中山大學 智能工程学院
SUN YAT-SEN UNIVERSITY SCHOOL OF INTELLIGENT SYSTEMS ENGINEERING

Sun Yat-sen University

School of Intelligent Systems Engineering

脊柱疾病智能诊断项目报告

Author:

19351003 蔡炼楷

19351083 林良涛

19351103 马　梁

19351115 谭源正

19351173 张礼霆

19351183 郑　均

19351185 郑　晔

Supervisor:

赵屾　副教授

Deep Learning

May 31,2021

目 录

1 项目介绍 **3**

2 问题分析 **3**

2.1 数据集 3

2.2 模型 5

3 模型原理 **6**

3.1 YOLOv5 6

3.1.1 输入端 7

3.1.2 Backbone 9

3.1.3 Neck 10

3.1.4 输出端 11

3.1.5 YOLOv5 四种网络的比较 12

3.2 ResNet50 13

3.2.1 ResNet 网络结构 13

3.2.2 ResNet 核心理论 13

3.2.3 ResNet 的网络设计规律 16

3.2.4 ResNet50 的 backbone 部分结构 16

4 实验与分析 **17**

4.1 数据清洗与数据增强 17

4.1.1 数据集划分 17

4.1.2 重定义数据集 18

4.1.3 数据增强 20

4.2 目标检测与分类 20

4.3 目标检测 20

4.3.1 区域回归 21

4.3.2 点回归 23

4.4 目标分类 23
4.4.1 ResNet50 的二分类任务 23
4.4.2 ResNet50 的七分类任务 24
4.4.3 其他尝试 25
4.5 实验结果 25
4.6 案例可视化 27

5 总结 **28**
5.1 不足 28
5.2 分工 29

1 项目介绍

脊柱影像是诊断脊柱疾病的重要依据。由此，我们尝试在 MR 图像上利用专业所长做出一些贡献，希求训练一个能辅助骨科医生定位并重点关注影像形态不正常椎间盘的神经网络模型。

本项目通过 YOLO5 和 ResNet50 训练脊椎和椎间盘的检测和分类模型。对给定的脊椎图像，检测时，通过 YOLO5 定位出对应的椎骨或椎间盘所在的区域，并回归到中心点，对每个中心点，根据相邻点之间的纵坐标之差形成框，框选出骨骼区域，并对其进行位置分类；分类时，根据上述框选出的区域，通过 ResNet50 进行椎骨和椎间盘的分类及病理检测。

本模型中，对给定的 200 份病人图像集，挑选 150 份作为训练集，50 份作为测试集。最终测试集结果如下：

目标检测 准确率	二分类 准确率	七分类 准确率	recall	precision	AP
0.8780	0.9982	0.6248	0.6667	0.4022	0.6574

图 1: 最终结果

代码地址：

https://github.com/ZhengJun-AI/Spinal-Disease-Detection

环境要求：

- python 3.8
- pytorch >= 1.7.0
- GPU 显存 >= 4G

2 问题分析

2.1 数据集

首先，对给定的数据集进行分析。

数据集中共有 200 份病人图像集。每份病人图像集中有 30 到 120 张不等的脊椎 MR 图像，其中，绝大部分图像集的图像数量在 40 张到 70 张范围内。

对每份图像集，包括脊椎的 T1 和 T2 矢状面影像，以及 T2 轴状位影像。给定的标注文件在每个数据集中选择一张矢状面影像，对影像中的椎骨和椎间盘进行中心点位置和病理类型标记，标注类型为点。

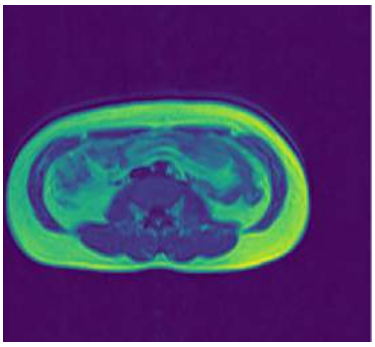


图 2: T2 轴状面

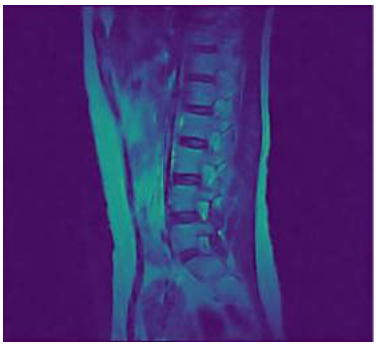


图 3: T1 矢状位

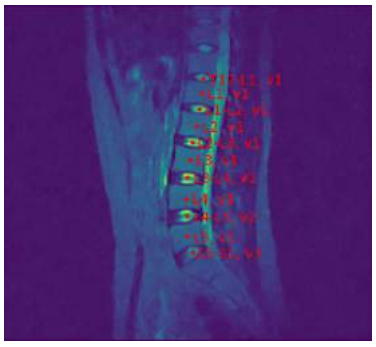


图 4: 有标注的 T2 矢状位

易得，每份图像集中，除标注图像外，应存在一定数量的其他未标注的有用图像。对标签图像进行观察统计，发现，大部分 MR 图像呈现正常且清晰，但存在部分图像亮度较低，并有一张图像出现颜色偏移、噪声较大的现象。另外，存在 1 张图像出现标注错误，2 张图像存在标注缺失现象。

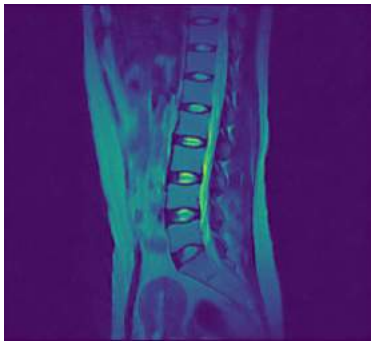


图 5: 正常图像

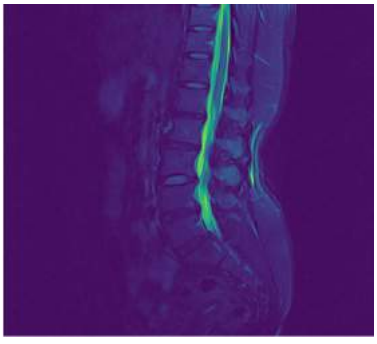


图 6: 不清晰图像

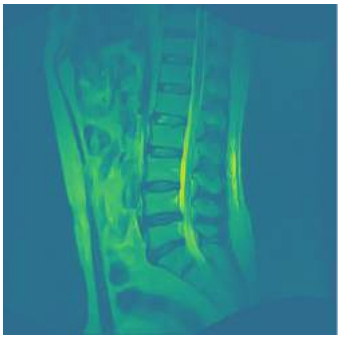


图 7: 噪声图像

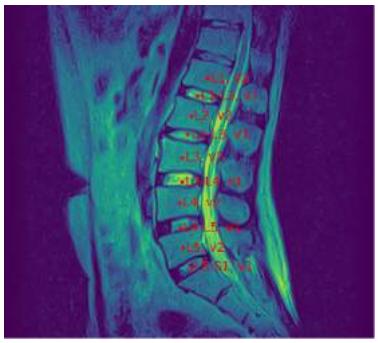


图 8: 标注错误：study82

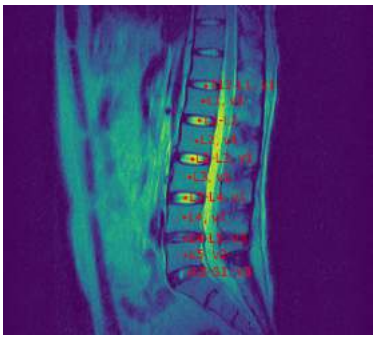


图 9: 标注缺失：study77

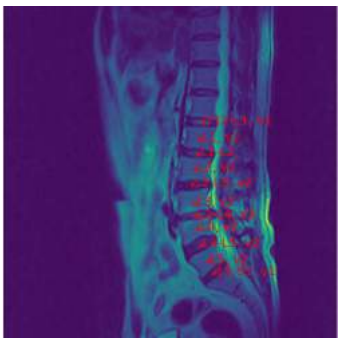


图 10: 标注缺失：study164

共计两百张标签图像，均为脊椎矢状面影像，每张图上定位 11 个点。各点及类别统计如下：

椎骨		椎间盘					椎间盘	
v1	v2	V1	V2	V3	V4	V5only	V5	noV5
191	814	535	300	285	47	36	64	2146

对上述数据，可以得出结论：数据集整体较小，数量不足；图像较为清晰；标签有极少部分出错，且存在较为严重的类别不平衡问题。因此，在进行模型训练之前，应完成数据增强与数据清洗。

2.2 模型

对脊椎的病理检测项目，明确为目标检测与分类任务。

目标检测任务意在检测椎骨与椎间盘的区域，最后回归到点，判断其在脊椎中的位置。数据集的标注数据为像素点，与传统的框选区域不同。考虑到后续任务需要进行分类，选择区域回归框架。

分类任务意在针对每一块椎骨/椎间盘进行病理分析。根据数据，椎骨有 V1 和 V2 两种病理类型，椎间盘则有 V1-V5 五种病理类型。其中，椎间盘的 V5 类病变可以同时和其他类型的病变共存，因此，分类任务可分开为两个模型。一个模型进行椎骨和椎间盘的 V1-V4 的分类；另一个进行目标区域是否发生 V5 病变的分类。

经调研，本次任务已有的取得较好效果的模型中，大多数模型将检测任务与分类任务分为两个阶段进行，由于样本量有限难以直接回归目

标坐标，所有方法均在将样本进行数据增强及一系列预处理后放入检测网络进行训练，再将检测网络最终生成的结果作为分类网络的输入。各个模型的主要区别在于对数据的处理以及检测目标检测网络的改进，分类网络使用的基本为传统的分类模型。使用的部分模型如下：

模型	定位(目标检测)	目标分类
模型1	HigherHRNet	se-resnext50
模型2	maskRCNN	EffcientNet-b5
		Res34-FPN
模型3	YOLOv5+FPN+PAN	
模型4	resnet18	
模型5	Cascade RCNN+FPN+DCN	ResNeSt269

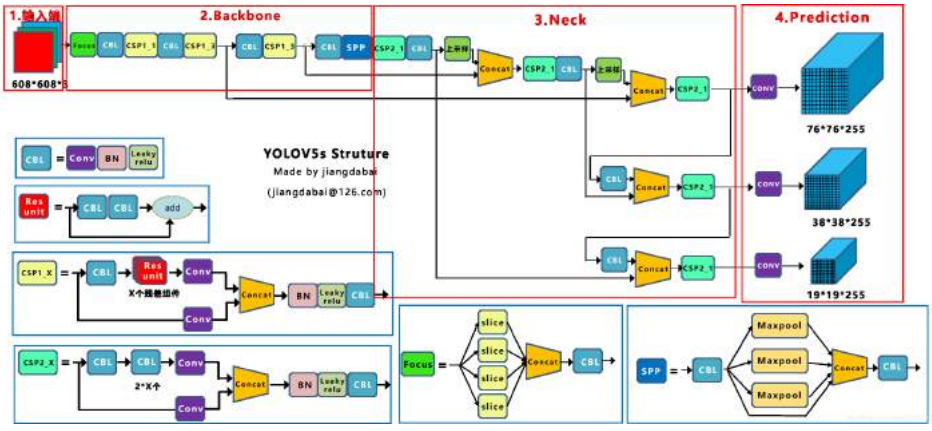
另外，考虑到数据集的不足，尽管模型训练前已进行数据增强，仍然要考虑到数据不足可能带来的欠拟合与过拟合问题。因此，我们将三折交叉验证法用于模型超参数的选择。

3 模型原理

3.1 YOLOv5

Yolov5 的目标检测网络中一共有 4 个版本,分别是 Yolov5s、Yolov5m、Yolov5l、Yolov5x 四个模型。

其中，yolov5s 的网络结构如下图所示，yolov5s 为深度最小，特征图宽度最小的版本，其它三种都是在 yolov5s 的基础上加深加宽。

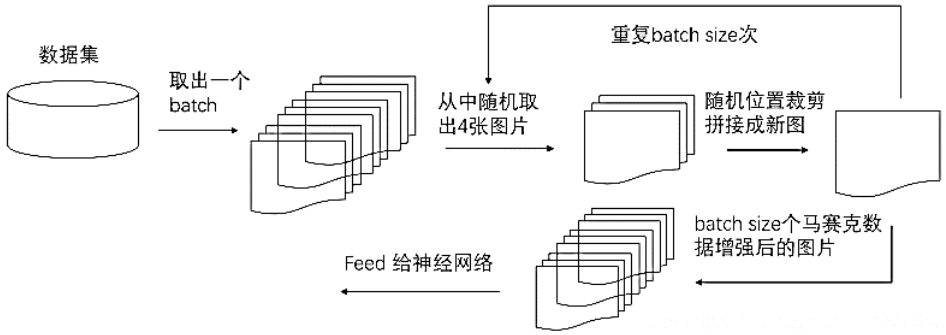


Yolov5 分为输入端、Backbone、Neck、Prediction 四个部分。

3.1.1 输入端

1.Mosaic 数据增强

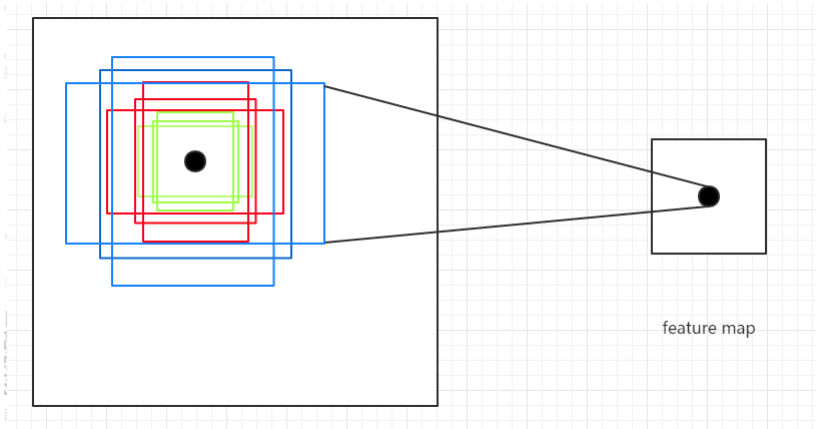
将 4 张图片，通过随机缩放、随机裁减、随机排布的方式进行拼接。该方法能极大地丰富检测数据集，特别是随机缩放增加了很多小目标，使网络的鲁棒性更好，且直接计算 4 张图片的数据，降低了 Mini-batch 的大小，使得在一个 GPU 的条件下就能达到比较好的效果。



2. 自适应锚框计算

预定义边框是一组预设的边框，在训练时，以真实的边框位置相对于预设边框的偏移来构建训练样本。

在网络训练中，网络在初始锚框的基础上输出预测框，进而和真实框 groundtruth 进行比对，计算两者差距，再反向更新，迭代网络参数。



Yolov5 中将计算初始锚框的值的功能嵌入到代码中，每次训练时，自适应的计算不同训练集中的最佳锚框值。Yolov3、Yolov4 则是通过单独的程序运行。

3. 自适应图片缩放（矩形训练）

在常用的目标检测算法中，不同的图片长宽都不相同，因此常用的方式是将原始图片统一缩放到一个标准尺寸，再送入检测网络中。

在项目实际的使用时，由于各张图片的长宽比不一定全部相同，因此缩放填充后，两端的黑边大小不同，但如果填充较多，则存在信息冗余，影响推理速度。而 Yolov5 对代码中 datasets.py 的 letterbox 函数中进行了修改，使原始图像自适应缩放后的添加最少的黑边。

4. 其它数据增强方式

Cutout：随机选择一个固定大小的正方形区域，然后采用全 0 填充。为了避免填充 0 值对训练的影响，应该要对数据进行中心归一化操作。

MixUp：从训练样本中随机抽取两个样本进行简单的随机加权求和，同时样本的标签也对应加权求和，然后预测结果与加权求和之后的标签求损失，在反向求导更新参数，公式定义如下：

$$\widetilde{x} = \lambda x_i + (1-\lambda)\, x_j$$

$$\widetilde{y} = \lambda y_i + (1-\lambda)\, y_j$$

效果示意图如下：

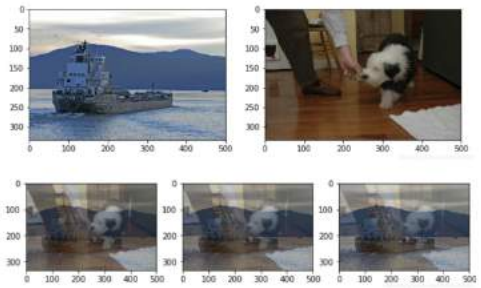


图 11: MixUp

图 12: CutMix

CutMix：随机生成一个裁剪框 Box，裁剪掉 A 图的相应位置，然后用 B 图片相应位置的 ROI 放到 A 图中被裁剪的区域形成新的样本，计算损失时同样采用加权求和的方式进行求解。两张图合并操作定义如下：

$$\widetilde{x} = M \odot x_A + (1-M) \odot x_B$$

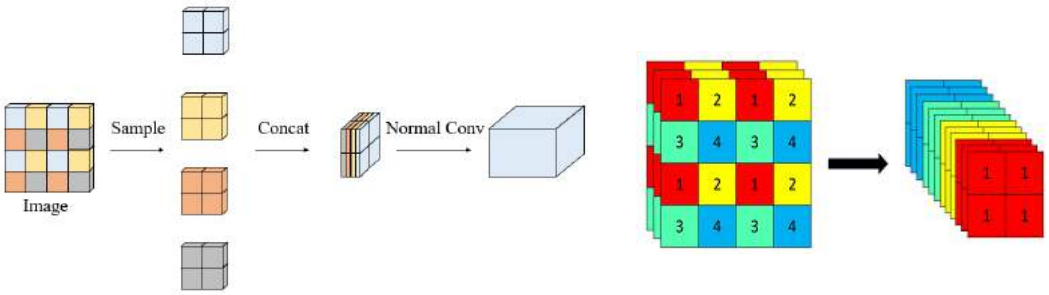
$$\widetilde{y} = \lambda y_A + (1-\lambda)\, y_B$$

除此之外 yolov5 还使用了图像扰动，改变亮度、对比对、饱和度、色调，加噪声，随机缩放，随机裁剪（random crop），翻转，旋转，随机擦除等方式对数据进行增强。

3.1.2 Backbone

1.Focus 结构

具体操作是在一张图片中每隔一个像素拿到一个值，类似于邻近下采样，这样就拿到了四张图片，四张图片互补，长的差不多，但是没有信息丢失，这样一来，将 W、H 信息就集中到了通道空间，输入通道扩充了 4 倍，即拼接起来的图片相对于原先的 RGB 三通道模式变成了 12 个通道，最后将得到的新的图片再经过卷积操作，最终得到了没有信息丢失情况下的二倍下采样特征图。



Focus 的作用是使图片在下采样的过程中，不带来信息丢失的情况下，将 W、H 的信息集中到通道上，再使用卷积对其进行特征提取。虽然增加了计算量，但是为后续的特征提取保留了更完整的图片下采样信息。

各个不同版本的卷积核大小有区别，最小的为 yolov5s，为 32 个。

2.CSP 结构

初衷是减少计算量并且增强梯度的表现。主要思想是：在输入 block 之前，将输入分为两个部分，其中一部分通过 block 进行计算，另一部分直接通过一个 shortcut 进行 concatenate。

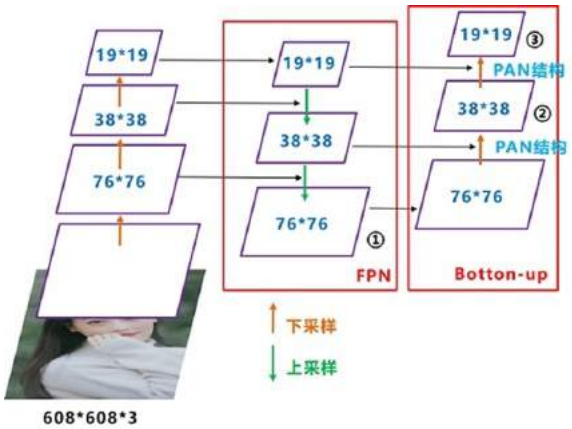
Yolov5 中设计了两种 CSP 结构，以 Yolov5s 网络为例，CSP1_X 结构应用于 Backbone 主干网络，另一种 CSP2_X 结构则应用于 Neck 中。

CSPDarknet53 网络结构图

	Type	Filters	Size	Output
	Convolutional	32	3x3	256x256
	Convolutional	64	3x3/2	128x128
	CrossStagePartial			
1x	Convolutional	32	1x1	
	Convolutional	64	3x3	
	Residual			128x128
	Convolutional	128	3x3/2	64x64
	CrossStagePartial			
2x	Convolutional	64	1x1	
	Convolutional	64	3x3	
	Residual			64x64
	Convolutional	256	3x3/2	32x32
	CrossStagePartial			
8x	Convolutional	128	1x1	
	Convolutional	128	3x3	
	Residual			32x32
	Convolutional	512	3x3/2	16x16
	CrossStagePartial			
8x	Convolutional	256	1x1	
	Convolutional	256	3x3	
	Residual			16x16
	Convolutional	1024	3x3/2	8x8
	CrossStagePartial			
4x	Convolutional	512	1x1	
	Convolutional	512	3x3	
	Residual			8x8
	Avgpool		Global	
	Connected		1000	
	Softmax			

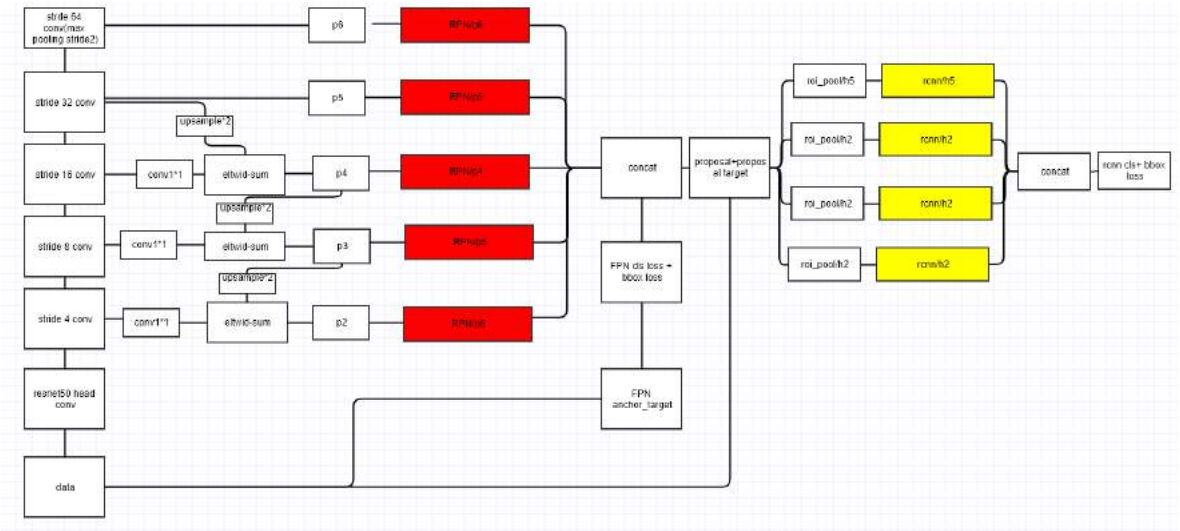
3.1.3 Neck

Yolov5 现在的 Neck 和 Yolov4 中一样，都采用 FPN+PAN 的结构。



1.FPN 结构

FPN 构架了一个可以进行端到端训练的特征金字塔。通过 CNN 网络的层次结构，FPN 可以高效地进行强特征计算。同时，它可以通过结合 bottom-up 与 top-down 方法获得较强的语义特征，提高目标检测和实例分割在多个数据集上面的性能表现。



2.PAN 结构

像素聚合网络 Pixel Aggregation Network 是 PSENet 的改进版，依旧是 segmentation-based 文本检测方法，可以检测任意形状的文本。主要改进了 PSENet 速度慢的缺点。

3.1.4 输出端

1.Bounding box 损失函数

yolov5 采用 GIOU_Loss 作为损失函数。

$$CIOU_{Loss} = 1 - (IOU - \frac{{Distance_2}^2}{{Distance_C}^2} - \frac{v^2}{(1-IOU)+v})$$

$$IOU = \frac{|A \cap B|}{|A \cup B|}$$

$$v = \frac{4}{\pi^2}(\arctan\frac{w^{gt}}{h^{gy}} - \arctan\frac{w^p}{h^p})$$

2.nms 非极大值抑制

nms 操作原理为将当前所有边框以 score 分数做降序排序，选出第一个做结果进入结果队列，将剩下的 box 与其比较 iou 值，抑制大于阀值的 box，将剩下的 box 作为新鲜数据开始第二轮，直到最终结束。

Yolov5 中采用加权 nms 的方式。

$$M = \frac{\sum_i \omega_i B_i}{\sum_i \omega_i}, \quad B_i \epsilon \{B | IOU\,(M, B) \geq thresh\} \cup M$$

其中，加权的权重为 $\omega_i = s_i IOU(M, B_i)$，表示得分与 IOU 的乘积。

3.1.5 YOLOv5 四种网络的比较

	Yolov5s	Yolov5m	Yolov5l	Yolov5x
第一个CSP1	CSP1_1	CSP1_2	CSP1_3	CSP1_4
第二个CSP1	CSP1_3	CSP1_6	CSP1_9	CSP1_12
第三个CSP1	CSP1_3	CSP1_6	CSP1_9	CSP1_12
第一个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第二个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第三个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第四个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4
第五个CSP2	CSP2_1	CSP2_2	CSP2_3	CSP2_4

	Yolov5s	Yolov5m	Yolov5l	Yolov5x
(1)卷积核数量	32个	48个	64个	80个
(2)卷积核数量	64个	96个	128个	160个
(3)卷积核数量	128个	192个	256个	320个
(4)卷积核数量	256个	384个	512个	640个
(5)卷积核数量	512个	768个	1024个	1280个

以 yolov5s 为例，第一个 CSP1 中，使用了 1 个残差组件，因此是 CSP1_1。而在 Yolov5m 中，则增加了网络的深度，在第一个 CSP1 中，使用了 2 个残差组件，因此是 CSP1_2。其它同理。

3.2 ResNet50

随着网络深度的增加，模型精度并不总是提升，并且这个问题显然不是由过拟合造成的，因为网络加深后不仅测试误差变高了，它的训练误差竟然也变高了。ResNet 的作者提出，这可能是因为更深的网络会伴随梯度消失/爆炸问题，从而阻碍网络的收敛。作者将这种加深网络深度但网络性能却下降的现象称为退化问题（degradation problem）。ResNet 就是为了解决这种退化问题而诞生的。

3.2.1 ResNet 网络结构

ResNet 的论文 *Deep Residual Learning for Image Recognition* 中介绍了各种 ResNet 的网络结构：

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

3.2.2 ResNet 核心理论

1.Residual Learning

存在这样一个假设：如果多个非线性层可以渐进地逼近复杂的函数，那么其相当于多个非线性层可以渐进地逼近复杂函数的残差函数：$H(x)-x$（假定输入和输出具有相同的维度）。所以，与其期望堆叠层去逼近 $H(x)$，不如明确的让这些堆叠层去逼近一个残差函数 $F(x) = H(x)-x$，则最初的函数变为 $F(x)+x$。尽管两种形式都可以渐进地逼近预期的函数，但是学习的容易程度也许会不一样。

重新公式化表示（即 $H(x) = F(x) + x$）受关于衰退问题（朴素神经网络越深，训练错误越高）的违反自觉现象所激励。衰退问题表明，求解程序在通过多个非线性网络层去逼近恒等映射时也许会存在困难。由于残差学习的重新公式化表示，如果恒等映射是最优的，求解程序也许只要简单地将多个非线性网络层的权重向 0 逼近，以此逼近恒等映射。

在现实情况下，恒等映射是最优的是不太可能的，但是我们的重新公式化表示也许有助于预处理网络衰退的问题。如果相较于接近一个 0 映射，最优化的函数更接近于一个恒等映射；那么对于求解程序来说，相较于像一个新的网络架构去学习目标函数，去发现关于恒等映射的扰动会更容易。

2.Identity Mapping by Shortcuts

考虑一个结构性模块定义为：$y = F\left(x, \{\omega_i\}\right) + x$

其中，x 和 y 是网络层的输入和输出向量。函数 $F\left(x, \{\omega_i\}\right)$ 代表需要学习的残差映射。以下图两层网络为例，$F = \omega_2\sigma(\omega_1 x)$，其中 σ 表示 ReLU 函数（忽略了偏置）。函数操作 $F + x$ 是由一个捷径连接和一个 element-wise addition 构成。在加法之后采用了第二个非线性函数（ReLU 函数，即 $\sigma(F + x)$）。

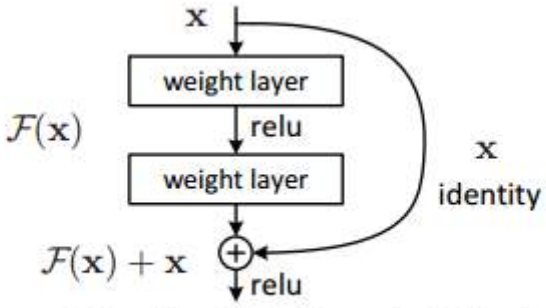


Figure 2. Residual learning: a building block.

在上述公式中，x 和 F 的维度必须相同，如果不同，则可以通过一个残差连接执行一个权重为 ω_s 的线性映射，用以匹配输入和输出维度。

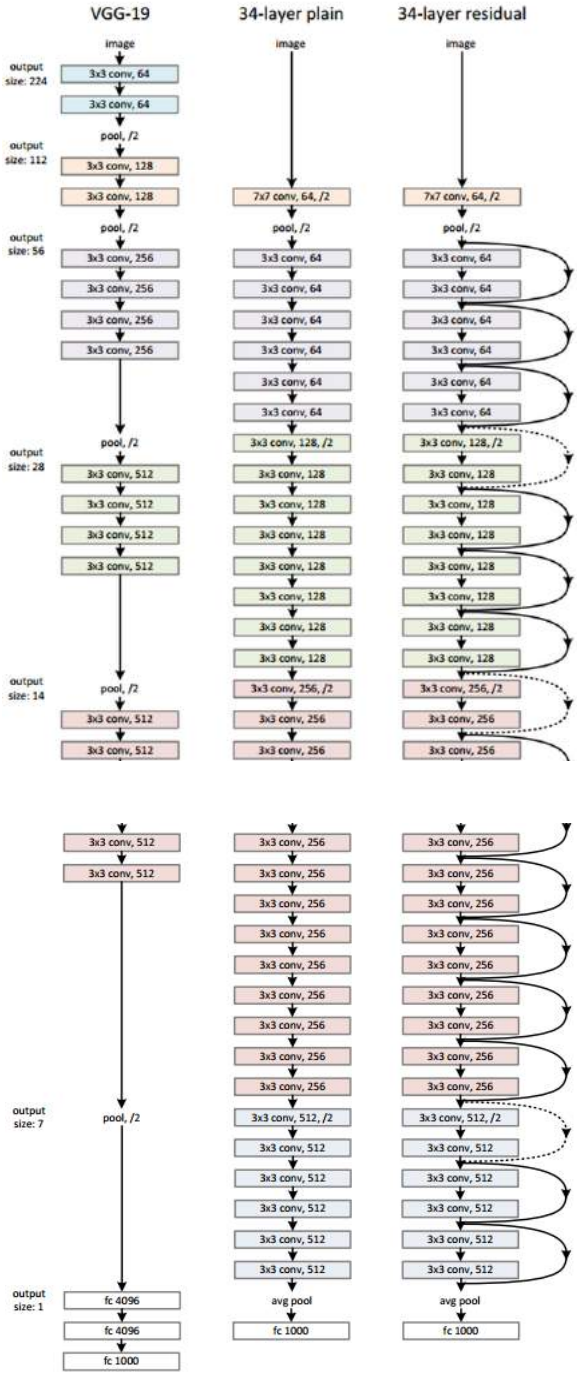
$$y = F\left(x, \{\ omega a_i\}\right) + \omega_s x$$

恒等映射足以解决网络衰退的问题，并且恒等映射是经济性的。

残差学习同样适用于卷积层。函数 $y = F(x, \{\omega_i\})$ 可以代表多个卷积层。

3.Network Architectures

基于朴素神经网络，能够如下图所示插入捷径连接，将其转变为与其对应的残差版本。（34-layer 的采用的为 Basicblock 结构）



4.Implemention

重新度量图片的大小：为了增加尺度，随机地在 [256, 480] 范围内进行取样，并用其较小边作为图片的大小。从一个图片中随机的取样一个大小为 $224 * 224$ 的修剪块，或者对该修剪块进行水平翻转，并且每个像素减去其平均像素值。在进行卷积之后及 ReLU 之前，我们采用了 batch normalization。

3.2.3 ResNet 的网络设计规律

1. 整个 ResNet 不使用 dropout，全部使用 BN；

2. 受 VGG 的启发，卷积层主要是 3×3 卷积；

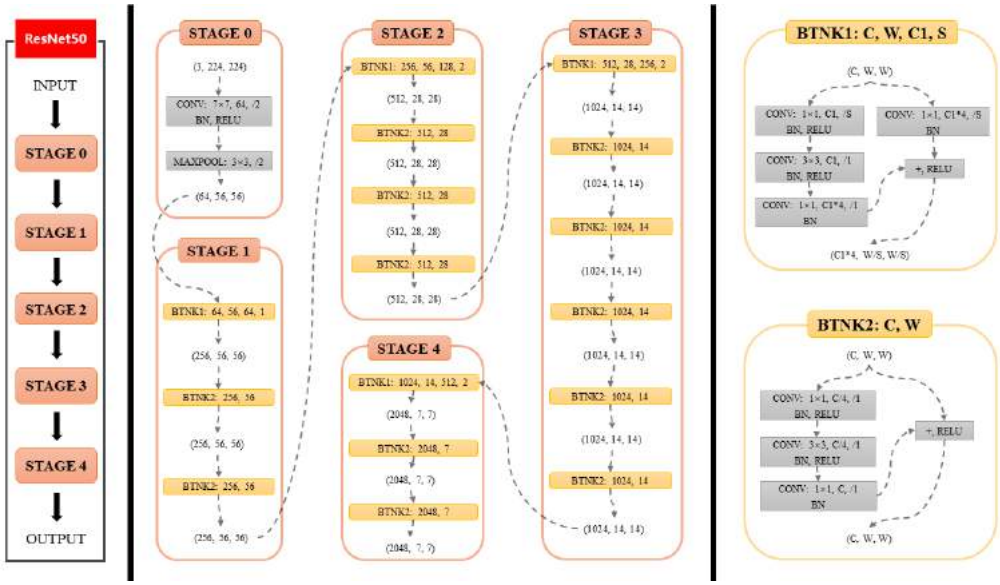
3. 对于相同的输出特征图大小的层，即同一 stage，具有相同数量的 3x3 滤波器；

4. 如果特征地图大小减半，滤波器的数量加倍以保持每层的时间复杂度；

5. 每个 stage 通过步长为 2 的卷积层执行下采样，而却这个下采样只会在每一个 stage 的第一个卷积完成，有且仅有一次；

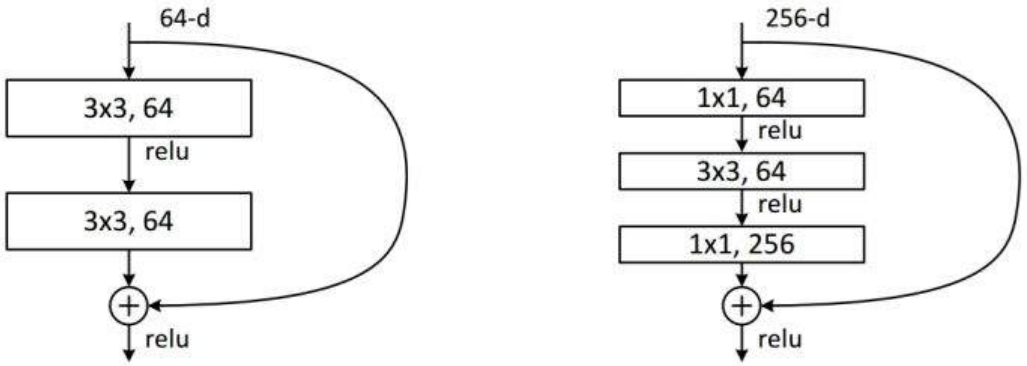
6. 网络以平均池化层和 softmax 的 1000 路全连接层结束，实际上工程上一般用自适应全局平均池化 (Adaptive Global Average Pooling)。

3.2.4 ResNet50 的 backbone 部分结构



其中，CONV 表示卷积层，$n \times n$ 表示卷积核大小，下一位数表示卷积核数量，n 表示步长，BN 表示 Batch Normalization，即 BN 层，RELU 为激活函数。BNTK 表示 Bottleneck。

下图为 Basicblock 和 Bottleneck 结构，左为 Basicblock，自 ResNet50 起，就采用 Bottleneck 结构。



其中主要区别为引入 1×1 的卷积，作用为：

1. 对通道数进行升维和降维（跨通道信息整合），实现了多个特征图的线性组合，同时保持了原有的特征图大小；
2. 相比于其他尺寸的卷积核，可以极大地降低运算复杂度；
3. 如果使用两个 3×3 卷积堆叠，只有一个 relu，但使用 1×1 卷积就会有两个 relu，引入了更多的非线性映射。

4 实验与分析

4.1 数据清洗与数据增强

4.1.1 数据集划分

对题给标注格式及图像集，提取标注数据及对应的原图像。

在给定的两百份图像集中，对每份图像集，我们挑选其中的标注图像及其对应的未标注图像作为数据集。

为方便后续操作，对每份数据进行重命名，命名格式为：xxx-xxx。其中，每个”x” 代表 0-9 中的一位数字。“-” 前 “xxx” 与图像集名称 study

后的数字相对应；“-”后“xxx”该代表图像集中的某张图像，“000”表示标注图像对应的未标注图像。如“study0”中的标注图像对应的未标注图像重命名为“000-000”。

直接根据题给数据集的编号，直接划分前 150 份图像为训练集，后 51 份图像为测试集。

根据数据集分析结果，对标注错误的图像 study82，将其从数据集中删去；对标注缺失的 study77 和 study164，将未标注数据设置为 NULL。

4.1.2 重定义数据集

1. 数据集格式

观察题给标注，从中提取有效字段，并将其重新编写成标注数据集格式，得到完整数据集，便于后续操作。

题给标注格式如下图：

```
{
    "studyUid": "1.2.840.473.8013.20181026.1091511.864.22434.92",  # dicom study UID
    "data": [
        {
            "instanceUid": "1.3.46.670589.11.32898.5.0.16316.2018102610554807009", # dicom instance UID
            "seriesUid": "1.3.46.670589.11.32898.5.0.16316.2018102610554554000", # dicom series UID
            "annotation": [
                {
                    "annotator": 13              #  可选
                    "point": [                   #关键点标注
                        {
                            "coord": [252, 435],        #点的像素坐标
                            "tag": {
                                "disc": "v2,"                 # 椎间盘类型膨出
                                "identification":"L1-L2"      # 椎间盘定位L1-L2间椎间盘
                            },
                            "zIndex": 0,                #第几个slice
                        },
                        {
                            "coord": [211,76],
                            "tag": {
                                    "identification": "L1", # 腰1椎体
                                    "vertebra": "v2".    # 退化椎体
                            },
                            "zIndex": 5
                        }
                    ],
                }
            ]
        }
    ]
}
```

易得，与标注数据相关的有效字段为：‘coord’，‘disc’，‘vertebra’。将提取出的数据有效字段重新编写成标准数据集格式（此处使用 COCO

格式）。针对椎体的病理情况，进行两种标签分类，如下表所示。

表1分类编码1对应表

	标注	编码1
椎体	v1	0
	v2	1
椎间盘	v1	2
	v2	3
	v3	4
	v4	5
	v5 only	6

表2分类编码2对应表

	标注	编码2
椎体	v1	0
	v2	0
椎间盘	v1	0
	v2	0
	v3	0
	v4	0
	v5	1

将中心点从上到下排序，按照我们的两种分类重新打上标签。

2. 图像分割

考虑到后续需对锥体进行病理分析，我们对椎体进行分割提取，便于逐个进行病理判断。

我们利用相邻的中心点进行锥体区域的分割。对于中间的点，考虑上下相邻两个点的纵坐标作差，取较大值乘二作为框的高，然后高乘以 1.4 作为宽；对于首位两个点，类似中间的点，直接取相邻点的纵坐标作差得到框的高，再乘 1.4 得到宽。

根据规则生成框后，得到 COCO 格式的完整数据集。与此同时，提取框的信息，进行图片分割，得到用于病理分类的图片。

综上，新的数据集标注格式如下：


```
{
    "id": "041-000",
    "point": [
        {
            "identification": 1,
            "class": "(1,0)",
            "coord": [169, 252],
            "bbox": [143.8, 234.0, 194.2, 270.0]
        },
        {
            "identification": 3,
            "class": "(1,0)",
            "coord": [169, 224],
            "bbox": [148.0, 209.0, 190.0, 239.0]
        }
    ]
}
```


接着，将 COCO 数据集转化为 txt、xml 格式的标准数据集，便于

后续实验框架的尝试。

4.1.3 数据增强

针对部分较不清晰的图像，考虑到现实图像的采样效果可能较差，为提高模型的鲁棒性，不予处理。

针对数据集规模较小的问题，对于上述分类用的训练集图片，进行中心点抖动。也即，给定一张图片，以原有标注点为中心点，以宽的 $\frac{1}{10}$ 像素为半径，上下左右分别抖动，生成四张新的图片。

针对数据集类别不平衡严重的问题，对训练集进行随机高斯噪声和重采样操作，以平衡类间数量。随机高斯噪声，是通过分别计算数量最大类与其他类的差，得到补充数目，然后迭代补充数目次，每次从较少数目类中随机选择一张已有的图片，加入噪声，得到新的图片。

4.2 目标检测与分类

首先，我们尝试端到端的模型直接完成椎体的定位和病理检测。尝试的两个模型准确率如下：

Faster RCNN：43.8%　　YOLOv3：51.2%

模型运行过程中，我们发现，椎体检测的准确率较高，保持在 80% 以上，能超过 90%。而病理分类的准确率较低，与检测的准确率平均后，最终结果的准确率较低。

对分类过程进行探究，发现，对于椎骨的分类，仅为二分类，比较简单；对于椎间盘的分类，一方面，病理类型较多，另一方面，v5 类型会与其他病变类型重叠，其特征难以提取，分类准确率较低。

因此，我们考虑将目标检测与分类任务分开进行训练。

4.3 目标检测

本任务中，我们尝试了区域回归和点回归两种方案。

4.3.1 区域回归

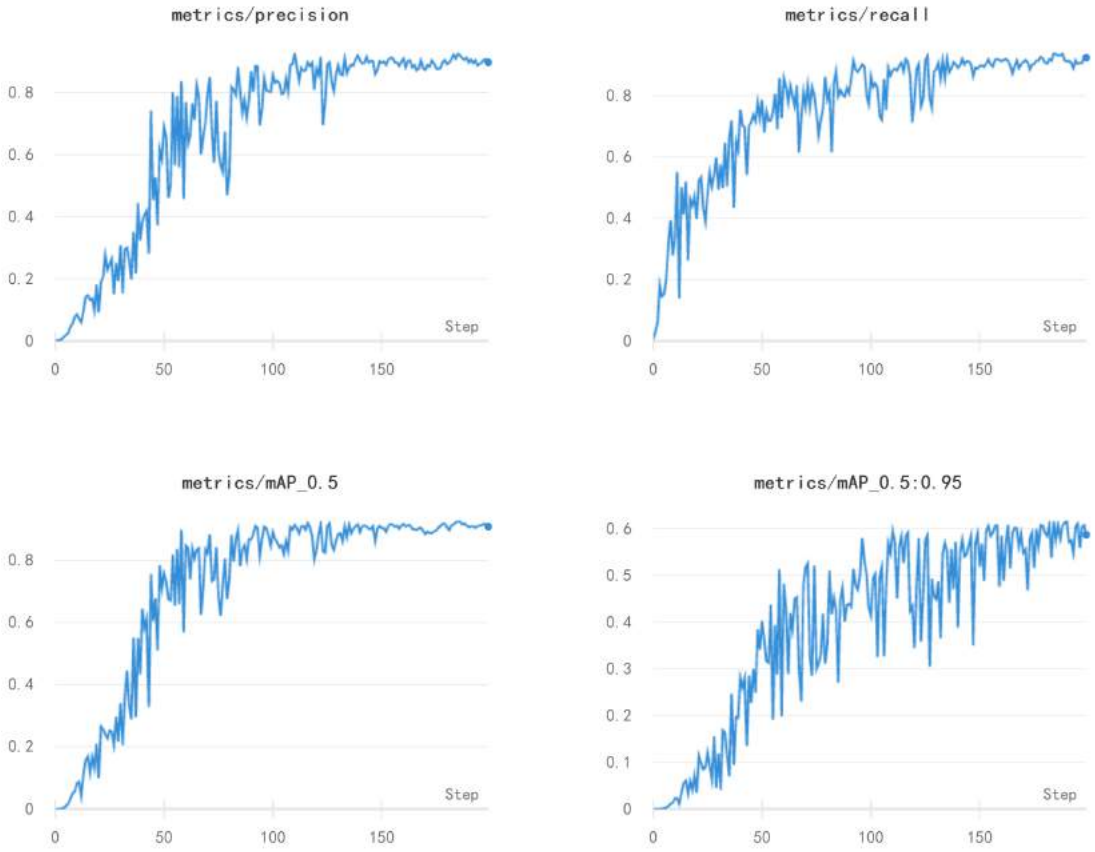
尝试使用 YOLOv5 框架进行目标的区域回归。由于数据规模较小，此处，我们使用三折交叉验证模型进行模型训练及模型超参数的选择。

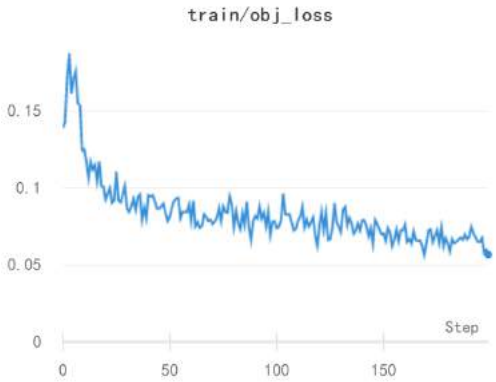
本次实验中，三折验证结果如下：

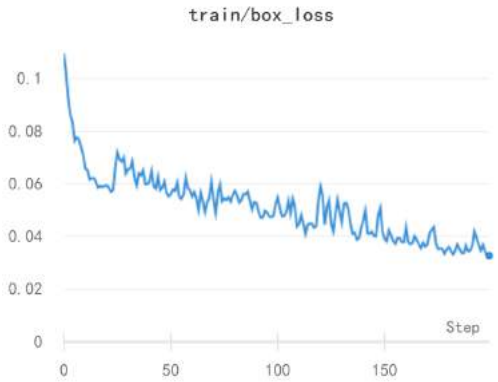
目标检测	三折准确率			均值
	一	二	三	
原始模型	0.843	0.838	0.867	0.849
Label smooth=0.1	0.843	0.889	0.849	0.86
Mixup=0.2	**0.87**	**0.9**	**0.865**	**0.878**

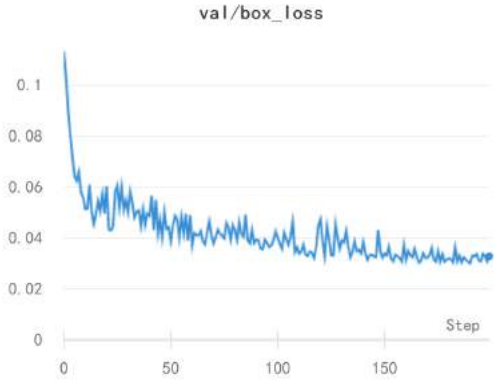
根据上述交叉验证结果可知，label smooth 方法在第三折上相对原模型效果显著降低，而 mixup 整体上有较大提升，因此，选取 mixup=0.2 作为训练最终模型的额外参数。

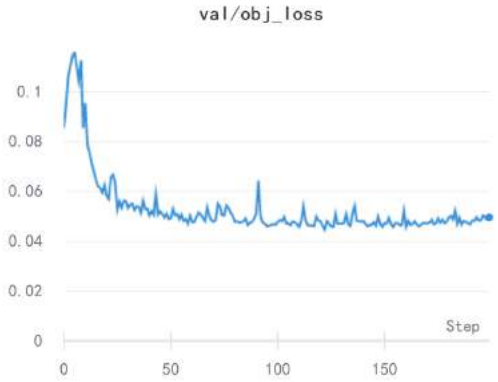
对 mixup=0.2 的检测模型，选取其中一折，观察其训练过程的参数变化如下：

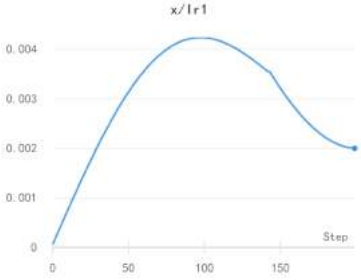


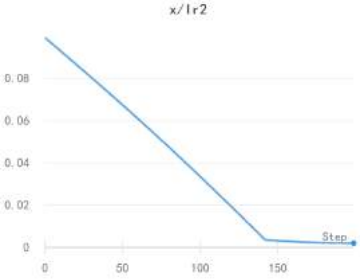


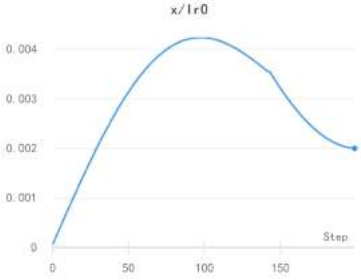












其中，可以看出，由于 YOLOv5 的特性，训练过程中，准确率等值局部波动较大，整体趋于收敛。

4.3.2 点回归

尝试用 U-net 直接对椎体做关键点检测，进行了如下操作：

1. 直接用一个全连接层输出一个与图片大小相同 one-hot 矩阵，效果较差。

2. 在 1 的基础上加上 heatmap，即以目标点为中心加上一个高斯分布，效果显著提高，在训练集及测试集上准确率如下：

trainset accuracy：0.9024　　test accuracy：0.1836

易知，模型在训练集上发生了过拟合。但因测试集上表现与 YOLOv5 相差较远，因此，没有进一步对过拟合进行处理，直接选择区域回归方案。

4.4 目标分类

按照分析，我们将目标分类分为二分类和七分类两个分类任务，并同时进行模型的训练。

我们使用 ResNet50 分别进行两个分类任务的模型训练，并尝试了不同的超参数。另外，由于训练集的样本平衡进行了不同尝试，因此，针对数据增强的不同处理，也进行了实验。

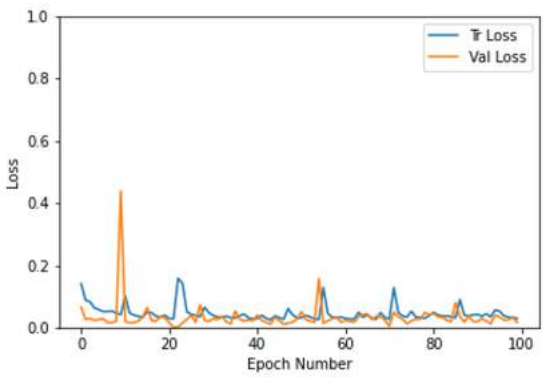
4.4.1 ResNet50 的二分类任务

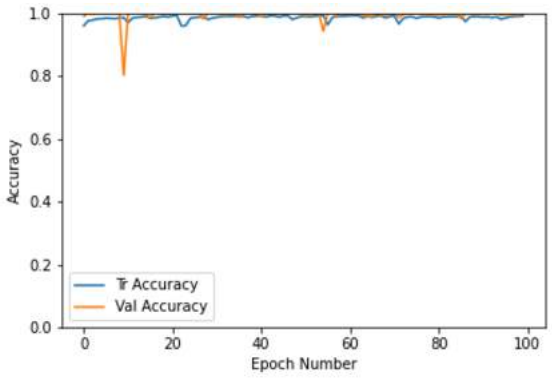
二分类结果如下，

二分类	Epoch	Training Data		Validation Data		Time	Best Accuracy for Validation
		Loss	Accuracy	Loss	Accuracy		
只做Resize	100	0.065	97.94%	0.0687	97.82%	7.8935	0.9782 at epoch 100
加上Normalize	100	0.0596	98.25%	0.0941	97.27%	8.8954	0.9727 at epoch 100
预处理加上加高斯噪声（加Normalize和Rotation）	**100**	**0.0298**	**99.07%**	**0.0166**	**99.82%**	**8.8665**	**1.000 at epoch 02**

容易得出，经过随机高斯噪声操作的数据增强，对实验效果的提升有较好作用。

训练过程的部分参数变化如下：





4.4.2 ResNet50 的七分类任务

七分类结果如下，

七分类	Epoch	Training Data		Validation Data		Time/s	Best Accuracy for Validation
		Loss	Accuracy	Loss	Accuracy		
只做Resize	100	0.1623	93.43%	1.986	57.92%	10.1214	0.5956 at epoch 79
修改Batch Size为1024	100	0.1914	92.69%	1.7707	57.01%	10.1310	0.5883 at epoch 47
加上Normalize	100	0.1402	94.15%	1.9421	58.47%	11.3012	0.6066 at epoch 30
加上旋转Rotation	100	0.8694	65.35%	1.1250	58.11%	12.8468	0.6193 at epoch 85
加上色彩调整ColorJitter	100	0.9778	61.11%	1.0964	58.29%	24.3365	0.6011 at epoch 77
预处理加高斯噪声（不加Normalize）	100	0.3230	89.07%	1.4794	64.12%	10.3528	0.6576 at epoch 99
预处理加上加高斯噪声（加Normalize和Rotation）	**100**	**1.1167**	**56.63%**	**1.0373**	**62.48%**	**12.7521**	**0.6539 at epoch 37**

对上述实验，我们可以发现：

• 修改 Batch Size = 1024 后，效果较 Batch Size = 512 时略差。

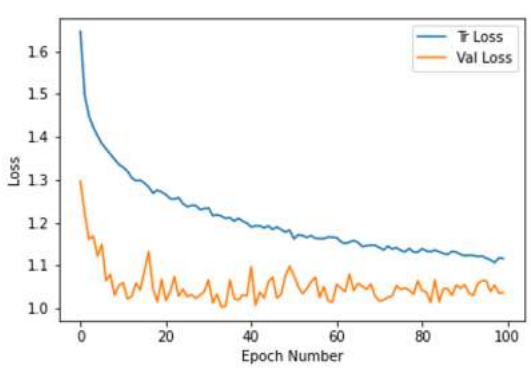
• 使用 Normalize 标准化后，效果有所提升。

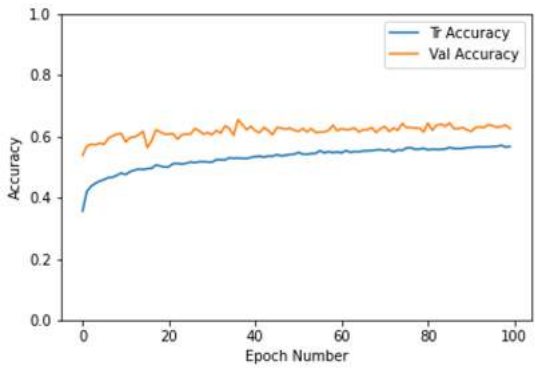
• 加上旋转 Rotation 后，发现，训练集拟合的难度加大，但模型在测试集上表现更好。

• 加上图片色彩调整 ColorJitter 后，训练难度增加，训练时长加倍，但模型在测试集上表现没有明显提升。

• 采用随机高斯噪声增强后的训练集，并加上 Normalize 标准化及旋转 Rotation 后，测试集上的 loss 和 accuracy 都要高于训练集，且最终准确率高于其他模型。因此，采用该种方案。

训练过程的部分参数变化如下：





4.4.3 其他尝试

实际过程中，尝试 VGG 以及其他 Resnet 模型：

1.VGG 效果较 Resnet 差

2.ResNet18、ResNet34 分类效果不如 ResNet50

3.ResNet101 训练时间长，且效果提升不明显。

但，因未做好结果的保存，未能在报告上呈现具体数据。

4.5 实验结果

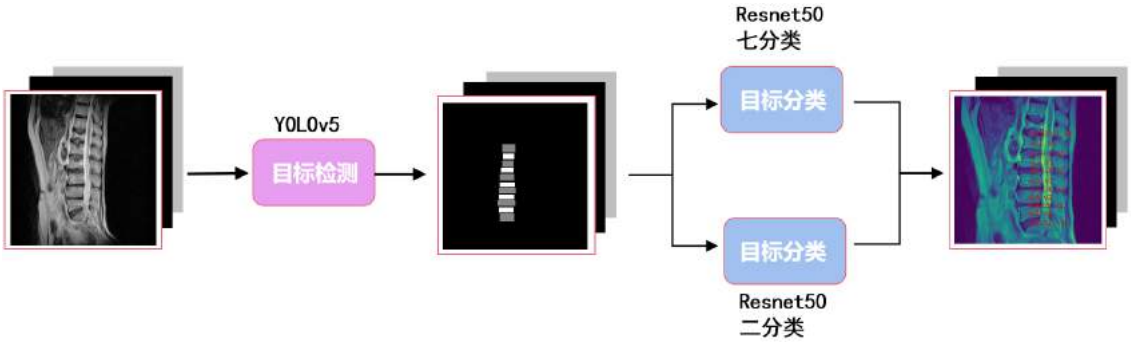
结合 YOLOv5 跑出来的结果框进行分类，并将两个任务结果结合，计算指标。

其中，将预测的坐标和标注坐标距离小于 8pixel 的点、且分类正确将被计数为真正例（TP）；label_counts 为测试数据集总点数（即 50*11-1=549，减去 1 是因为有一个点数据为空）；pred_counts 为 yoloV5 预测出来的总点数（也等于分类总数，这里为 912），设定 YOLO 框架中重新筛选框的置信度阈值选的是 0.25，iou 阈值是 0.8，YOLOv5 会对每一个标注点输出 0-n 个预测点。则有，

```
print("recall:", TP / label_counts)
print("precision:", TP / pred_counts)
print("AP:", mean([mean([ap_list[j]['TP'] for j in range(i)]) for i in range(1, len(ap_list))]))
```

```
recall: 0.6666666666666666
precision: 0.4021978021978022
AP: 0.6574225641879335
```

综上，本项目的模型框架如图所示：



超参数设置如下：

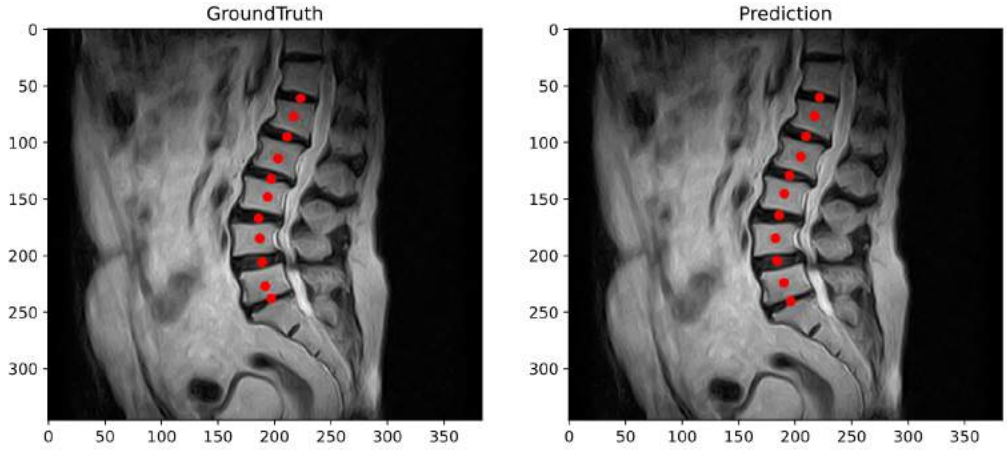
```
# YOLO 超参数设置
lr0: 0.01  # initial learning rate (SGD=1E-2, Adam=1E-3)
lrf: 0.2  # final OneCycleLR learning rate (lr0 * lrf)
momentum: 0.937 # SGD momentum/Adam beta1
weight_decay: 0.0005 # optimizer weight decay 5e-4
warmup_epochs: 3.0 # warmup epochs (fractions ok)
warmup_momentum: 0.8 # warmup initial momentum
warmup_bias_lr: 0.1 # warmup initial bias lr
box: 0.05  # box loss gain
cls: 0.5  # cls loss gain
cls_pw: 1.0  # cls BCELoss positive_weight
obj: 1.0  # obj loss gain (scale with pixels)
obj_pw: 1.0 # obj BCELoss positive_weight
iou_t: 0.20  # IoU training threshold
anchor_t: 4.0  # anchor-multiple threshold
fl_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)
hsv_h: 0.015 # image HSV-Hue augmentation (fraction)
hsv_s: 0.7  # image HSV-Saturation augmentation (fraction)
hsv_v: 0.4  # image HSV-Value augmentation (fraction)
degrees: 0.0  # image rotation (+/- deg)
translate: 0.1  # image translation (+/- fraction)
scale: 0.5  # image scale (+/- gain)
shear: 0.0  # image shear (+/- deg)
perspective: 0.0  # image perspective (+/- fraction), range 0-0.001
flipud: 0.0  # image flip up-down (probability)
fliplr: 0.5  # image flip left-right (probability)
mosaic: 1.0  # image mosaic (probability)
mixup: 0.2  # image mixup (probability)
```

```
# ResNet50 超参数设置
NUM_EPOCH = 100
batch_size = 512
learning_rate = 0.001
criterion  = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = torch.optim.lr_scheduler.StepLR(
        optimizer, step_size=5, gamma=0.9) # 学习率下降
```

4.6 案例可视化

模型设置置信度阈值为 0.25，nms 阈值为 0.8，以下仅选取置信度最高的结果进行展示。

原图以及中心点预测，（编号 155-000）



类别	真实坐标	预测坐标	置信度
T12-L1	(223,61)	(221.5,60.0)	0.92
L1	(217,77)	(217.4,76.7)	0.87
L1-L2	(211,95)	(209.7,94.5)	0.88
L2	(203,114)	(204.9,112.5)	0.89
L2-L3	(197,132)	(195.1,129.4)	0.89
L3	(194,148)	(190.5,145.3)	0.89
L3-L4	(186,167)	(185.7,164.4)	0.89
L4	(187,185)	(182.6,184.7)	0.87
L4-L5	(189,206)	(184.0,204.6)	0.88
L5	(192,227)	(190.2,224.0)	0.91
L5-S1	(197,238)	(196.3,240.5)	0.87

所有预测点与真实值距离都在 8 像素内，点回归准确率：100% 。

预测的中心点、生成分割框及结果比对如下，

图 13: 预测中心点及框

预测标签	真实标签
突出	正常
退行性改变	退行性改变
突出，椎体内疝出	膨出
退行性改变	退行性改变
膨出	膨出
退行性改变	退行性改变
突出	膨出
退行性改变	退行性改变
膨出	膨出
退行性改变	退行性改变
膨出	膨出

分类准确率：72.7% 。

5 总结

本项目中，我们通过 YOLO5 和 ResNet50 训练了椎体的定位检测和病理分析模型，并且在这个过程中逐步提升了数据处理（如制定合理的数据格式并整合成通用 json 文件）、数据增广（如进行中心点抖动以及使用 torchvision.transforms 内的方法）和模型训练（如超参数的选择以及 gpu 使用的技巧）等技能。

5.1 不足

1. 未能充分利用数据集。

题给数据集中，每个病人的图像集除了给定的标注图像外，还有相当一部分清晰的未标注图像没有使用，实际可用数据集应比实验中实验所用数据集规模要大。

针对这种情况，考虑是否可以通过不同图像的混合增强，或是选择与标注图像视角相近的图像进行处理，以扩大数据集。

2. 图像增强的效果比对。

数据集中存在部分较不清晰的图像，可以考虑对该部分图像进行图像增强，并进行模型效果的比对。

5.2 分工

数据预处理：谭源正、郑均、郑晔、马梁

模型训练：郑均、林良涛、张礼霆、谭源正、蔡炼楷、马梁

报告撰写：郑晔、蔡炼楷、马梁、张礼霆、林良涛、郑均