度得分为 0.9 的“理想边界框”重叠程度很高，它就会被抑制输出。以此类推，一个个地检查其他目标的矩形框，为每一个目标找到一个最理想的边界框，接着运用非最大值抑制算法剪枝其他的“雷同”的边界框。这就是非最大值抑制的核心用法。

2.3 缺陷检测任务分析

本文的研究目标为发动机表面的缺陷，目前市面上用途最广泛，制造技术最成熟的汽车发动机是铁铸发动机。发动机缸体在铸件中算得上是结构复杂，不仅要求尺寸精度高度达标还要满足复杂的内外形结构，所以它的铸造难度是很高的[24]，因此本文选取铁铸发动机的缸体作为缺陷检测的对象。铸铁在铸造原料中属于塑性较差的一类，为了满足发动机缸体的复杂结构，通常自动化流水线生产中采用砂型铸造的方式，而砂型铸造也是最常用的铸造方式，例如钢、铁和大多数有色金属铸件都可以通过砂型铸造的方式获得，是比较具有代表性的铸造方式。因此，本文选择的铁铸发动机缸体表面缺陷的检测对其他广泛的铸件的表面缺陷检测亦有具体启发意义。

2.3.1 常见缺陷类型

一般来说，目标检测算法对于输入图像的质量有较高要求，图片的像素会直接影响到识别结果，但是 YOLO v3 算法由于其独特的统一网络结构，固定了输入图像的尺寸为 320×320，416×416 或者 608×608，所以采集的图片只需要大于它的要求尺寸即可。本文的程序编写采用 416×416 的输入图片尺寸，利用 500 万像素工业摄像机的灰度模式收集原始图片为 2448×2050 的单通道图片(灰度图)，在预处理后可以满足 YOLOv3 算法的输入要求。本文的算法编写时考虑到推广应用范围的需求，即可能处理三通道图片（如 RGB 格式）的需求，在预处理亦将原始图片拓展为三通道，除原通道外两新增通道为空白的通道，这样完成训练的程序也可以处理三通道的图片。考虑到发动机表面缺陷可能会十分微小，本文收集原始数据时多采用局部特写，力求记录下每一个细小的瑕疵，见图 2.6：

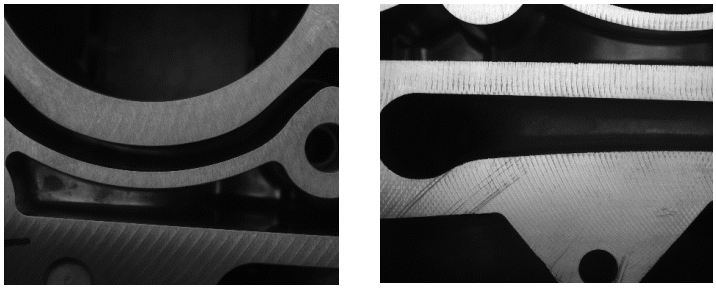


图 2.6 原始图像的两个样例

本文的缺陷大致分类两类：点状缺陷（以沙眼为主）和线状缺陷（以划痕为主）。首先，沙眼主要为形状不规则的沙眼浅坑，是在铸造时由于熔融金属飞溅产生的，由于其的的几何形状毫无规则，并且在不同的采光环境下，由于光源方向问题，可能还会产生颜色不甚均匀的图样，见图 2.7：

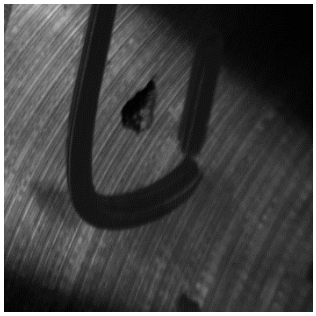


图 2.7 一个典型的沙眼

是本文需要检测的缺陷中的一个难点。其次，划痕是最常见的表面现状缺陷，通常由铸件生产中的意外撞击产生。虽然其线状的特征明显，但由于存在相对较多且微小，检测到所有划痕的难度依然不低；另外某些碰撞也可能造成小坑状的，非线性的划痕需要另加考虑，虽然从缺陷分类的角度看，这种“小坑”依然属于一个划痕，但它于大部分划痕的线性特征有区别，见图 2.8，如果仅设置“划痕”这一大类，可能会增大训练难度。

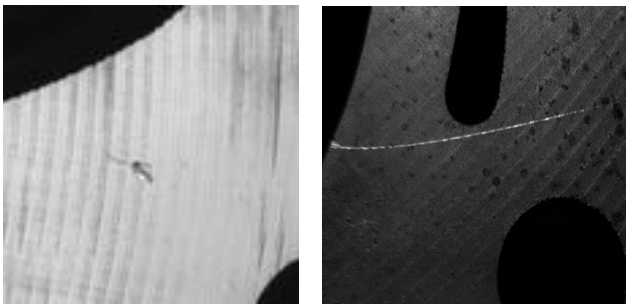


图 2.8 (a)非线状的划痕 (b)线状的划痕

总结，本文的缺陷类型分为：1）划痕，2）非线状划痕，3）沙眼。

2.3.2 任务难点分析

YOLO 算法在实际中已经应用在生活中的很多领域，如交通监控车辆来往实时识别，自然图像中的野生动物识别，甚至是艺术品中人物形象识别等[20]，但在缺陷目标的探测上研究是相对不足的。主要的原因是，缺陷探测任务中的目标，也就是“缺陷”，无论在颜色还是几何特征上，与原算法设计的目标车辆，人物，动物等区别较大，直接原封不动地套用算法，得出的结果往往鲁棒性差。为了训练出更加契合于发动机缺陷特征的模型，接下来我们对发动机表面缺陷探测的独特性，也是难点作出研究：

1）缺陷目标尺寸小

YOLO 算法的核心是通过每个单元网格分别作出边界框预测取最佳边界框，这样的思想省去了像 Fast R-CNN 算法中的预测目标可能的存在区域这一步骤，极大优化了速度，但也带来一些限制。比如，难以检测出微小且集群的目标。具体来说，如果 416×416 网格中某个 1×1 的小网格中存在两个或两个以上的微小目标，网格仅有能力探测出其中的一个。对于微小的划痕来说，在某个小区域内大量出现是可能的且需要考虑的事件。比较直接的解决方案是改变程序设计中一个单元网格仅保存一个最佳边界框预测的设定，这样可以保存一个单元格中的多个小微目标，不过这样的不良影响是巨大的。对于每一个卷积层需要卷积的数据量大量增加，会极大的拖慢训练进程；且对于大多数单元格其实是不存在目标的，让他们也一起多做数倍的运算，无疑是吃力不讨好的。因此，本文的解决办法是改变训练样本集时的思路，与其一一标注出所有集中在一起的微小目标，不如标注出这个微小目标的集群。与一般 YOLO 算法详细探测出个目标数量和位置不同，发动机表面缺陷我们更需要确定存在缺陷与否和存在缺陷的位置，例如对于 YOLO 交通监控，我们需要了解画面远处究竟聚集了多少车辆，而对于发动机表面缺陷，我们只需要了解画面的哪个框中存在缺陷。因此这个难点可以通过定义不同的标注方法解决。

2）同一种缺陷不同个体差距大

发动机表面的缺陷都是生产过程中意外导致的，与传统 YOLO 检测中的目标不同，例如汽车都有着相似外轮廓形状，人脸都有着相似构造等等，对于缺陷来说，没有规律的样式是最大的规律。作为同类缺陷，例如沙眼，在图像有的个体可能近乎全黑的类圆形，有的个体却也可能是黑吧阴阳颜色的不规则图形；再如线状划痕，全黑和全白的划痕可能会出现在同一张发动机表面图像中，因为光源方向的关系。YOLO 的训练把它们自动学习成一类缺陷的概率自然不如程序原先设计的学习除同类汽车的概率高，要解决这个问题只能从训练的程序入手，加入一些特别的，针

对于发动机表面缺陷特征的判断。

3）干扰性强

发动机表面的缺陷相对于原设计的目标检测具有更大的干扰性[24]，对于缺陷探测的干扰主要有二：一是发动机表面特殊的纹理特征对缺陷检测的干扰，二是发动机生产过程中的其他因素对缺陷检测的干扰。

发动机的表面通常需要机械自动化的或者手工的打磨，打磨操作过后表面有出现特殊纹理或者是斑纹的可能，甚至在光线角度和打磨产生的凹凸的共同作用下，还可能产生沙眼的假阳性，如图 2.8(a)所示。理论上讲，这些特征是不符合百分之百理想的设计的，但是设计中同时有规定，不平整的特征不超过一定程度就是合格的，虽然它们在图像上非常明显，但是不应被识别成缺陷。另外，在生产过程中，由于油污，打磨残渣等可能粘在表面，会遮掩一部分的缺陷，而且容易被训练算法学习为缺陷的一部分，这样检测正常的缺陷时就会降低识别能力。在本文的训练中，我们采用油性笔在缺陷上的图画，见图 2.8(b)，来模拟这种情况。

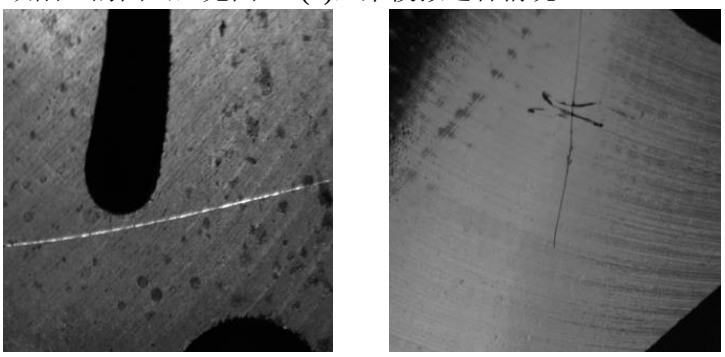


图 2.8 (a)暗斑干扰的样例 (b)黑色油污干扰的样例

2.4 本章小结

本章详细介绍了卷积神经网络和 YOLO 算法，并研究了将它们应用到发动机表面缺陷检测问题的可行性。为此，本章还详细分析了，在运用 YOLO 算法时，发动机表面缺陷独特的特征可能带来的问题以及相应的解决方法。缺陷检测和普通目标检测的区别就在于，无论什么缺陷在生活中都不常见，算法的最初设计者不会考虑这样特定的问题，实验中需要特别考虑缺陷独特的特征，而这也是任务的难点，需要利用多种调试，使原始算法“适应”本实验特定的问题，即发动机表面缺陷作为目标的检测。

3 基于 YOLO v3 算法的发动机表面缺陷检测

近年来，通过学者的不断研究，最开始的 YOLO 算法已经经过了两次升级达到了 YOLO v3 的版本，在保持了原有的检测速度快的特点的同时，极大地提高了精度。本文的实验部分将采取 YOLO v3 版本，通过新的网络结构的应用，最大程度地提升发动机表面缺陷检测的精度。本章会先详述发动机表面缺陷的数据集的建立过程，将有限的原始数据处理并扩容成容量大且质量高的定制数据集，然后介绍一张符合输入标准的图像如何在 YOLO v3 的网络中完成学习并最终参与检测。

3.1 发动机表面数据集的建立

3.1.1 数据增强

标准的 YOLO v3 训练通常使用 VOC 2007 标准数据集或者 COCO 数据集[19]，内含图片数量有三万之众，如此才可以训练一个实际应用价值的模型。在研究发动机表面缺陷时，我们可以接触到的实例是有限的，即使是特写每一处微小的缺陷，得到的原始数据量依然非常的有限。因此数据增强是必要的，增大数据集的容量对训练成功有很大的帮助。

数据增强需要对原始图片直接进行加工，加工的方法一般有：旋转、翻转、加背景噪点、缩放等等[25]。数据集的容量通过这些方法变大的同时，还可以改进模型对模糊的图像，含噪点多的图像的鲁棒性

增强之后，数据集的容量在两千左右，由于我们检测的目标只有 3 类，对比有 20 类目标 COCO 数据集，是可以满足需求的。每一张标准的输入图像即 416×416 的三通道图像。

3.1.2 数据标注

本文使用专业的目标探测标注软件 labelImg 进行数据标注。

首先，在 data/predefined_classes.txt 预定义类别文本的中输入我们的类别：scratch(划痕)、nonlinear scratch(非线状划痕)和 hole(沙眼)。

这里，举例一张划痕缺陷图片的标注过程。首先在 anaconda prompt 中进入 LabelImg 的目录运行 labelImg.py 打开软件的交互界面并选择要标注的图片，见图 3.1：

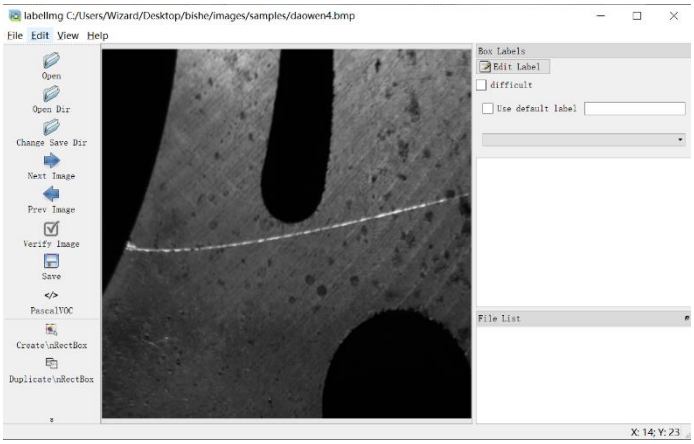


图 3.1 选择发动机表面缺陷图像

点击“create\nRectBox”创建一个边界框，框住图中的划痕，然后从设置好的三类类别中选择划痕（scratch），见图 3.2，最后保存到储存路径上的文件夹。

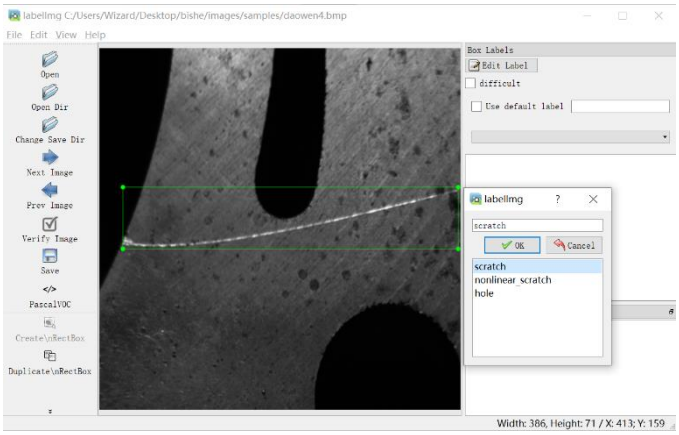


图 3.2 选择好边界框并确定类型

最后把保存的格式选择为 YOLO 格式，每一张图片都会输出一个同名的文本框文件。其中有且只有 n 行，每行 5 个参数：边界框的类别,中心 x 坐标，中 y 坐标，边界框的宽，边界框的高，其中 n 就是边界框的数量。

3.1.3 数据集的划分

如果直接使用原始数据中的所有图片来训练模型，得到的模型会最大程度上地拟合出所有图片，这时再用原始图片中的图片去进行测试可能会得到很好的检测结果，这样的结果可能是不真实的；当新的样本图片出现的时候，准确率可能会大幅度下降，然而由于没有划分数据集，在程序调试的阶段就不能实时了解模型的真实情况，盲目乐观。因此，将数据分为训练集、检验集和测试集是必要的。

1）训练集（Training set）

即用来训练的数据集，将它输入 Train 程序，形成一个基本的模型，之后的检验集将在这个基本模型上加以改良。本文程序中，训练集占比 80%。

2）检测集(Cross Validation set)

在已经训练出基本模型以后，修改一些关键的参数以后，使用基本模型对检测集来进行预测，记录下每个新模型的准确度，在各种改良模型中选择出准确程度最高的。本文程序中，检测集占比 10%。

3）测试集(Test set)

经过训练集和检测集对模型的训练，一个最优的模型已经被得出，我们再用测试集来最终检验该模型的性能和准确度。这样，测试集对于模型来讲就是全新的样本，可以得出最真实的评价。本文程序中，测试集占比 10%。

3.2 YOLO v3 算法理论

YOLO v3 是最开始的 YOLO 算法经过两次升级以后得到的。YOLO 算法的速度是超越所有算法，它最大的缺点是定位不准确[20]，我们认为导致这个问题的原因是 YOLO 中使用的全连接层

预测边界框造成了空间信息丢失的问题。同 YOLO v2 一样，YOLO v3 借用了 Faster R-CNN 中的 anchor box 的思想[19,26]。在每个单元格预测目标边界框之前，算法会首先提出 5 个初始的 anchor box，见图 3.3，这样预测边界框就变成了预测边界框与 anchor box 的差异。这样不仅加快了模型的收敛有提高的定位精度。

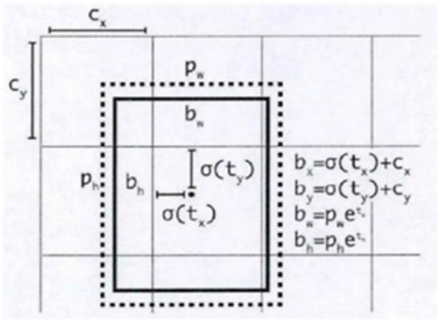


图 3.3 anchor box 在 YOLO v3 中的设计

其中，t_x, t_y, t_w, t_h分别是网络预测出四个坐标值，代表了预测框和 anchor box 的中心点横坐标差异，中心点纵坐标差异，框宽差异和框高差异，然后带入下列算式可以得到预测框的绝对位置：

$$b_x=\sigma(t_x)+c_x \tag{3.1}$$

$$b_y=\sigma(t_y)+c_y \tag{3.2}$$

$$b_w=p_w e^{t_w} \tag{3.3}$$

$$b_h=p_h e^{t_h} \tag{3.4}$$

其中，σ 函数的作用是将函数值归一化为[0,1]，（c_x，c_y）是当前单元格左上角与图像左上角的偏移。这样的计算中包含了单元格参数对最终预测边界框的约束，所以比起直接预测一个完整的边界框及其的四个位置参数会使函数收敛的更快，并且结果也更精确。

3.2.1 网络模型

1)内置网络（backbone）

YOLO v3 的设计者采用了 Darknet53 作为基础网络模型[19]，见图 3.4：

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

图 3.4 darknet-53 各层结构

Darknet-53 使用的卷积核尺寸为 1×1 和 3×3 两种，前者的功能是限制卷积层的通道数不要太多，降低特征矩阵的维度数，从而精简神经网络，后者的功能则是正常的卷积提取特征。该网络共卷积了 53 次，并且借鉴 Res Net 的残差连接思想来加深深度，又用改变步长（padding）的方式来精简特征矩阵的尺寸来取代池化层的作用。

2)整体结构

可以看到，darknet 网络最终采取的是全连接的方式映射特征矩阵，但显然这样的处理计算量极大，是极度消耗时间的，所以 YOLO v3 在使用 darknet-53 网络的时候舍弃了这种设计，取而代之的是，利用上采样特征矩阵然后与预测层拼接起来的方式实现多尺度的特征图像输出，整体结构见图 3.5：

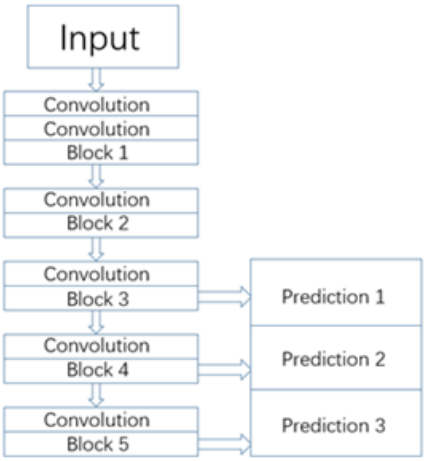


图 3.5 YOLO v3 整体网络结构

如图所示，网络中一共有 5 个卷积模块（conv & block）和 3 个预测层（predict）构成。网络最终的输出是 3 个预测层拼接成的含有不同尺寸的特征的特征图，除预测层 1 是 5 个模块预测出的，预测层 2 和 3 分别从模块 3 和 4 中采集出来。

3.2.1 样本检测过程

我们可以得出缺陷检测的操作分为以下几步：

1） 将输入的图片缩放为 416×416 的尺寸，并按照 416×416 的规格画好单元格。

2） 每个单元格内生成 2 个边界框，随后遍历搜索图中所有目标有没有某个或某几个目标的中心落在此单元格内。

3） 对于落在单元格内的目标，先预测其类别及其概率，随后预测其中心点的位置及边界框的宽高并计算出置信度得分。

4） 所有类的类别概率与置信度得分相乘即为该边界框的综合置信度得分。

5） 对网格中所有边界框进行 1)-4)操作最后输出 416×416×(2×5+3)的张量。

流程如图 3.6 所示：

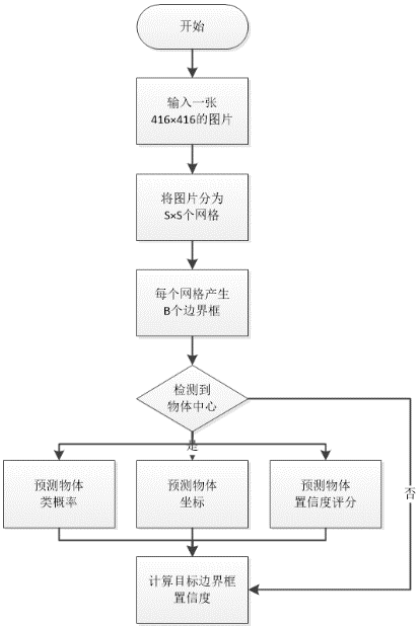


图 3.6 YOLO 网络的检测流程

至此，判断边界框中存在目标中心为真的边界框将会输出：目标类别，置信度，边界框中心点横纵坐以及边界框的宽高，6 项数据，结合绘图程序可以将预测的边界框标注在原图之上，见图 3.7：

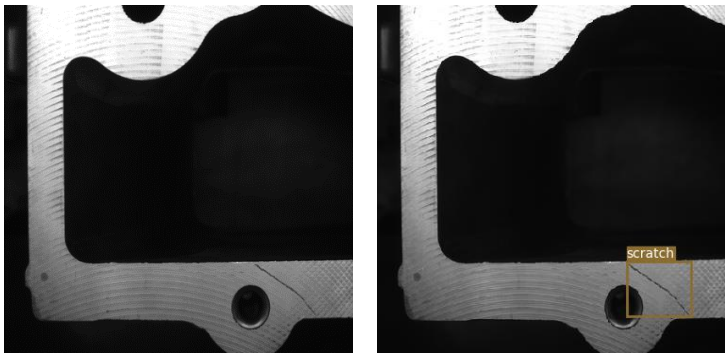


图 3.7 原始缺陷图与标注的预测图

3.3 本章小结

本章介绍了实验开始之前的所有实验用数据集的准备工作，为了制作足量的数据集，实验对收集的数十张原始图片首先进行了增强，即扩充了数据集的容量；为了对所有的图片标注其中的缺陷信息，实验中使用了专业的数据标注软件，并在本章中解释了详细的操作；为了将整个数据集划分为需要使用的三类数据集：训练集、检验集和测试集，实验中编写了划分数据集的算法，把所有标注好的数据按 8：1：1 的比例划分成了上述三类，分别执行训练、检验和测试的任务。接下来，本章介绍了在理想状态下，输入的图像在 YOLO v3 网络中的运行，YOLO v3 模型由此获得的更新。最后，本章还说明了，在最终检验模型质量时测试集的图片会通过的标准流程（pipeline），解释如何输出一个边界框，标注出输入图像中的缺陷。

4 发动机表面缺陷检测实验结果分析

4.1 实验环境

本实验所有代码都在 MobaXterm 提供的虚拟器终端上，使用 Ubuntu16.04.4(GNU/Linux 4.4.0-116-generic x86_64)操作系统，利用深度学习框架 Pytorch 进行编辑和运行。硬件环境为 GPU：GeForce RTX 2080，显存 11019Mb。所有程序由 python 脚本语言进行编写和调试。

4.2 模型的训练过程分析

4.2.1 损失函数收敛曲线

在训练的最开始我们采用 YOLO 官方网站提供的预训练权重“darknet53.conv.74”[20],并设置推荐的每批次容量（batch_size）为 8，迭代次数（epochs）为 500，即在初始模型的基础上对训练集进行 500 个迭代周期的训练，每个迭代周期训练训练集中的每张图片一次，并且记录下每个周期下所有批次的 loss 函数的平均值作为该个迭代周期的平均损失函数。

在训练中，loss 函数随迭代周期数的变化曲线可以反映出模型学习进化的速度，当损失函数稳定在一个比较低的值的时候就可以说明这个模型已经训练完成。更具体来说，训练集的损失函数下降的越快意味着模型更新的越快，学习速度也越快；而损失函数稳定以后，即模型学习完成以后，观察它的绝对值越低，说明模型输出结果的精确度越好。另外，实时观察 loss 函数也可以知道网络是否出现了“过拟合”[20]的情况，即损失函数在稳定了一段时间以后开始不降反增。

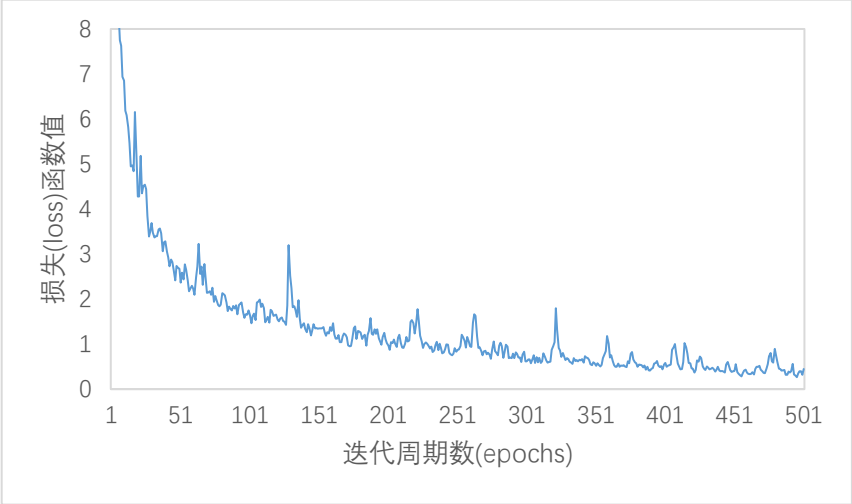


图 4.1 损失函数收敛曲线

从上图 4.1 可见，训练集的损失函数大约在前 150 个周期都在 3 以上，此时模型的参考价值非常的有限，可以想见如果使用 epoch 小于 150 训练出的模型，在测试集或检验集的 AP（average precision）会非常小。在约 400 个迭代周期以后，loss 函数的值基本稳定在了 1 以下，这之后的一段时间它的震荡非常小，但仍然有时有小段的上升，这就是模型遇到了过拟合的情况。在 500 个迭代周期时，实验决定停止训练，因为可以下结论：以官方原始权重训练的模型（以下称之为初始模型），已经达到了最佳表现，损失值约在 0.5 左右，之后的实验优化模型都将基于这个模

型并且加入检验集（valuation set）实验。

4.2.2 模型评价

我们先用检验集来对这个初始模型进行检验，检验集中的每张图片都被标注了可能存在的缺陷类型和所在的位置信息，可以直接与原始图像中的标注做对比，我们可以统计得出预测出的缺陷是否正确，有无漏检以及有无误检，来确定模型的鲁棒性。

表 4.1 初始模型验证集结果

缺陷类别	缺陷数	识别正确数	识别正确率（%）	漏检数	漏检率（%）
线状划痕	124	117	94.3	7	5.7
非线状划痕	45	39	86.7	2	4.4
划痕总计	169	155	91.7	9	5.9
沙眼	89	85	95.5	3	3.5
总计	**258**	**242**	**93.8**	**12**	**4.3**

由表 4.1 可以观察出，划痕和沙眼的识别准确率较高，尤其是划痕的识别正确率在初始模型就能够达到 94.3%，说明线状划痕的特征在算法的学习中是比较明显的。另外线状划痕缺陷的检测还有另外两个特点：其一：识别准确率与漏检率之和为 100%，即没有出现错检的情况，这在我们主观上也比较好理解，毕竟将一根较为平直的线段误认为是一个凹坑是比较困难的；其二，漏检线状划痕大多有共同的几个特征，细而短，颜色与背景相近，大多是跟背景中的刀纹近乎平行的。而非线状的划痕与沙眼之间的互相错检的情况是导致这种缺陷识别准确率下降的重要原因，其中尤其亦将非线状划痕错检为沙眼的情况最为严重，其原因也可以理解，当图像的颜色上干扰较大时，即着色较深时，撞击形成的凹坑有一定的概率被错认为深色的铸造沙眼。

总结，线状划痕特征最为明显但由于其微小的特点可能被忽略，而撞击凹坑和沙眼虽然各自有独特的几何外形，但有时易被混淆。不过，同样的，我们相信经过进一步的训练之后，三种缺陷的各自的特征会被更深刻地学习，正确检测率都将提高。

4.3 关键参数对检测性能的影响

原始模型的准确度尚可接受，但我们希望在此基础上更进一步，通过改变 epochs 和 batch_size 这两个关键的参数观察模型的性能。为了更加量化对一个模型的评价，这里引入两个常用的评价参数标准：

1) 精确度（precision）

即对于每一类别的目标，正确检测的数量与检测出的所有该类别的数量的比值：

$$precision = \frac{t_p}{t_p + f_p} = \frac{t_p}{n} \tag{4.1}$$

其中，t_p, f_p 分别是正确检测数（true positive）和错检数（false positive），n 也就是该类别检测出的总数。

2) 召回率（recall）

即对于每一类别的目标，正确检测的数量与真实的该类别的数量的比值：

$$recall = \frac{t_p}{t_p + f_n} \tag{4.2}$$

其中，t_p,f_n分别是正确检测数（true positive）和漏检数（false negative）。

4.3.1 Epochs 参数的影响

在模型的训练中，Epochs 迭代周期数对模型的性能影响很大，迭代次数不足会导致模型欠拟合，不能充分地学习到每一类目标的特征，从而导致漏检率上升；迭代数太多则会导致模型过分拟合，即对训练集的检测效果好但是对新数据的鲁棒性差，同样导致漏检率上升。本文的对 epochs 迭代数对模型影响的实验中，采取控制变量法，控制 batch_size 参数为推荐的值：8。

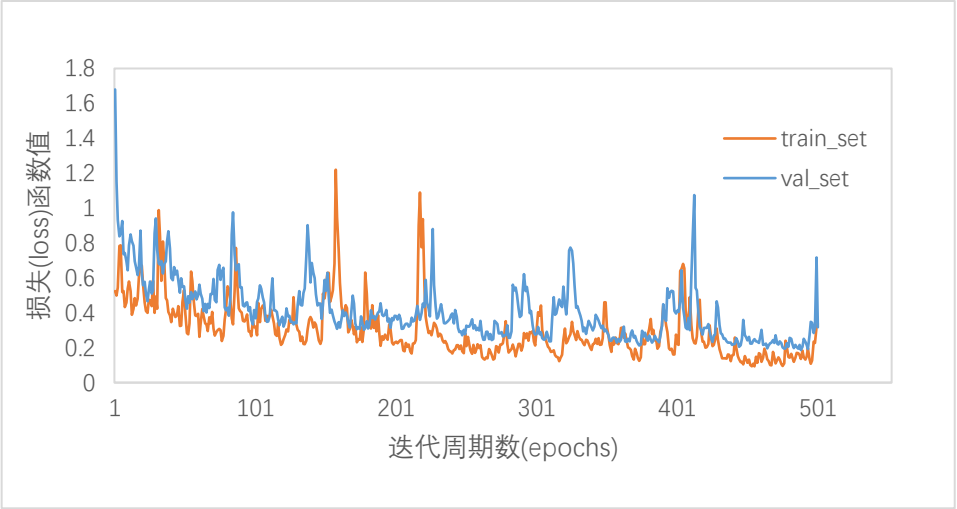


图 4.2 训练原始模型的损失函数收敛曲线

由图 4.2 可知，检验集的损失函数在迭代周期数相同的时候普遍大于训练集的损失函数，这是因为原始模型本来就是由训练集训练而来，与训练集的拟合度非常高，并且观察训练集损失函数曲线的几个异常高峰可以得出结论过拟合现象确实存在且对模型精度影响确实较大。为了研究新数据采用时对模型学习的影响，下面截取 epochs 等于某些特定值时，检验集中检测精确程度的表现。

表 4.2 模型关键性能参数与训练周期的关系

训练周期 epochs	识别正确数(t_p) true positive	召回率(%) recall	错检数(f_p) false positive	精确度(%) precision
50	242	93.8	12	95.3
100	246	95.3	10	96.1
150	244	94.6	13	94.9
200	247	95.7	9	96.5
250	250	96.9	8	96.9
300	248	96.1	8	96.9
350	251	97.3	8	96.9
400	245	95.0	10	96.1
450	252	97.7	7	97.3
500	246	95.3	11	95.7
总缺陷目标数			**258**	

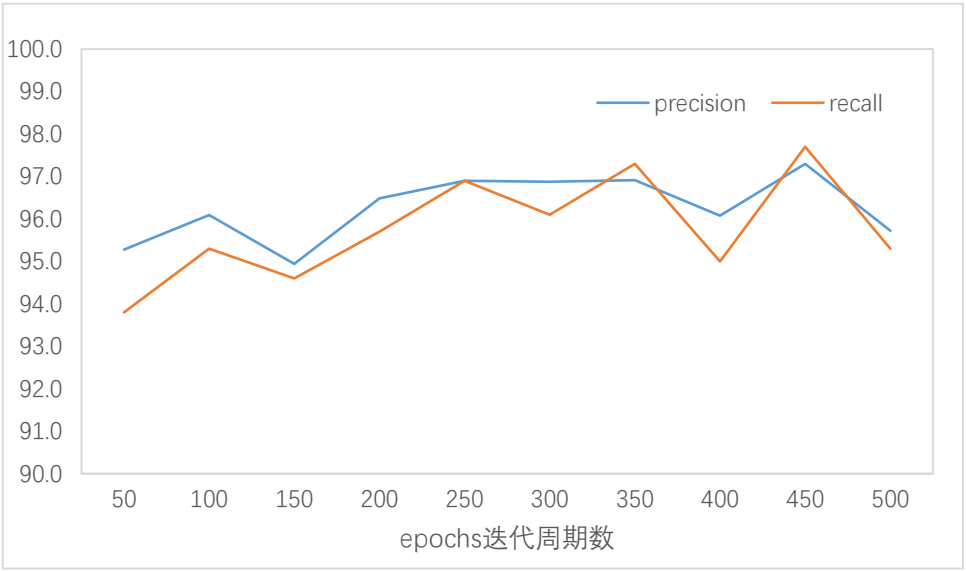
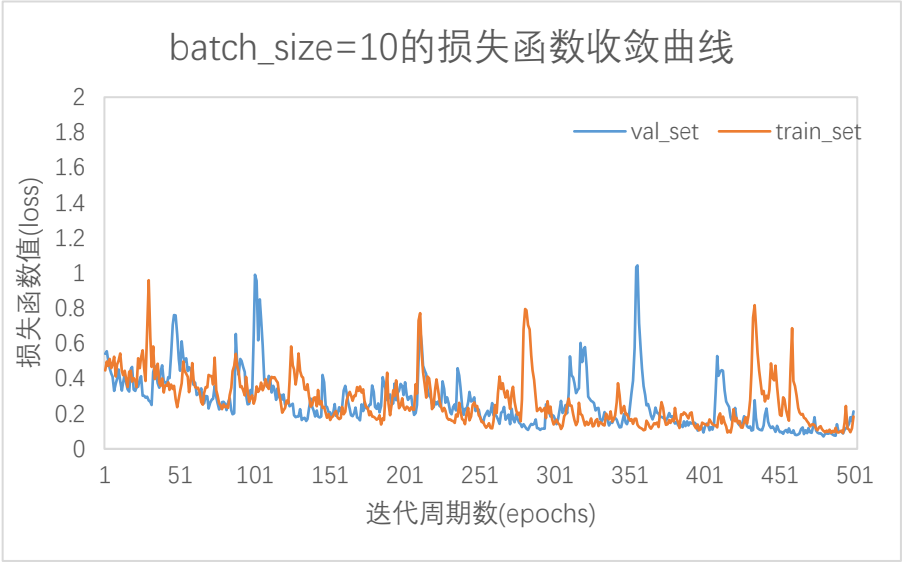


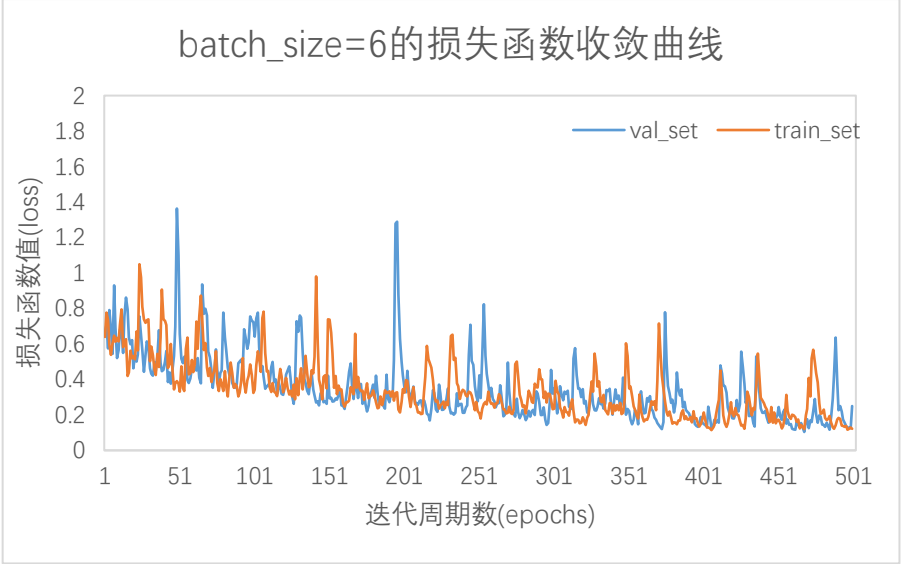
图 4.3 模型关键性能参数与训练周期的关系

由表 4.2 和图 4.3 可见，自迭代周期 200 以后，模型已经基本稳定，epoch=400 左右时的震荡属于过拟合现象，precision 和 recall 都相应的有所下降，这与 epoch=400 左右的损失函数上升是吻合的。据此，本研究发动机表面缺陷的实验模型应在 epoch=300 时拟合效果最好；epoch＜300 时，模型尚处于欠拟合状态，精度没有达到最高；epoch＞300 时，过拟合的情况开始出现并影响模型检测的精度。

4.3.2 Batch_size 参数的影响

Batch_size，即批量大小，是做深度学习时针对神经网络所必须设置的一个参数。对于 YOLO 这种模型结构复杂，训练集容量巨大的算法来说，如何使损失函数下降更快，收敛更迅速，batch_size 的值是需要考虑的。试想，如果不设置 batch_size,训练集中的所有数据将会直接被输入网络，当训练集极大的时候，网络将长时间无法更新，这样就导致了损失函数收敛慢，另外网络同时处理大批量数据对硬件的要求也非常的高，不符合普通的工业生产的环境和条件，以本实验进行的环境为例，GPU 显存为 11019Mb，批量大小处理上仅为 10，即处理 10 张图片后更新一次网络。batch_size 太小也有对模型的负面影响，事实上，每次只处理一张图片是一个特殊的处理方法，叫随机梯度下降算法（Stochastic Gradient Descent, SGD）[27]，虽然这样的处理可以充分学习每个样本的特殊性，但是每张训练样本图得到的权重差距很大，一张张地更新相当于对每一张训练用图片求平均值，反而无法代表每个样本。换句话说，batch_size 过小会使模型对新输入图片的泛化能力下降，最终降低检测精度。





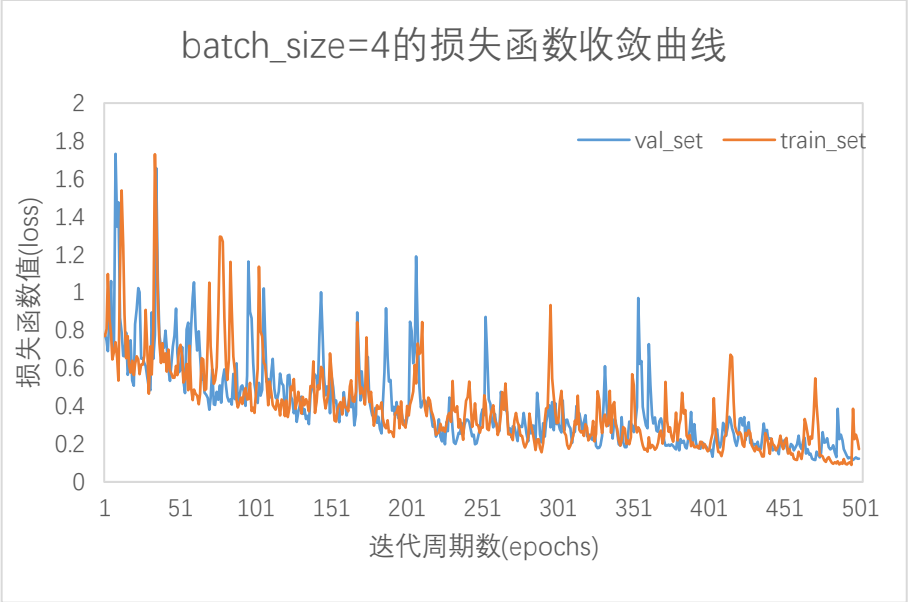


图 4.4 不同的 batch_size 对收敛函数的影响

装 订 线

从图 4.4 和图 4.2 中四种批大小可以看出这个参数对损失函数收敛速度和对模型训练效果的影响。无论是训练集还是检验集，无论 batch_size 的大小，损失函数(loss)的趋势都是下降，当 batch_size 较小（实验中等于 6）时，趋势平滑下降得也更慢，曲线是比较理想的；而当 batch_size 更小（实验中等于 4）时，函数不仅收敛相对缓慢，而且异常的高峰明显变多，这是泛化能力降低的结果，检验集的高峰比训练集更多是因为新数据加入的结果，这也可以证明这个模型的低泛化能力；当 batch_size 很大（实验中为 10）时，只能隐约看出下降趋势，因为函数在前半部分就收敛到了 0.2 左右；收敛的损失函数值平均比其他三种批大小都要低，然而它的异常高峰的峰值也比它们都要高，即使在迭代了 400 多个周期以后，还有损失函数达到 0.8 的情况，而 batch_size 为 4 或 6 时，迭代周期 400 以后，最大损失就不超过 0.65 了。

另外，四种参数设置下，检验集的平均损失函数值都大于训练集的平均损失函数值，但是绝对差距很小，说明无论是那种参数设定下训练出的模型泛化能力都较为理想，只是相对彼此有一些差别。

接下来，取四种参数下训练出的模型(为避开 loss 函数异常高峰，batch_size=4,6,8 的时候,取 epochs=450；batch_size=10 的时候,取 epochs=480)分别进行精确度分析，这里会输出最终的理想模型，所以我们输入的原始图像集为测试集（test_set）：

表 4.3 测试集在 batch_size=4 模型中的表现

模型	缺陷类别	缺陷数	识别正确数	召回率(%)	错检数	精确度(%)
batch_size =4	线状划痕	130	123	94.6	0	100.0
	非线状划痕	44	39	88.6	3	92.9
	划痕总计	174	162	93.1	3	98.2
	沙眼	87	84	96.6	3	96.6
	总计	**261**	**246**	**94.3**	**6**	**97.6**

表 4.4 测试集在 batch_size=6 模型中的表现

模型	缺陷类别	缺陷数	识别正确数	召回率(%)	错检数	精确度(%)
batch_size =6	线状划痕	130	125	96.2	0	100.0
	非线状划痕	44	41	93.2	2	95.3
	划痕总计	174	166	95.4	2	98.8
	沙眼	87	85	97.7	2	97.7
	总计	**261**	**251**	**96.2**	**4**	**98.4**

表 4.5 测试集在 batch_size=8 模型中的表现

模型	缺陷类别	缺陷数	识别正确数	召回率(%)	错检数	精确度(%)
batch_size =8	线状划痕	130	126	96.9	0	100.0
	非线状划痕	44	41	93.2	1	97.6
	划痕总计	174	167	96.0	2	98.8
	沙眼	87	85	97.7	3	96.6
	总计	**261**	**252**	**96.6**	**5**	**98.1**

表 4.6 测试集在 batch_size=10 模型中的表现

模型	缺陷类别	缺陷数	识别正确数	召回率(%)	错检数	精确度(%)
batch_size =10	线状划痕	130	124	95.4	1	99.2
	非线状划痕	44	40	90.9	3	93.0
	划痕总计	174	164	94.3	4	97.6
	沙眼	87	84	96.6	3	96.6
	总计	**261**	**248**	**94.6**	**7**	**97.3**

相较于原始模型，四种模型的召回率和精准度都有上升，批大小为 6 时精确度最高为 98.4%，而批大小为 8 时召回率最高为 96.6%。值得一提的是，四种模型都基本上继承了原始模型线状划痕缺陷错检率为 0 的优点，除 batch_size=10 的模型有一个错检以外，说明模型对线状划痕探测的把握非常大。

4.4 本章小结

本章介绍了，在将一般用的 YOLO v3 检测模型训练成缺陷检测专用模型的过程中，最能代表模型拟合度的参数，模型的损失函数的变化过程。详细的介绍了理想的损失函数的收敛，以及实际曲线收敛时的异常波动出现的时间以及其出现的原因，并初步定性地评价了模型的性能。之后，为了定量地评价模型的准度，实验引入了两个评价模型常用的参数，召回率和精确度。在此二参数的帮助下，实验调试了算法中的两个关键参数 epochs 迭代周期数和 batch_size 批大小，来最大程度地优化模型。最后得出了最佳的模型，该模型设定的参数为 batch_size=8，epochs=450，此时召回率为 96.6%，精确度为 98.1%，可以用来制作原型系统。

装

订

线

5 发动机表面缺陷检测原型系统平台的搭建

可以用来进行检测的 YOLO 模型已经训练完成，且在检测程序检测了测试机的数据后证明了程序的有效性。本文还需要将该程序的操作再简化，搭建一个无需掌握编程知识即能使用的原型系统平台，让操作人员的操作更加直观，以此证明 YOLO v3 算法可以广泛地应用于发动机表面的缺陷检测。

5.1 原型系统的搭建工具

本实验采用 pyqt5 组件来制作交互的 gui 界面。Qt 是 c++语言库，功能是可以访问多种系统的高级 api，包括定位服务，多媒体和蓝牙连接等 UI 开发，而 pyqt 是为了将 Python 语言编写的程序开发为能在各平台（如 Windows 和 Android）上都能运行的应用程序，而设计的一系列 Python 模块。而本实验具体使用了 Qt Designer,通过在 Pycharm 上配置 Qt Designer，实现 gui 的可视化设计。可视化设计的好处就是“所见即所得”，和传统 gui 开发是直接编辑代码不同，Qt Designer 可以可视化地设计好界面，随后将界面以及界面所实现的功能自动转化为代码。另外，Qt Designer 可以让使用者直观地看到可以调动的模块，见图 5.1，省去了设计者记忆模块函数名的时间也防止设计者将函数名混淆而导致程序出现 bug。

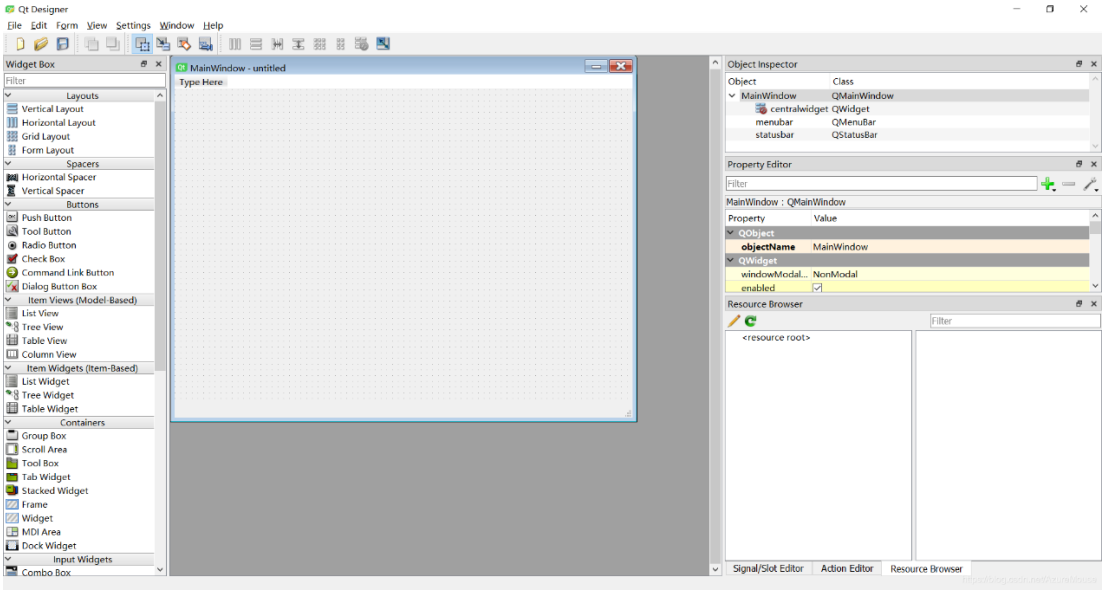


图 5.1 Qt Designer 简明的操作界面

如图 5.1 所示，左侧的 Widget Box 中有各种常用的组件，只需用鼠标拖取就可以使用，它们在窗口中位置也可以通过滑动自由的改变；右上方的 Object Inspector 则是当前 ui 的结构示意，下拉框设计，简单明了；中间的 MainWindow 则是 gui 设计好之后的模样，可以直接呈现在设计者面前。

在实验的 gui 设计完成并转化为 py 文件之后，将继续使用 pyinstaller 组件将 py 文件封装成 windows 可执行文件，即 exe 文件。

5.2 原型系统的搭建过程

5.2.1 开始界面设计

首先，明确实验设计的原型系统所能够实现的功能：通过简单的操作，能够大批量的处理原始图片，并且能够直观地呈现出加注释后的对比，并且将标注好的图片保存起来以便查询。其中，图片对比是整个程序的核心，按照任务要求，实验还应该输出边界框的信息，包括中心点坐标，边界框的宽和高以及置信度，因此这个步骤的界面应该会较大，其中会包含两张图片以及一个文本框；而开始的步骤为了使操作简洁，本文认为界面不应太复杂。据此，实验中，我们会将原型系统分为两个界面：start 和 result。

对于开始界面的输入图像信息，有几种可行的方案：

1） 从指定的文件夹中选择一张图片并输入，在处理完成后再选择另外一张。

2） 选择一个文件夹路径并输入其中的所有图片。

3） 固定一个文件夹路径并输入其中的所有图片。

从指定文件夹中选择一张图片输入并处理的好处是可以让使用者直观地看到最感兴趣的一张图片的输出结果，而这样操作的局限也是显而易见的，在实际的工业生产环境中，会有数条生产线同时运行且每条生产线上都有数个照相机监测生产情况，如果靠值班人员一张张的图片的检测，显然不能满足生产需求。因此，本实验在会 2)3)方案中继续选择。

现在，继续考虑实际的工业生产环境。通常，一个厂房内的所有照相机会通过局域网相连接，并定时上传一批摄取的图像，上传文件的路径一般是固定的，即所有图片最终会聚集到值班人员的电脑的某个指定文件夹中。所以，既然输入的图片一般不会改变存在的路径，我们就没有必要让使用者每次都选择一次文件路径，这样反而是浪费时间的。本系统设计的思想就是，简洁明了，打开就能使用，且能直观地看到结果和比对。因此，我们最终选择 3)号方案。

具体实现的方式则是，在检测程序 detect.py 中使用指定的文件目录接口，使程序一次性输入该文件目录下所有的图片，然后依次检测。具体到代码中的表现为 parser.add_argument("--image_folder",type=str, default="data/samples", help= "path to dataset")，即设置输入文件夹路径为"data/samples"，固定在 data 文件夹下的 samples 文件夹中读入所有的图片。

5.2.2 结果界面设计

实验程序设计的 ui 面板需要让对比效果直观，且文字信息也直观对应，实现的思路如下：

1) 在运行 detect.py 程序时记录下所有图片的名称

2) 将 detect.py 输出的边界框信息与图片名称建立一一对应的关系

3) 输出图片取名与输入图片同名

4) 在 result 界面遍历 1)中记录的图片的名称，分别在输入文件夹(data/samples)和输出文件夹(output/samples)中找到图片，并分别展示在 ui 界面上，再通过 2）中的对应关系输出缺陷检测的信息到图片旁边。

考虑到每次输入的图片量很大，设计上不应让使用者一张张地取保存想要的图片，所以本实验决定采取先统一保存再做预览的处理。对比界面的设计目的应该是让使用者了解程序有没有正

常的运行，以及有没有出现明显的错误，所以对比界面不设“保存”按钮，且该界面会在开始界面的运行进度条读完以后自动跳出，如不需查看也可直接关闭，并不影响标注好的输出图片的保存。

5.2.3 主要设计的代码实现

为了突显最终的标注图片与原始图片的区别，本实验设计 ui 将两张图片并列排放。对于展示图片，我们使用展示图片常用的函数 QLabel()，首先规定展示框的大小，让两个图片框大小相同，随后将已经储存在文件夹中的原始图片和标注图片都缩放到展示框的大小。

在具体代码编写中，首先创建一个 QLabel()：

```
l1=QLabel()
l1.width = 400
l1.height = 400
```

如上，我们为第一个图片框取名为 l1，随后设定它的长宽和高度都为 400。之后，我们将图片放入图片框中：

```
png=QPixmap(result['original_path'])
png = png.scaled(l1.width, l1.height, Qt.IgnoreAspectRatio)
l1.setPixmap(png)
```

其中，QPixmap()即为加载本地图片，括号中的是第一张图片的路径，因为我们载入原始图片的路径是固定的，所以只需要在文件夹的路径最后加上图片的名字即可直接引用，因此我们在程序输出 result 以后就先设定好它的路径，即'original_path'，在这里我们直接引用；同理，生成的标注图片文件夹路径也是固定的，我们命名为'generate_path'。

Png.scaled()即为图片缩放函数，我们将 png 重新规划尺寸到 l1.width, l1.height 的尺寸，Qt.IgnoreAspectRatio 意为不按比例缩放，这样无论长宽比所有的图片都可以缩放到正方形方便我们对比。理论上讲我们的原始图片和标注图片都是正方形的，但是有时候会有边界框及其文字超出原图片尺寸的情况，所以我们加上这一限制。

l1.setPixmap(png)就是将缩放好的 png 放入 l1 展示框中。

类似的，第二个展示框可以写成：

```
l2=QLabel()
l2.width = 400
l2.height = 400
png=QPixmap(result['generate_path'])
png = png.scaled(l2.width, l2.height, Qt.IgnoreAspectRatio)
l2.setPixmap(png)
```

为了将这两个展示框排列在一起，我们再创建一个 3×1 的“表格”，前两个单元格中放置图片，最后一个放置填充缺陷信息的文本框。这里的表格我们调用 QWidget()函数，与展示框函数相似地，我们先规定它的长宽:

```
form1    = QWidget()
```

```
form1.width = 1100
form1.height = 400
```

然后我们排列以上两个展示框：

```
formLayout1 = QHBoxLayout(form1)
formLayout1.addWidget(l1)
formLayout1.addWidget(l2)
```

我们由此得到展示的图片如下：

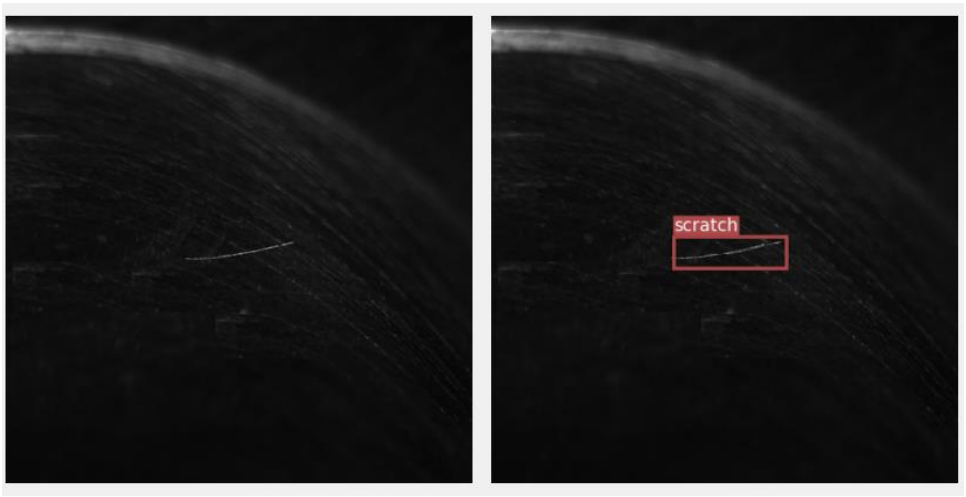


图 5.2 原始图片与标注图片的展示

接下来，我们输出具体的缺陷信息。在预处理输出结果时，我们命名一个 result 结构储存我们所有的数据，一对原始图片和标注图片就对应一个 result。一个 result 中会储存若干个缺陷，我们称之为 detection，每个 detection 结构中又会缺陷类型信息，缺陷位置和置信度。

首先我们创建一个文本展示框，设置好长宽，并设置为只读：

```
l3=QTextEdit()
l3.setReadOnly(True)
l3.setFixedSize(250, 400)
```

随后我们编辑文本内容：

```
text = '缺陷数：%s' % (len(result['detections']))
for detection in result['detections']:
    index +=1
    text += '\n'
    text += '\n' + '缺陷: ' + str(index) + '\n'
    text += '类别: '+ str(detection['label']) + '\n'
    text += '位置: \n
    text += '        x1: %.2f\n' % (detection['position'][0])
    text += '        y1: %.2f\n' % (detection['position'][1])
    text += '        x2: %.2f\n' % (detection['position'][2])
```

```
text += '          y2: %.2f\n' % (detection['position'][3])
text += '置信度: '+ str(detection['conf']) + '\n'
```

其中，label，position 和 conf 即分别代表缺陷类型，位置和置信度。值得一提的是，实验中得到的置信度是一个很精确地小数，为了完整表达，我们将它转换为字符串 string 输出，这样缺陷信息 text 就编辑完成了，然后编写 l3.setText(text)将 text 放置入文本框，效果如下：

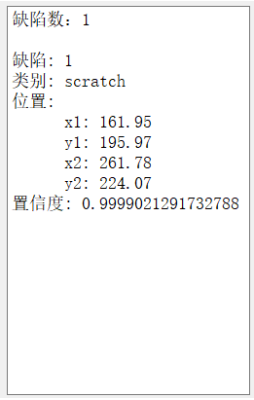


图 5.3 缺陷信息文本框的展示

最后我们将 l3 排列进之前创建的 3×1 表格的最后一格，编写如下：

formLayout1.addWidget(l3)

结果显示界面核心编写就此完成，如下：

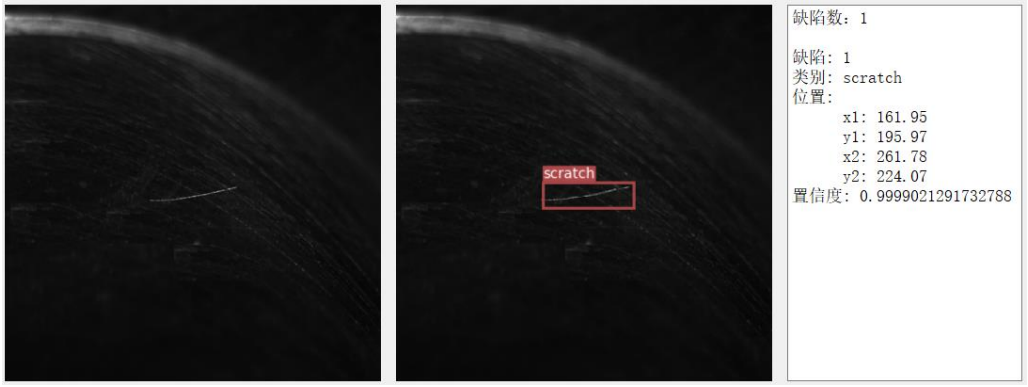


图 5.4 结果显示界面核心内容

5.3 原型系统的功能

所有程序和运行环境都已经打包封装进 ui.exe 直接点击即可使用。

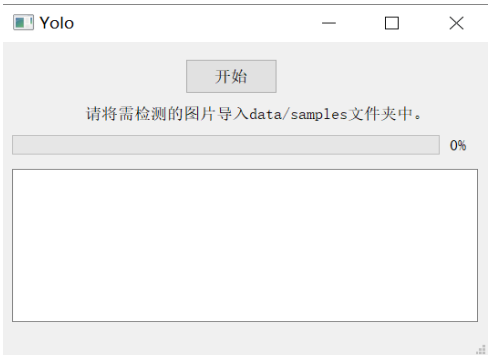


图 5.5 ui.exe 的开始界面

如同设计时的思想一样，使用者无需选择图片路径的文件夹，只需要将输入图片按照指示导入 data/samples 文件夹中，即可让程序开始运行；在实际的工厂运行环境下，则需要将局域网中连接的摄像头图片输出文件路径定为 data/samples 即可。

点击开始之后，进度条将开始读取，同时文本框内出现程序运行信息。程序的运行分为两个部分，即检测图片和标注并保存图片，如图 5.3 所示。在检测图片时，我们设置 detetct.py 中的一个 batch 只包含一张输入图片，而 Inference Timed 对应一个 batch 的检测时间，也就是一张图片的检测时间。在标注图片时，也可以查看每张图片标注的边界框的个数以及缺陷数，和它们分别的置信度。

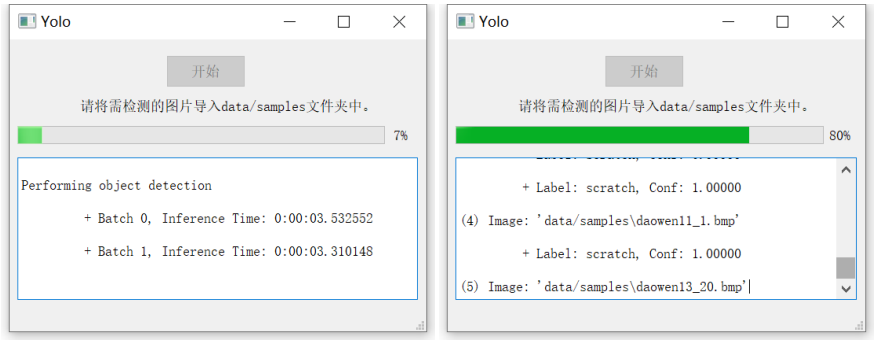


图 5.6 (a)检测图片的开始界面 (b)标注图片中的开始界面

接下来的对比界面由于实验扁平化设计理念的关系，也非常的简洁，对比清晰明了，只设置了翻页的按钮，因为保存操作已经自动完成，如图 5.4 所示。图片与旁边文本框的对应为：缺陷数 n 对应了标注的图里有 n 个边界框，每个缺陷所显示的信息中有类别，也可以在标注好的图片中二次确定。如果不确定文本框中的每一个缺陷的信息具体对应哪一个边界框中的缺陷，可以查看位置信息，x1,y1,x2,y2 分别表示边界框的左上角坐标和右下角坐标，我们定义图片左上角为原点(0,0)，向右向下分别是 x 轴 y 轴的正方向。

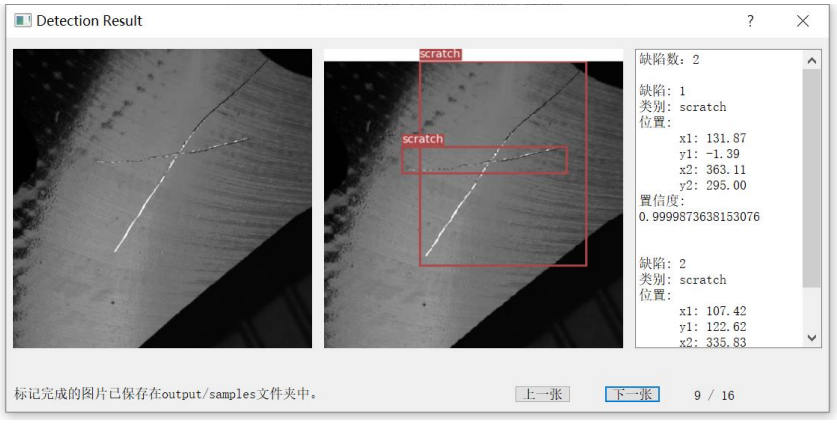


图 5.7 标注完成后的对比界面

所有缺陷的信息都显示在两张图片的旁边，实验设计的目标已经达成。

5.4 本章小结

本实验先明确了原型系统的核心思想，即尽量减少操作人员的操作步骤，实现半自动化的批

量处理原始图片和批量保存标注好的图片这两个功能，并尤其着重考虑了使用者的操作数量越少越好这个因素。如果使用者有需要，系统同时还可以让使用者预览标注结果并进行直观对比。随后通过实用的原型平台搭建工具 pyqt5、pyinstaller 等，最终完成了 ui.exe 的制作，并编写了易懂的教程。

至此，基于 YOLO v3 发动机表面缺陷检测的研究告一段落，内含一个优秀的 YOLO v3 训练出的发动机表面缺陷检测模型的，具有实用价值的原型系统平台已经搭建完成。

6 总结与展望

6.1 论文总结

本文以多样的发动机外表缺陷为研究对象，基于 YOLOv3 设计针对于此问题的创新算法。首先从历史追溯的角度出发，了解缺陷检测方法的发展以及对现在主流方法的启示意义；从发动机表面缺陷产生的原因和由此拥有的特征考虑，结合目前外观缺陷检测的主流方法，深刻分析 YOLO 网络在发动机表面缺陷检测中如何有效应用，最后通过大量的调试提出了一套完整的算法。

第一章为绪论，首先阐述课题的研究背景和意义，内容为发动机表面缺陷的产生原因和对发动机成品的影响；接着查阅并总结了国外缺陷检测的发展现状和深度学习的发展现状；最后是本文主要内容的概述。

第二章是图像目标检测的相关知识以及 YOLO 算法的设计。本章最开始对卷积神经网络的基本原理进行了介绍。接下来详细介绍了 YOLO 算法的基本思想，着重解析 YOLO 统一网络的思想和这样的核心思想带来的好处。最后结合了本文具体的场景案例，即发动机表面缺陷检测，提出要解决的难点和将 YOLO 检测最大化适配到这个场景的办法。

第三章成功建立了适用的数据集，分别进行数据增强，数据标注和数据集划分的步骤。标准的 YOLO 采用 COCO 数据集或是 VOC 数据集训练，其中多为生活中常见物品，在本文讨论的特殊场景中无任何参考价值，因此有必要遵照标准数据集的制作流程制作场景适配的数据集。这部分介绍了如何从一张工业相机拍下的高清 bmp 灰度图，处理成能够进行训练的标准图并附带 YOLO 专门格式的标注信息，并将数据集扩大化和样例多样化。之后继续介绍每一张图片如何经过 YOLO 网络优化整个模型。

第四章最后得出了最佳的模型，该模型设定的参数为 batch_size=8，epochs=450，此时召回率为 96.6%，精确度为 98.1%，可以用来制作原型系统。本章介绍了实际训练时模型的优化过程，以及作为评价模型精度的函数 Loss 函数在收敛时有何特征。本文之后介绍了“端对端”的优化，解释实验中如何通过调试训练程序中的两个关键参数：epoch 迭代周期数和 batch_size 批大小，来优化模型。

第五章完成了原型平台系统的搭建，介绍了在挑选出最佳的模型以后，将检测程序一起封装到适用性广泛的 Windows 应用程序（.exe）中的过程。最后达到的效果是，使用者不需要了解深度学习算法，甚至不需要会 python 程序编程就可进行大批量发动机表面图像的识别工作。本章详述了搭建平台的过程，并介绍了使用来搭建工具。

第六章为总结与展望。对 YOLOv3 应用到发动机表面缺陷的算法创新进行总结，并提出设计的局限性和在此基础上对未来的展望。

6.2 展望

本文的实验也有一些不足之处，比如YOLO v3的具体网络结构，本文只进行了详细的了解和分析，而没有作出一些针对缺陷检测的改良，模型应该还有改进的余地。比如，对于训练的模型本实验都采用416×416×3的输入图片，对于单通道的图片也先预处理为三通道的图片，更进一

步的研究可以继续研究直接在416×416×1的输入上进行卷积，这样对于单通道的图片或许能实现更高的性能。再深入的研究也可以考虑直接修改卷积层的结构，YOLO v3本来的网络舍弃了全部的池化层，而采用大步长的卷积层代替，这样做对一般目标是有利的，但对于特征相对独特的缺陷目标，这样做法的利弊尚有疑问，可以做探讨。

本课题的研究成果可以对以后发动机自动化流水线生产中的实时缺陷检测，提供可参考的程序设计方案，对发动机表面缺陷之外的其他缺陷探测的研究也有借鉴意义。本课题将深度学习中最前沿的算法之一，YOLO v3算法很好结合进了实际的问题，对其他深度学习概念在实际生产生活中的应用也提供可参考的思路。在这个信息自动化发展迅速的时代，本课题为提高我国汽车制造智能化水平贡献了一点绵薄的力量，希望更多的深度学习理念可以尽快融入中国工业的实际生产中，尽快增强国产汽车的国际竞争力，向中国智造这个目标继续迈进，并最终实现科技强国的大目标。

参考文献

[1] D. J. Purll. Automated Surface Inspection With Solid-State Image Sensors[C]// Industrial Applications of Solid State Image Scanners. 1978.

[2] 张浩. 基于深度学习的表面缺陷检测方法研究[D]. 2018.

[3] Yousefi B, Sharifipour H M, Castanedo C I, et al. Automatic IRNDT inspection applying sparse PCA-based clustering[C]// Electrical and Computer Engineering. IEEE, 2017.

[4] 姚忠伟. 基于机器视觉的 PCB 缺陷检测算法研究[D]. 哈尔滨工业大学, 2013.

[5] 李江昀, 任起锐, 张杰,等. 一种基于 Faster R?CNN 网络的金属板带表面缺陷检测方法及装置.

[6] 王宪保，李洁，姚明海，等. 基于深度学习的太阳能电池片表面缺陷检测方法[J]. 模式识别与人工智能, 2014.

[7] Wiesel T N, Hubel D H. Comparison of the effects of unilateral and bilateral eye closure on cortical unit responses in kittens[J]. Journal of Neurophysiology, 1965.

[8] Fukushima K. Neocognitron: A hierarchical neural network capable of visual pattern recognition[J]. Neural Networks, 1988.

[9] 殷瑞刚,魏帅,李晗,等.深度学习中的无监督学习方法综述[J].计算机系统应用, 2016.

[10] Le cun Y. Generalization and Network Design Strategies[C]//. Connectionism in Perspective, 1989.

[11] Le cun Y, Chopra S, Ranzato M, et al. Energy-Based Models in Document Recognition and Computer Vision[C]//. International Conference on Document 53 Analysis and Recognition. 2007.

[12] Hinton G E, Salakhutdinov RR. Reducing the Dimensionality of Data with Neural Networks[J]. Science, 2006.

[13] Deng J ,Dong W, Socher R, et al. Image Net: A large-scale hierarchical image database[J].Proc of IEEE Computer Vision & Pattern Recognition, 2009.

[14] Krizhevsky A, Sutskever I, Hinton G E. Image Net classification with deep convolutional neural networks[C]// International Conference on Neural Information Processing Systems. 2012.

[15] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[J]. 2014.

[16] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.

[17] He K, Zhang X, Ren S, et al. Spatial pyramid pooling in deep convolutional networks for visual recognition[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015, 37(9): 1904-1916.

[18] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition// IEEE Conference on Computer Vision and Pattern Recognition. IEEE Computer Society, 2016.

[19] Redmon J, Farhadi A. Yolov3: An incremental improvement[J]. 2018.

[20] Redmon J , Divvala S , Girshick R , et al. You Only Look Once: Unified, Real-Time Object Detection[J]. 2015.

装 订 线

[21] Hector Torres-Silva, Diego Torres Cabezas. Improved Faster RCNN Object Detection. 2014.

[22] Zeiler M D , Fergus R . Stochastic Pooling for Regularization of Deep Convolutional Neural Networks[J]. Eprint Arxiv, 2013.

[23] Gu J , Wang Z , Kuen J , et al. Recent Advances in Convolutional Neural Networks[J]. Computer Science, 2015.

[24] 蒋相哲. 基于深度学习的铸件外观缺陷检测研究[D].武汉科技大学,2019.

[25] 朱虹兆. 基于退化 YOLO 网络的树脂镜片缺陷识别方法研究[D].哈尔滨理工大学,2019.

[26] Redmon J, Farhadi A. YOLO9000: better, faster, stronger[C]// Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[27] Amari Shun-ichi. Backpropagation and stochastic gradient descent method[J]. Elsevier,1993.

附录

附录 1 数据集的划分程序 divide.py

```
import os
import glob
import random
import shutil

dataset_dir = os.path.join("..", "..", "Data")
train_dir = os.path.join("..", "..", "Data", "train")
valid_dir = os.path.join("..", "..", "Data", "valid")
test_dir = os.path.join("..", "..", "Data", "test")

train_per = 0.8
valid_per = 0.1
test_per = 0.1

def createdir(new_dir):
    if not os.path.exists(new_dir):
        os.createdirs(new_dir)

if __name__ == '__main__':

    for root, dirs, files in os.walk(dataset_dir):
        for sDir in dirs:

            imgs_list = glob.glob(os.path.join(root, sDir, '*.png'))
            random.seed(520)
            random.shuffle(imgs_list) #  随机打乱图片顺序
            imgs_num = len(imgs_list)

            train_point = int(imgs_num * train_per)
            valid_point = int(imgs_num * (train_per + valid_per))

            for i in range(imgs_num):
                if i < train_point:
```

```
            out_dir = os.path.join(train_dir, sDir) #前 80%的图片输出到训练集
        elif i < valid_point:
            out_dir = os.path.join(valid_dir, sDir) #80%-90%的图片输出到训练集
        else:
            out_dir = os.path.join(test_dir, sDir) #后 10%的图片输出到训练集

        createdir(out_dir)

        out_path = os.path.join(out_dir, os.path.split(imgs_list[i])[-1])
        shutil.copy(imgs_list[i], out_path)

    print('Class:{}, train:{}, valid:{}, test:{}'.format(sDir, train_point, valid_point-train_point, imgs_num-valid_point))
```

装

附录 2 训练程序 train.py

```
from __future__ import division

from models import *
from utils.logger import *
from utils.utils import *
from utils.datasets import *
from utils.parse_config import *
from test import evaluate

from terminaltables import AsciiTable

import os
os.environ['CUDA_VISIBLE_DEVICES']='3'
import sys
import time
import datetime
import argparse

import torch
from torch.utils.data import DataLoader
from torchvision import datasets
```

订

线

```
from torchvision import transforms
from torch.autograd import Variable
import torch.optim as optim

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--name", type=str, default="next", help="add your model name")
    parser.add_argument("--epochs", type=int, default=100, help="number of epochs")
    parser.add_argument("--batch_size", type=int, default=8, help="size of each image batch")
    parser.add_argument("--gradient_accumulations", type=int, default=2, help="number of gradient accums before step")
    parser.add_argument("--model_def", type=str, default="config/yolov3-custom.cfg", help="path to model definition file")
    parser.add_argument("--data_config", type=str, default="config/custom.data", help="path to data config file")
    parser.add_argument("--pretrained_weights", type=str, help="if specified starts from checkpoint model")
    parser.add_argument("--n_cpu", type=int, default=8, help="number of cpu threads to use during batch generation")
    parser.add_argument("--img_size", type=int, default=416, help="size of each image dimension")
    parser.add_argument("--checkpoint_interval", type=int, default=1, help="interval between saving model weights")
    parser.add_argument("--evaluation_interval", type=int, default=1, help="interval evaluations on validation set")
    parser.add_argument("--compute_map", default=False, help="if True computes mAP every tenth batch")
    parser.add_argument("--multiscale_training", default=True, help="allow for multi-scale training")
    opt = parser.parse_args()
    print(opt)

    logger = Logger("logs")
    print("cuda",torch.cuda.is_available())
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    os.makedirs("output", exist_ok=True)
```

```
os.makedirs("checkpoints", exist_ok=True)

# Get data configuration
data_config = parse_data_config(opt.data_config)
train_path = data_config["train"]
valid_path = data_config["valid"]
class_names = load_classes(data_config["names"])

# Initiate model
model = Darknet(opt.model_def).to(device)
model.apply(weights_init_normal)

# If specified we start from checkpoint
if opt.pretrained_weights:
    if opt.pretrained_weights.endswith(".pth"):
        model.load_state_dict(torch.load(opt.pretrained_weights))
    else:
        model.load_darknet_weights(opt.pretrained_weights)

# Get dataloader
#print("train_path",train_path)
dataset = ListDataset(train_path, augment=True, multiscale=opt.multiscale_training)
#print("dataset",dataset)
dataloader = torch.utils.data.DataLoader(
    dataset,
    batch_size=opt.batch_size,
    shuffle=True,
    num_workers=opt.n_cpu,
    pin_memory=True,
    collate_fn=dataset.collate_fn,
)

optimizer = torch.optim.Adam(model.parameters())

metrics = [
    "grid_size",
    "loss",
```

```
            "x",
            "y",
            "w",
            "h",
            "conf",
            "cls",
            "cls_acc",
            "recall50",
            "recall75",
            "precision",
            "conf_obj",
            "conf_noobj",
        ]

        for epoch in range(opt.epochs):
            model.train()
            start_time = time.time()
            #print("batches_done========", len(dataloader))
            for batch_i, (_, imgs, targets) in enumerate(dataloader):
                batches_done = len(dataloader) * epoch + batch_i

                imgs = Variable(imgs.to(device))
                targets = Variable(targets.to(device), requires_grad=False)

                loss, outputs = model(imgs, targets)
                loss.backward()

                if batches_done % opt.gradient_accumulations:

                    optimizer.step()
                    optimizer.zero_grad()

                log_str = "\n---- [Epoch %d/%d, Batch %d/%d] ----\n" % (epoch, opt.epochs, batch_i,
len(dataloader))

                metric_table = [["Metrics", *[f"YOLO Layer {i}" for i in
range(len(model.yolo_layers))]]]
```

```
                    # Log metrics at each YOLO layer
                    for i, metric in enumerate(metrics):
                        formats = {m: "%.6f" for m in metrics}
                        formats["grid_size"] = "%2d"
                        formats["cls_acc"] = "%.2f%%"
                        row_metrics = [formats[metric] % yolo.metrics.get(metric, 0) for yolo in model.yolo_layers]
                        metric_table += [[metric, *row_metrics]]

                        # Tensorboard logging
                        tensorboard_log = []
                        for j, yolo in enumerate(model.yolo_layers):
                            for name, metric in yolo.metrics.items():
                                if name != "grid_size":
                                    tensorboard_log += [(f"{name}_{j+1}", metric)]
                        tensorboard_log += [("loss", loss.item())]
                        logger.list_of_scalars_summary(tensorboard_log, batches_done)

                    log_str += AsciiTable(metric_table).table
                    log_str += f"\nTotal loss {loss.item()}"
                    # Determine approximate time left for epoch
                    epoch_batches_left = len(dataloader) - (batch_i + 1)
                    time_left = datetime.timedelta(seconds=epoch_batches_left * (time.time() - start_time) / (batch_i + 1))
                    log_str += f"\n---- ETA {time_left}"

                    print(log_str)

                    model.seen += imgs.size(0)

                if False: #epoch % opt.evaluation_interval == 0:
                    print("\n---- Evaluating Model ----")
                    # Evaluate the model on the validation set
                    precision, recall, AP, f1, ap_class = evaluate(
                        model,
                        path=valid_path,
```

```
            iou_thres=0.5,
            conf_thres=0.5,
            nms_thres=0.5,
            img_size=opt.img_size,
            batch_size=8,
        )
        evaluation_metrics = [
            ("val_precision", precision.mean()),
            ("val_recall", recall.mean()),
            ("val_mAP", AP.mean()),
            ("val_f1", f1.mean()),
        ]
        logger.list_of_scalars_summary(evaluation_metrics, epoch)
        print("evaluate done ============here")
        # Print class APs and mAP
        ap_table = [["Index", "Class name", "AP"]]
        for i, c in enumerate(ap_class):
            ap_table += [[c, class_names[c], "%.5f" % AP[i]]]
        print(AsciiTable(ap_table).table)
        print(f"---- mAP {AP.mean()}")


    torch.save(model.state_dict(), f"checkpoints/yolov3_c%s.pth" % opt.name)
```

附录 3 检测程序 detect.py

```
from __future__ import division

from models import *
from utils.utils import *
from utils.datasets import *

import os
import sys
import time
import datetime
import argparse
```

```
import math

from PIL import Image

import torch
from torch.utils.data import DataLoader
from torchvision import datasets
from torch.autograd import Variable

import matplotlib.pyplot as plt
import matplotlib.patches as patches
from matplotlib.ticker import NullLocator

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--image_folder", type=str, default="data/samples", help="path to dataset")
    parser.add_argument("--model_def", type=str, default="config/yolov3-custom.cfg", help="path to model definition file")
    parser.add_argument("--weights_path", type=str, default="checkpoints/yolov3_c.pth", help="path to weights file")
    parser.add_argument("--class_path", type=str, default="data/custom/classes.names", help="path to class label file")
    parser.add_argument("--conf_thres", type=float, default=0.8, help="object confidence threshold")
    parser.add_argument("--nms_thres", type=float, default=0.4, help="iou thresshold for non-maximum suppression")
    parser.add_argument("--batch_size", type=int, default=1, help="size of the batches")
    parser.add_argument("--n_cpu", type=int, default=0, help="number of cpu threads to use during batch generation")
    parser.add_argument("--img_size", type=int, default=416, help="size of each image dimension")
    parser.add_argument("--checkpoint_model", type=str, help="path to checkpoint model")
    opt = parser.parse_args()
    #print(opt)

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

```
os.makedirs("output", exist_ok=True)

# Set up model
model = Darknet(opt.model_def, img_size=opt.img_size).to(device)

if opt.weights_path.endswith(".weights"):
    # Load darknet weights
    model.load_darknet_weights(opt.weights_path)
else:
    # Load checkpoint weights
    model.load_state_dict(torch.load(opt.weights_path))

model.eval()  # Set in evaluation mode
#dataset = ListDataset(train_path, augment=True, multiscale=opt.multiscale_training)
dataloader = DataLoader(
    ImageFolder(opt.image_folder, img_size=opt.img_size),
    batch_size=opt.batch_size,
    shuffle=False,
    num_workers=opt.n_cpu,
)

# dataloader = torch.utils.data.DataLoader(
#     dataset,
#     batch_size=opt.batch_size,
#     shuffle=False,
#     num_workers=opt.n_cpu,
#     #collate_fn=dataset.collate_fn,
# )

classes = load_classes(opt.class_path)  # Extracts class labels from file
#print("classes", classes)
Tensor = torch.cuda.FloatTensor if torch.cuda.is_available() else torch.FloatTensor

imgs = []  # Stores image paths
img_detections = []  # Stores detections for each image index
```

```
print("\nPerforming object detection:")
prev_time = time.time()
for batch_i, (img_paths, input_imgs) in enumerate(dataloader):
    # Configure input
    input_imgs = Variable(input_imgs.type(Tensor))
    #print(input_imgs.shape)
    # Get detections
    with torch.no_grad():
        detections = model(input_imgs)
        detections = non_max_suppression(detections, opt.conf_thres, opt.nms_thres)

    # Log progress
    current_time = time.time()
    inference_time = datetime.timedelta(seconds=current_time - prev_time)
    prev_time = current_time
    print("\t+ Batch %d, Inference Time: %s" % (batch_i, inference_time))

    # Save image and detections
    imgs.extend(img_paths)
    img_detections.extend(detections)

# Bounding-box colors
cmap = plt.get_cmap("tab20b")
colors = [cmap(i) for i in np.linspace(0, 1, 20)]

print("\nSaving images:")
# Iterate through images and save plot of detections
print("dectected==================",img_detections)
for img_i, (path, detections) in enumerate(zip(imgs, img_detections)):

    print("(%d) Image: '%s'" % (img_i, path))

    # Create plot
    img = np.array(Image.open(path))
    plt.figure()
    fig, ax = plt.subplots(1)
    ax.imshow(img)
```

```
# Draw bounding boxes and labels of detections
if detections is not None:
    # Rescale boxes to original image
    detections = rescale_boxes(detections, opt.img_size, img.shape[:2])
    unique_labels = detections[:, -1].cpu().unique()
    n_cls_preds = len(unique_labels)
    bbox_colors = random.sample(colors, n_cls_preds)

    # filter according to motor-scratch case
    last_x1 = -1
    last_x2 = -1
    last_y1 = -1
    last_y2 = -1
    for x1, y1, x2, y2, conf, cls_conf, cls_pred in detections:

        if conf<0.9:
            continue

        box_w = x2 - x1
        box_h = y2 - y1

        if last_x1!=-1:
            last_ctr_x = (last_x1 + last_x2)/2
            last_ctr_y = (last_y1 + last_y2)/2
            ctr_x = (x1 + x2)/2
            ctr_y = (y1 + y2)/2
            slope_ctr = math.atan((ctr_y - last_ctr_y)/(ctr_x - last_ctr_x))
            slope_last_1 = math.atan((last_y1-last_y2)/(last_x1-last_x2))
            slope_last_2 = math.pi - slope_last_1

            if abs(slope_ctr - slope_last_1)<0.18 or abs(slope_ctr - slope_last_2)<0.18:
                continue
            if abs(slope_ctr - slope_last_1)<0.4 or abs(slope_ctr - slope_last_2)<0.4:
                if abs(ctr_x - last_ctr_x)<0.5*abs(last_x1 - last_x2) and abs(ctr_y - last_ctr_y)<0.5*abs(last_y1 - last_y2):
                    continue
```

```
                if x1>last_x1 - 0.5*last_box_w and x2<last_x2 + 0.5*last_box_w:
                    if y1>last_y1 - 0.5*last_box_h and y2<last_y2 + 0.5*last_box_h:
                        continue

                print("\t+ Label: %s, Conf: %.5f" % (classes[int(cls_pred)], cls_conf.item()))



                color = bbox_colors[int(np.where(unique_labels == int(cls_pred))[0])]
                # Create a Rectangle patch
                bbox = patches.Rectangle((x1, y1), box_w, box_h, linewidth=2, edgecolor=color, facecolor="none")
                # Add the bbox to the plot
                ax.add_patch(bbox)
                # Add label
                plt.text(
                    x1,
                    y1-18,
                    s=classes[int(cls_pred)],
                    color="white",
                    verticalalignment="top",
                    bbox={"color": color, "pad": 0},
                )
                last_x1 = x1
                last_x2 = x2
                last_y1 = y1
                last_y2 = y2
                last_box_w = box_w
                last_box_h = box_h


        # Save generated image with detections
        plt.axis("off")
        plt.gca().xaxis.set_major_locator(NullLocator())
        plt.gca().yaxis.set_major_locator(NullLocator())
        filename = path.split("/")[-1].split(".")[0]
        plt.savefig(f"output/{filename}.png", bbox_inches="tight", pad_inches=0.0)
```

```
        plt.close()
```

附录 4 开始界面 ui start_ui.py

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'start_ui.ui'
#
# Created by: PyQt5 UI code generator 5.14.1
#
# WARNING! All changes made in this file will be lost!

from PyQt5 import QtCore, QtGui, QtWidgets
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(532, 371)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.pushButton_start = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_start.setGeometry(QtCore.QRect(200, 20, 101, 41))
        self.pushButton_start.setObjectName("pushButton_start")
        self.progressBar = QtWidgets.QProgressBar(self.centralwidget)
        self.progressBar.setGeometry(QtCore.QRect(10, 110, 511, 23))
        self.progressBar.setProperty("value", 0)
        self.progressBar.setObjectName("progressBar")
        self.textEdit = QtWidgets.QTextEdit(self.centralwidget)
        self.textEdit.setGeometry(QtCore.QRect(10, 150, 511, 181))
        self.textEdit.setObjectName("textEdit")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(90, 69, 411, 31))
        self.label.setObjectName("label")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 532, 18))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
```

```
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        self.pushButton_start.clicked.connect(MainWindow.start)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Yolo"))
        self.pushButton_start.setText(_translate("MainWindow", "start"))
        self.label.setText(_translate("MainWindow", "请将需检测的图片导入 data/samples 文
件夹中。"))
```

装

附录 5 结果界面 ui result_ui.py

订

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'result_ui.ui'
#
# Created by: PyQt5 UI code generator 5.14.1
#
# WARNING! All changes made in this file will be lost!


from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_Dialog_result(object):
    def setupUi(self, Dialog_result):
        Dialog_result.setObjectName("Dialog_result")
        Dialog_result.setWindowModality(QtCore.Qt.ApplicationModal)
        Dialog_result.resize(1100, 500)
        self.stackedWidget = QtWidgets.QStackedWidget(Dialog_result)
        self.stackedWidget.setGeometry(QtCore.QRect(0, 0, 1100, 430))
        self.stackedWidget.setObjectName("stackedWidget")
```

线

```
        self.pushButton_pre = QtWidgets.QPushButton(Dialog_result)
        self.pushButton_pre.setEnabled(False)
        self.pushButton_pre.setGeometry(QtCore.QRect(680, 465, 75, 23))
        self.pushButton_pre.setObjectName("pushButton_pre")
        self.pushButton_next = QtWidgets.QPushButton(Dialog_result)
        self.pushButton_next.setEnabled(False)
        self.pushButton_next.setGeometry(QtCore.QRect(800, 465, 75, 23))
        self.pushButton_next.setObjectName("pushButton_next")
        self.label = QtWidgets.QLabel(Dialog_result)
        self.label.setGeometry(QtCore.QRect(920, 470, 111, 16))
        self.label.setText("")
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(Dialog_result)
        self.label_2.setGeometry(QtCore.QRect(10, 465, 511, 21))
        self.label_2.setObjectName("label_2")

        self.retranslateUi(Dialog_result)
        self.stackedWidget.setCurrentIndex(-1)
        self.pushButton_pre.clicked.connect(Dialog_result.click_pre)
        self.pushButton_next.clicked.connect(Dialog_result.click_next)
        QtCore.QMetaObject.connectSlotsByName(Dialog_result)

    def retranslateUi(self, Dialog_result):
        _translate = QtCore.QCoreApplication.translate
        Dialog_result.setWindowTitle(_translate("Dialog_result", "Detection Result"))
        self.pushButton_pre.setText(_translate("Dialog_result", "上一张"))
        self.pushButton_next.setText(_translate("Dialog_result", "下一张"))
        self.label_2.setText(_translate("Dialog_result", "标记完成的图片已保存在output/samples 文件夹中。"))
```

附录 6 ui 主程序 ui.py

```
# -*- coding: utf-8 -*-

import sys

# 加载 main.py 中的 Ui_MainWindow
from start_ui import Ui_MainWindow
from result_ui import Ui_Dialog_result

from PyQt5.QtWidgets import QDialog, QMainWindow, QApplication, QLabel, QWidget, QHBoxLayout, QTextEdit
from PyQt5.QtCore import QObject, pyqtSignal, QThread, Qt
from PyQt5.QtGui import QPixmap

from predict import detect

class Worker(QObject):
    finished = pyqtSignal()
    showResult = pyqtSignal(dict)
    intReady = pyqtSignal(int)
    printLog = pyqtSignal(str)

    def work(self): # A slot takes no params
        arrData = detect(self)
        """
        for i in range(0, 5):
            print('worker: ' + str(i))
            self.intReady.emit(i)
            time.sleep(5 + random.randint(0, 9))

        """
        self.showResult.emit(arrData)
        self.finished.emit()
        print('worker finished')

class Result(QDialog, Ui_Dialog_result):
```

```
        """
        """

    def __init__(self, parent=None, data=None):
        super(Result, self).__init__()

        self.setupUi(self)


        self.arrData = data
        self.drawData()


    def click_pre(self):
        """
        """
        self.page_index -= 1
        self.refresh_top()

        self.stackedWidget.setCurrentIndex(self.page_index)

    def click_next(self):
        """
        """
        self.page_index += 1
        self.refresh_top()

        self.stackedWidget.setCurrentIndex(self.page_index)

    def refresh_top(self):
        """
        """
        self.label.setText('%s / %s' % (self.page_index + 1, self.page_count))
        if self.page_count == 0:
            self.pushButton_pre.setEnabled(False)
            self.pushButton_next.setEnabled(False)
            return
```

```
        if self.page_index == 0:
            self.pushButton_pre.setEnabled(False)
            self.pushButton_next.setEnabled(True)

            return

        if self.page_index == self.page_count - 1:
            self.pushButton_pre.setEnabled(True)
            self.pushButton_next.setEnabled(False)
            return

        self.pushButton_pre.setEnabled(True)
        self.pushButton_next.setEnabled(True)

    def drawData(self):
        """
        """
        arrResult = self.arrData['result']

        self.page_count = len(arrResult)
        self.page_index = 0

        self.refresh_top()


        for result in arrResult:

            # 创建 QLabel 用于展示  原始图片
            l1=QLabel()
            l1.width = 400
            l1.height = 400

            # QPixmap  加载本地图片
            png=QPixmap(result['original_path'])

            # 图片缩放
```

```
                png = png.scaled(l1.width, l1.height, Qt.IgnoreAspectRatio,
Qt.SmoothTransformation)
                # 图片放到 QLabel 里面
                l1.setPixmap(png)

                # 创建 QLabel 用于展示 生成的图片
                l2=QLabel()
                l2.width = 400
                l2.height = 400

                # QPixmap 加载本地图片
                png=QPixmap(result['generate_path'])

                # 图片缩放
                png = png.scaled(l2.width, l2.height, Qt.IgnoreAspectRatio,
Qt.SmoothTransformation)
                # 图片放到 QLabel 里面
                l2.setPixmap(png)

                # self.gridLayout.addWidget(l1, row, 0)
                # self.gridLayout.addWidget(l2, row, 1)

                l3=QTextEdit()
                l3.setReadOnly(True)
                # l3.width = 250
                # l3.height = 400
                l3.setFixedSize(250, 400)
                l3.setAlignment(Qt.AlignTop)
                text = '缺陷数：%s' % (len(result['detections']))

                index = 0
                for detection in result['detections']:
                    index += 1
                    text += '\n'
                    text += '\n' + '缺陷: ' + str(index) + '\n'
                    text += '类别: '+ str(detection['label']) + '\n'
                    text += '位置: \n'
```

装 订 线

```
                text += '          x1: %.2f\n' % (detection['position'][0])
                text += '          y1: %.2f\n' % (detection['position'][1])
                text += '          x2: %.2f\n' % (detection['position'][2])
                text += '          y2: %.2f\n' % (detection['position'][3])
                text += '置信度: '+ str(detection['conf']) + '\n'

            l3.setText(text)

            form1    = QWidget()
            form1.width = 1100
            form1.height = 400
            formLayout1 = QHBoxLayout(form1)
            formLayout1.addWidget(l1)
            formLayout1.addWidget(l2)
            formLayout1.addWidget(l3)

            self.stackedWidget.addWidget(form1)
            """
            print('form1: %sx%s' %(form1.width, form1.height))
            print('l1: %sx%s' %(l1.width, l1.height))
            print('l2: %sx%s' %(l2.width, l2.height))
            print('l3: %sx%s' %(l3.width, l3.height))
            """



# 继承于 QMainWindow 和 Ui_MainWindow
class UI(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(UI, self).__init__()


        # 调用父类的 setupUi
        self.setupUi(self)

        # 1 - create Worker and Thread inside the Form
        self.worker = Worker()
```

```
        self.thread = QThread()
        self.worker.moveToThread(self.thread)

        self.worker.intReady.connect(self.refreshProgress)
        self.worker.printLog.connect(self.printLog)
        self.worker.finished.connect(self.thread.quit)
        self.worker.showResult.connect(self.showResult)

        self.thread.started.connect(self.worker.work)


    def start(self):
        """
        when start button clicked.
        """
        self.thread.start()
        self.pushButton_start.setEnabled(False)

    def showResult(self, arrData):
        """
        show result
        """
        self.textEdit.append("\ndetection finish and prepare to open result window")
        self.dialog = Result(parent=self, data=arrData)
        self.dialog.show()

    def refreshProgress(self, i):
        """
        更新进度条
        """
        self.progressBar.setProperty('value', i)

    def printLog(self, log):
        """
        print log to label.
        """
        self.textEdit.append('\n' + log)
```

```
# 程序入口
if __name__ == '__main__':
    # 实例化一个 app
    app = QApplication(sys.argv)
    # 实例化
    bb = UI()
    # 展示
    bb.show()
    # app 开始运行
    app.exit(app.exec_())
```

装

订

线

谢辞

至此，基于 YOLO v3 的发动机表面缺陷检测研究就告一段落了。行文至此，正是 2020 年 5 月 20 日，身在意大利，我百感交集。从 19 年暑假开始就与我敬爱的徐立云导师讨论了这个项目的可操作性，开始做了一些准备，当时与学姐讨论这个课题的时候，其实已经预想到了困难出现的诸多不易，但真实地一路走来，我的心中其实有更多的感慨。2020 年初，Covid-2019 新冠疫情爆发，意大利是中国外首当其冲的重灾区，作为出国的学生，留学在意，不仅是身体状况要时刻担心，与国内的家人，朋友的分离所带来的忧虑与不安也时刻困扰着我，这种情况下，支持着做完这个毕业设计并完成在意留学的人们我都铭记在心。

首先，我要再次感谢徐立云老师和李博宇学姐对我的关心和照顾，意大利与中国有 6 个小时的时差，课题沟通和问题讨论都因此有一些困难，但我始终可以得到他们耐心的问题回答；尤其还要感谢徐立云老师在疫情期间给我的生活上的关心，着实令人感动。然后，我也要感谢在意大利留学期间与我同吃同住的中意班同班同学，徐宏玮同学和陈悦敏同学，特殊时期大家都不能出门，是我们互相支持互相开导才一起度过了这些不易的日子，还留下了一些开心的回忆。另外一个重要的人是我高中时就认识的好兄弟，现在在香港大学理学院就读的马佳成同学，作为一个 python 方面的初学者，马同学给了我很多帮助，指导我调试程序，配置环境和使用服务器等。当然，我还要感谢我的家人，虽然相隔千里，对我每天的关心是实实在在的，温暖至致的

YOLO v3 确实是目前最近先进的算法之一，它的原始论文不过被提出两三年而已，能够借助这次机会接触到这么前沿的知识，我也是心怀感激，感激我能在同济大学，米兰理工大学这样优秀的大学就读。接下来，我会前往卡耐基梅隆大学继续深造机械工程，希望我能够利用好这样的机会，认真学习，不辜负学校和社会的期望，争取以后也能对祖国的建设贡献力量。

最后，对在百忙之中能对我的论文提出宝贵意见的各位评审老师表示由衷的感谢！

装 订 线