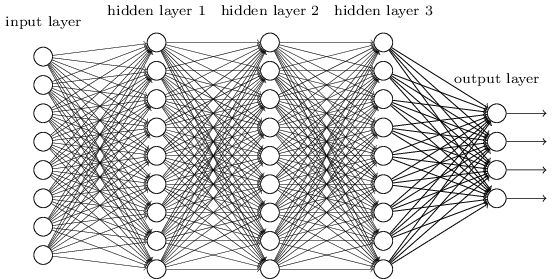
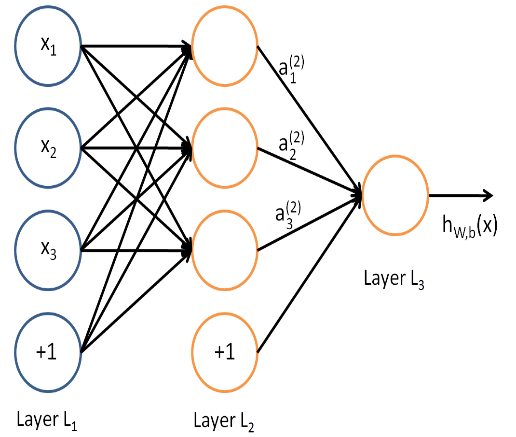
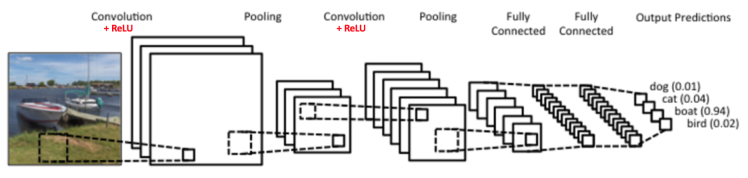
**一.神经网络结构：**

全连接网络的内部，层与层之间是全连接的，也就是说，第i层的任意一个神经元一定与第i+1层的任意一个神经元相连。全连接网络不具备对输入数据的空间结构处理能力。它将输入数据展平为一维向量，并丢失了原始数据的空间信息。



图b-1 全连接网络结构

而CNN的隐藏层由卷积层、池化层和全连接层组成。卷积层用于提取输入数据中的局部特征，池化层用于降低特征图的尺寸并保留主要特征，全连接层用于将提取的特征映射到输出类别上。CNN的输入通常是张量，例如RGB图像，能够处理具有空间结构的数据。



图b-2 CNN网络结构

**2.参数和数据效率：**

使用卷积核在输入上滑动，可以共享权重来检测相同的特征模式，CNN通常具有较少的参数数量，尤其适用于处理大规模数据，例如图像。这也降低了计算复杂度，使得CNN更容易训练和部署。而全连接网络的参数数量较大

，尤其是在输入维度较高的情况下。CNN以从数据中提取出更少但更有代表性的特征，因此对于大规模数据集的训练效果较好。全连接网络通常需要更多的数据样本来获得好的泛化性能。否则，容易出现过拟合现象。并且，全连接网络由于每个神经元都有独立的权重，很难评估网络中每个神经元对结果的贡献。

3.**算法上：**

由于CNN多了卷积层和池化层，这两层的前向传播和后向传播的数学推导和具体实现上也不同。

**二、推导神经网络前向后向传播算法的优化迭代公式**

**1.DNN前向传播**

从小的局部模型来看，即一个线性关系 和一个激活函数。

将图c-1一般化, 可以总结出DNN前向传播的一般形式：,假设第 层共有m个神经元，则对于第l层的第j个神经元的输出,有：

用矩阵表示为：

 （1）

**2.DNN后向传播**

选择一个损失函数，来度量训练样本计算出的输出和真实的训练样本输出之间的损失。以最常见的均方差损失函数，来推导DNN反向传播。

为输出层，y为输出层理论值。DNN反向传播是基于链式求导法则的，知道第层的梯度，想得到第-1层的梯度，就要定义中间变量 :

 （2）

先求出输出层的：

 （3）

对于均方差损失函数，，所以最终。

接下来需要通过数学归纳法，求出与的传递关系：

 （4）

 （5）

写成矩阵形式为：  （6）

所以，第层的梯度为：

 ， （7）

写成矩阵形式为：

 ，  （8）

就可以采用梯度下降的方法更新：

 ，  （9）

**3.CNN前向传播**

与DNN相比，从输入层前向传播到卷积层和隐藏层前向传播到卷积层，区别在于使用卷积，并且输入是张量,卷积层激活函数是ReLU;隐藏层前向传播到池化层只需对输入的矩阵进行缩小，池化标准一般是MAX或者Average；隐藏层前向传播到全连接层和DNN相同，但最后一层的输出层激活函数是softmax。

是卷积层： \*表示卷积运算

是池化层： ，pool表示按池化大小和标准将输入张量缩小

是全连接层：

对于输出层第层:

**4.CNN反向传播：**

与DNN相同，主要求出中间变量和的传递关系，来推导出上一层的梯度。

全连接层反向传播与DNN相同，主要探讨卷积层和池化层。

**池化层的推导上一隐藏层的：**

反向传播时，我们首先会把的所有子矩阵矩阵大小还原成池化之前的大小，然后如果是MAX，则把的所有子矩阵的各个池化局域的值放在之前做前向传播算法得到最大值的位置。如果是Average，则把的所有子矩阵的各个池化局域的值取平均后放在还原后的子矩阵位置。这个过程一般叫做upsample（上采样）。

 （10）

**卷积层的推导上一隐藏层的：**

 （11）

与DNN不同就是使用了卷积预算\*并且卷积核被旋转了180度（先上下翻转一次再左右翻转一次）。

求出每层的梯度后可以求改成的的梯度，池化层没有就不用求，仅考虑卷积层。

由 ，得出  （12）

对于b,则有些特殊，因为是高维张量，而b只是一个向量，不能像DNN那样直接和相等。通常的做法是将的各个子矩阵的项分别求和，得到一个误差向量，即为b

的梯度：  （13）

所以CNN反向传播使用梯度下降更新表示为

1）若为全连接层（则与DNN相同）： ，

2）若为卷积层： ，

**三、标注与训练**

**3.1 环境搭建**

**安装Anaconda**

Ubuntu（Linux）下安装方法（在适当的目录下命令行操作：

1. wget https://repo.anaconda.com/archive/Anaconda3-2024.06-1-Linux-x86\_64.sh

2. sudo chmod a+x Anaconda3-2024.06-1-Linux-x86\_64.sh

3. sh（or bash） Anaconda3-2024.06-1-Linux-x86\_64.sh

How to use Anaconda gently

1.conda create -n 环境名

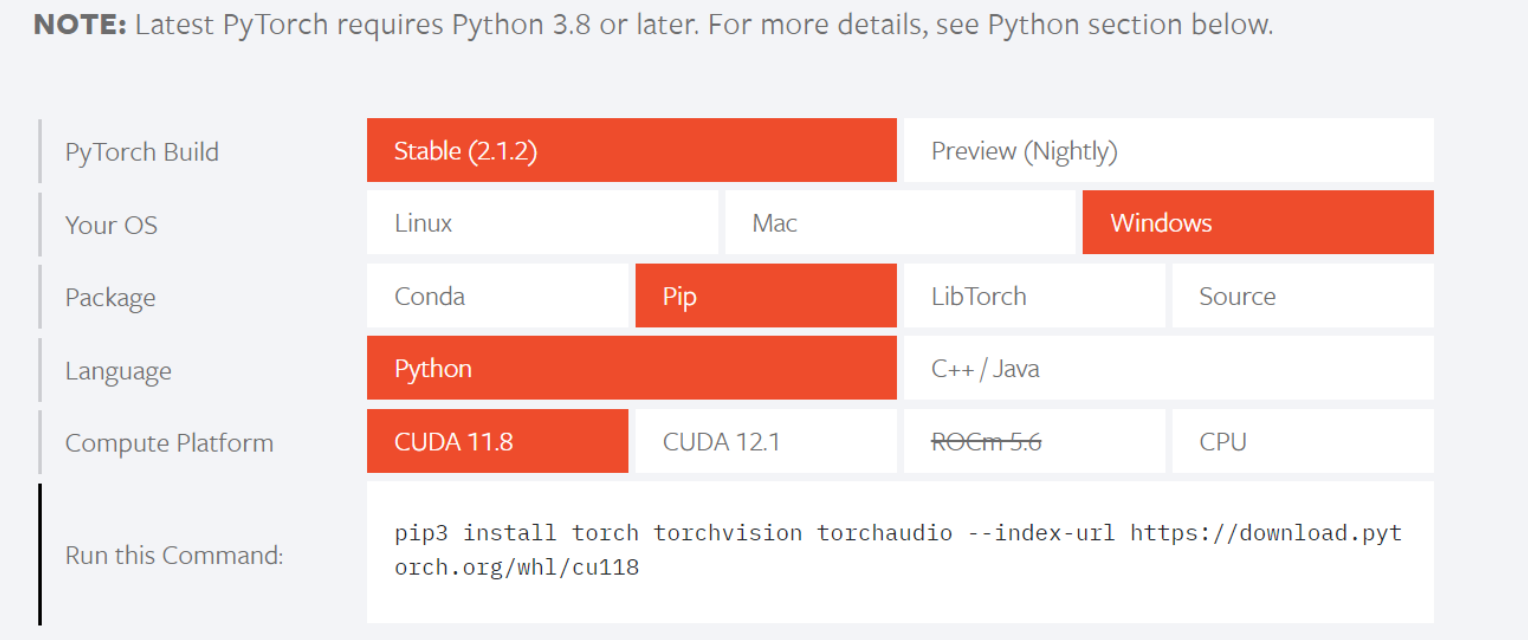
**2.conda create -n 环境名 python=3.x**

激活

conda activate 环境名

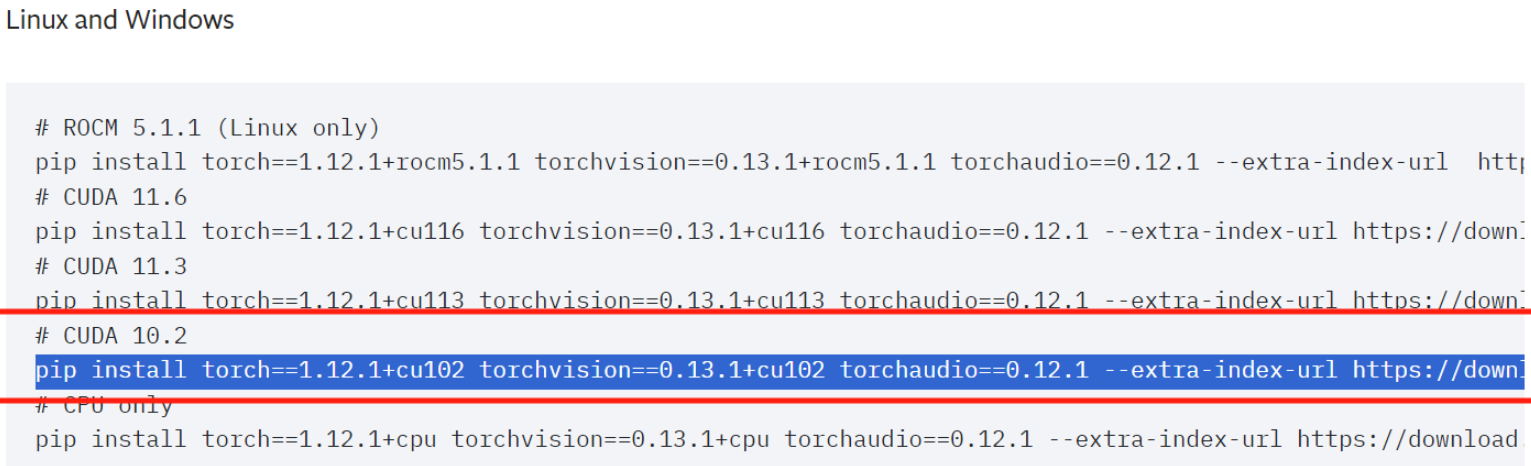
**PyTorch**

**打开pytorch官网**



**图3.1.1pytorch官网**

**在终端执行命令**



**图3.1.2 安装命令**

**yolov5环境**

进入<https://github.com/ultralytics/yolov5>，终端中执行，

git clonehttps://github.com/ultralytics/yolov5.git,或者Download ZIP下载代码

然后cd yolov5

pip install -r requirements.txt # install，配置所需环境

**标注工具（labeling）**

pip install labeling

python labelImg.py，可以打开图片文件夹开始数据标注工作

opencv > 4.6.0; openvino > 2024.3

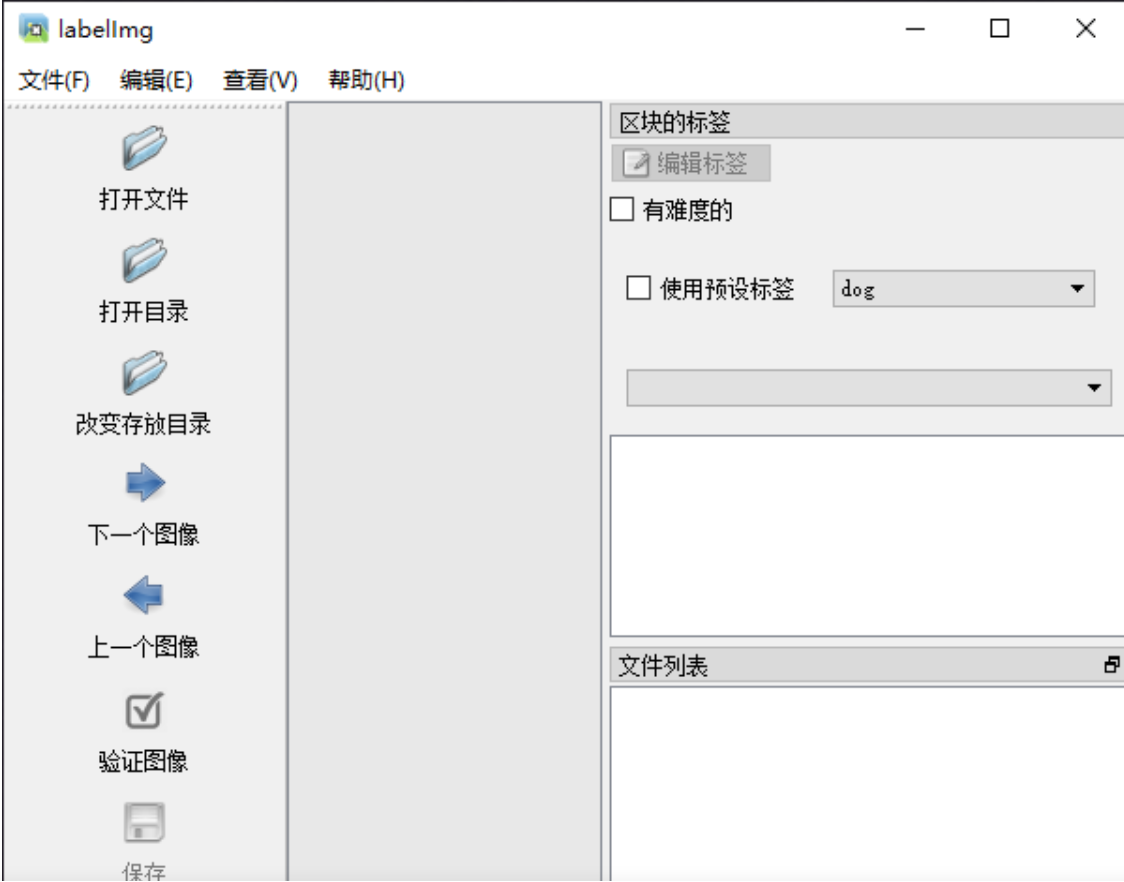


图3.1.3标注工具

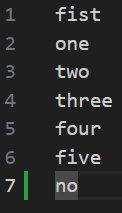
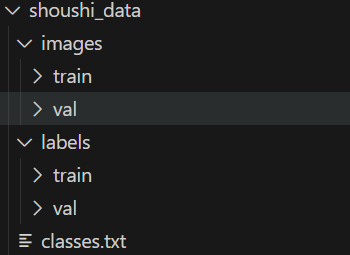
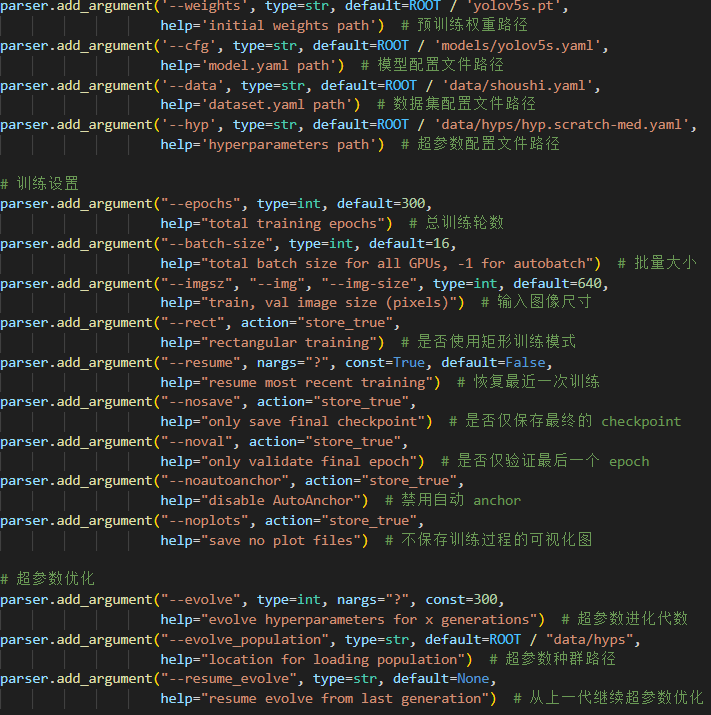


图3.1.3 数据集文件和类别

**3.3 设置训练参数**





**3.4 训练与结果**

**架构：**可以在netron.app中可视化模型的架构

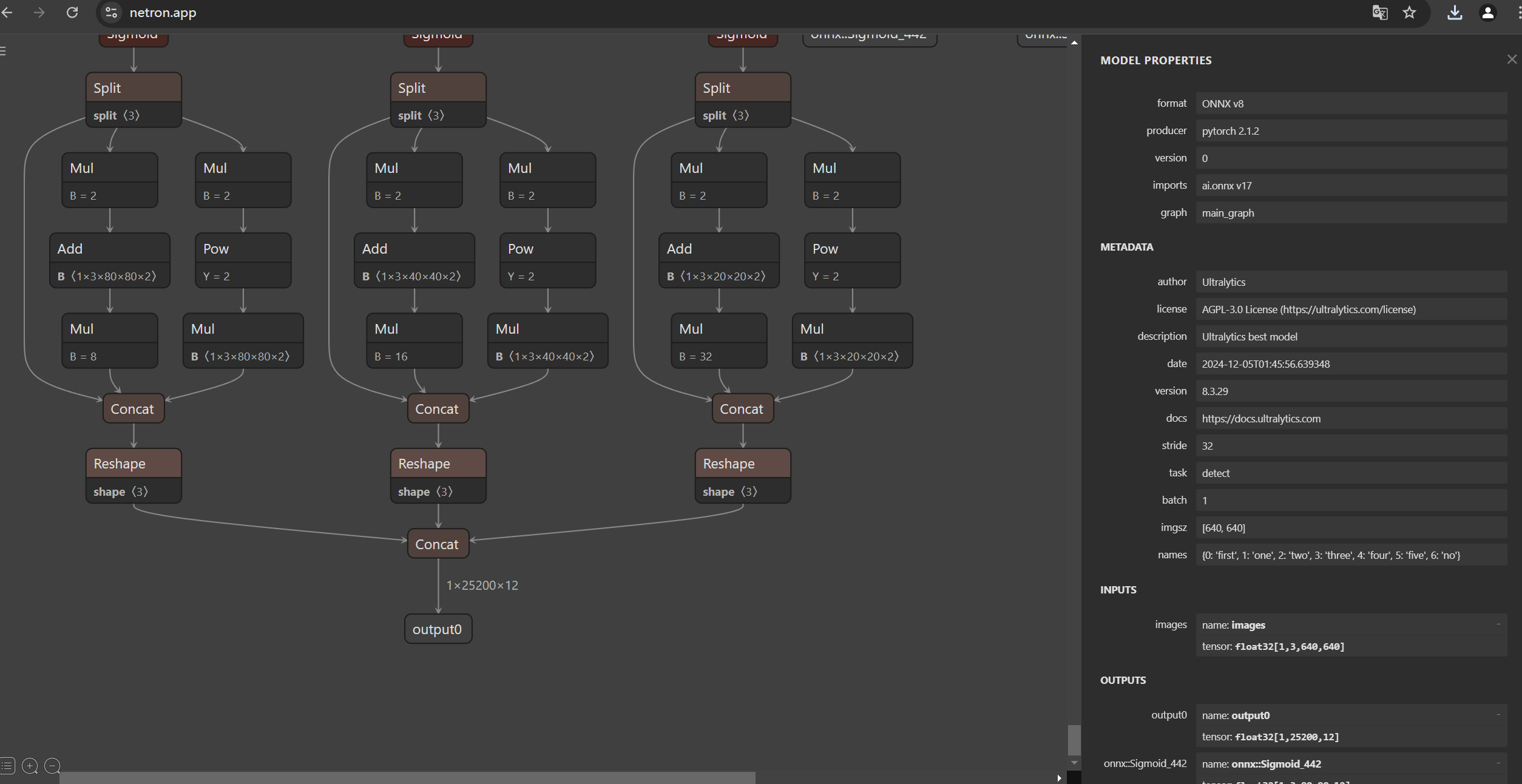


图3.4.1 模型架构

* 从图中可以看到，训练和验证的 box\_loss 都随着 epoch 增加逐渐减小，并趋于稳定，说明模型在学习目标的边界框预测任务上表现良好。训练和验证的 obj\_loss 明显下降并趋于稳定，说明模型逐渐适应了目标检测任务的置信度预测。损失持续下降并趋于稳定，说明模型在类别预测任务上逐渐提高了性能。mAP 快速上升并趋近于 1.0，说明模型在检测任务上表现优秀，且大部分目标能够被正确预测。
  + 最终稳定在 0.8 左右，略低于 mAP\_0.5，表明模型在更严格的 IoU 阈值下表现仍然不错。

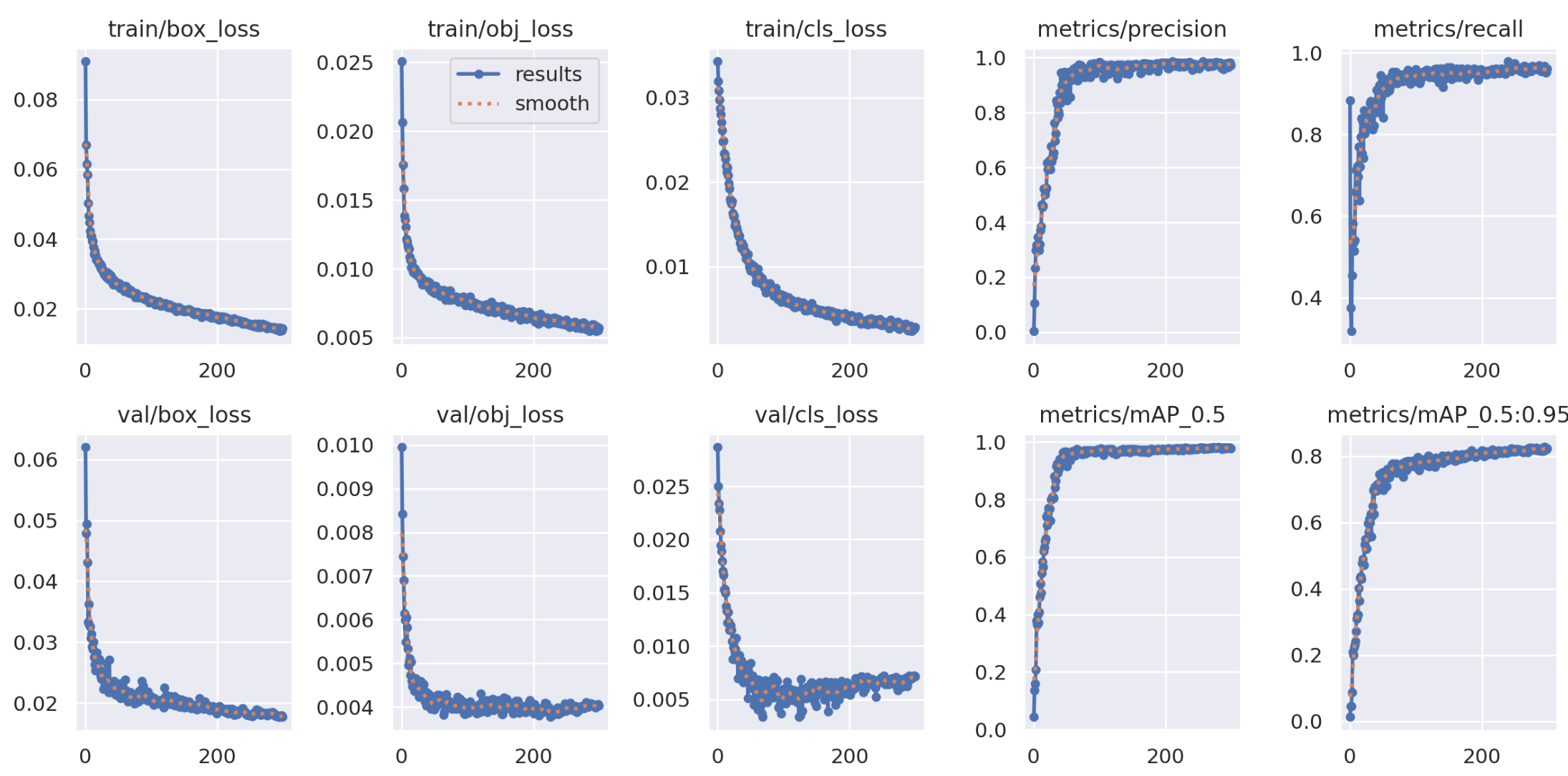


图3.4.2 result

模型的训练损失（如 box\_loss, obj\_loss, cls\_loss）和验证损失均随迭代次数显著下降，且在训练后期趋于稳定，表明模型收敛良好。评估指标如精度 (Precision)、召回率 (Recall)、以及 mAP (mean Average Precision) 均达到了较高水平，显示模型对目标检测任务的适配性强。

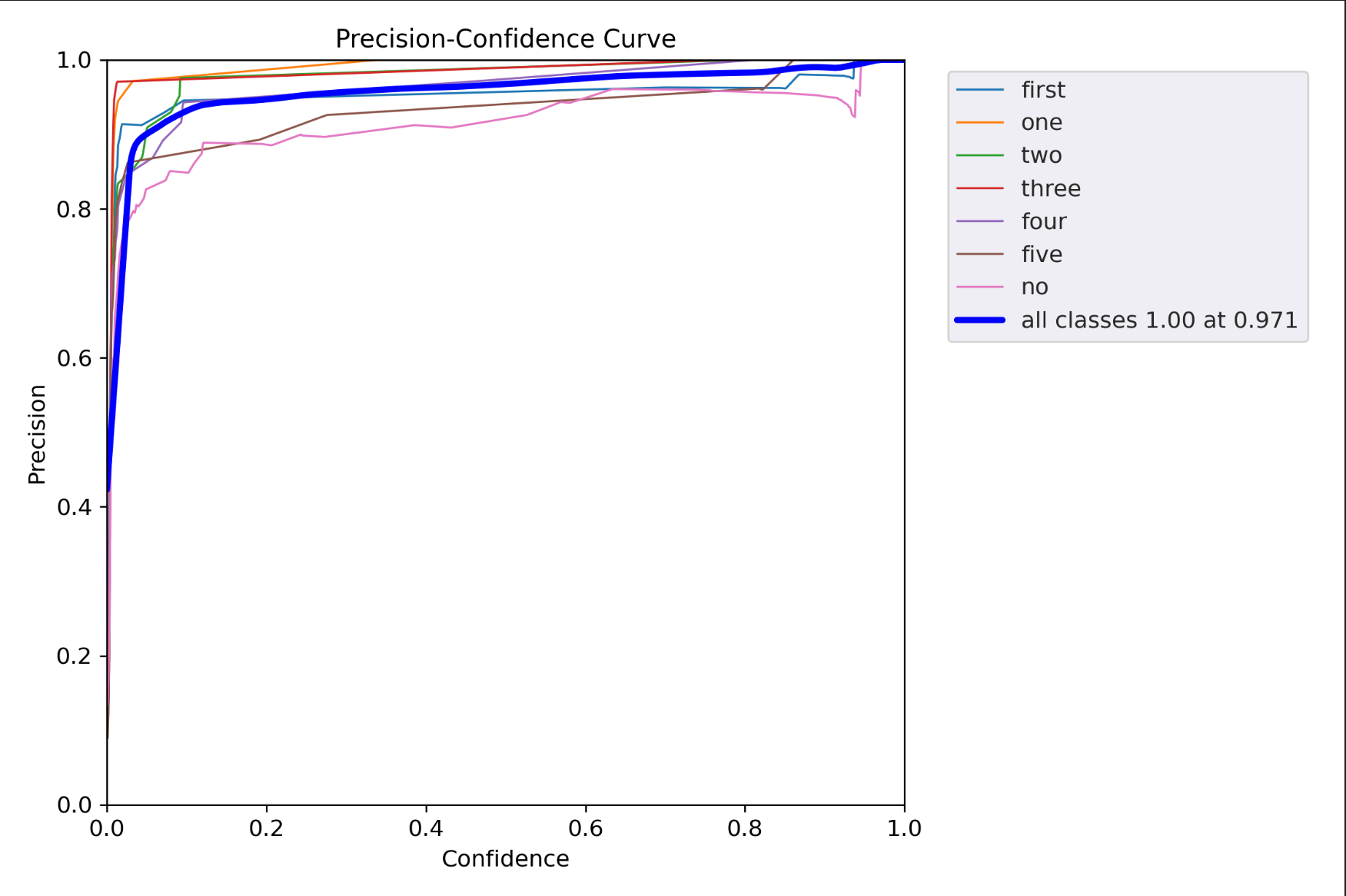


图3.4.3 Precision-Confidence

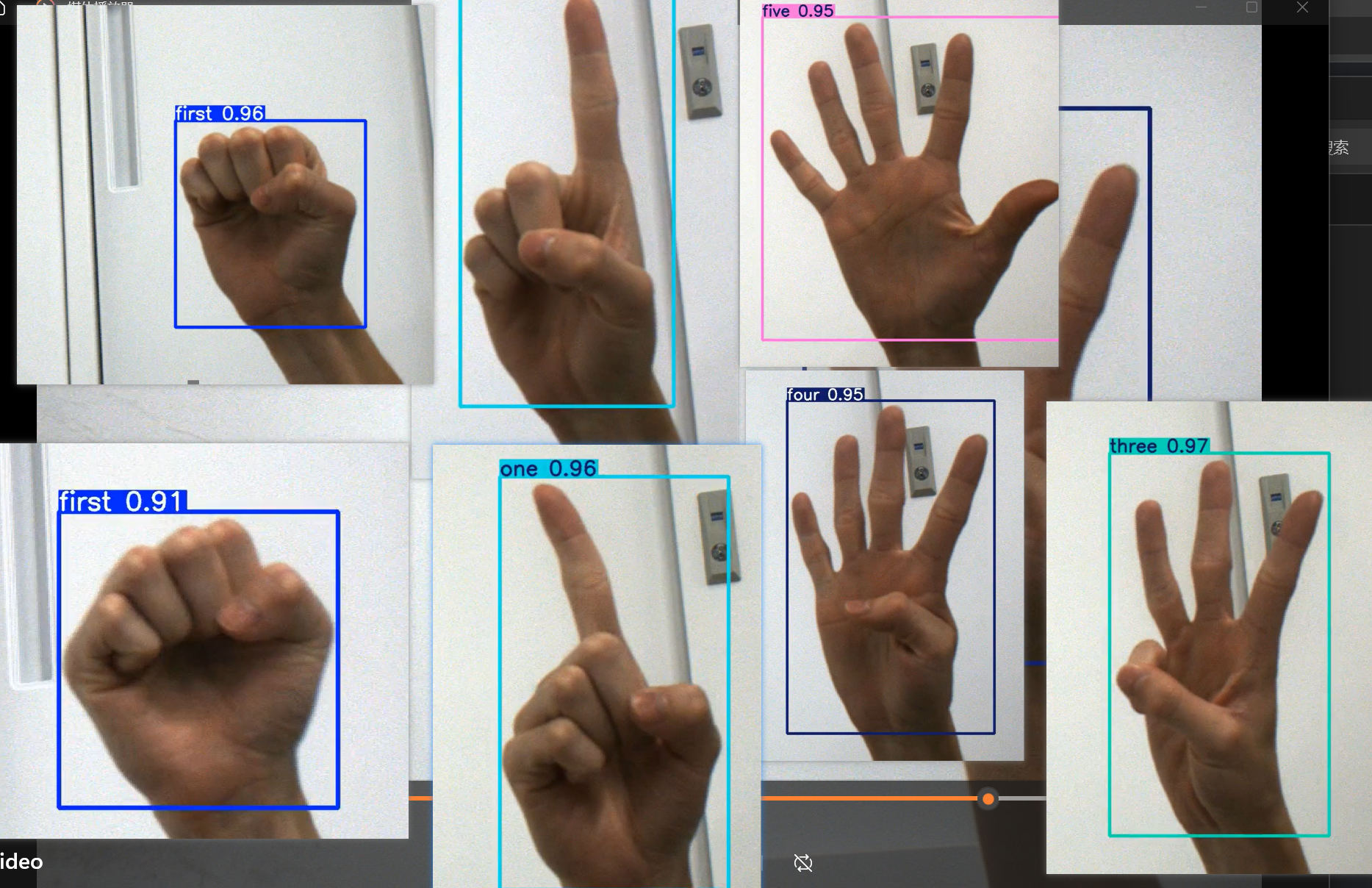


图3.4.4 结果

**四、模型部署**

**4.1 文件处理**

**PyTorch 模型转为 ONNX 格式**

加载已经训练好的模型（best.pt）,并将其切换到评估模式。根据模型的输入，构建一个模拟输入，通过pytorch的torch.onnx.export方法，将模型导出为ONNX格式的文件，由于onnx文件的跨框架兼容性：ONNX 支持主流的深度学习框架，如 PyTorch、TensorFlow、MXNet 等，模型可以在这些框架之间自由转换，避免了绑定某一框架的限制。

将ONNX模型转为OpenVINO格式

通过openvino（2024.3）版本的工具（ovc），将onnx模型转换成xml和bin文件。

 XML 文件：描述模型的结构。

 BIN 文件：包含模型的权重。

在转换的时候可以指定模型的输入形状、输出目录以及目标精度。

使用OpinVINO运行优化后的模型

使用 OpenVINO Runtime 加载生成的 XML 和 BIN 文件，根据需求可以悬着运行设备，例如CPU、GPU。然后就可以进行部署了

4.2 代码部署

1. 模型加载与预处理  
使用OpenVINO的Core接口加载XML和BIN格式的模型文件，初始化模型预处理器PrePostProcessor，并设置输入和输出的张量属性。

* 输入张量采用NHWC布局，颜色格式为BGR，随后转换为NCHW布局，并归一化至[0,1]范围。
* 输出张量数据类型为f32。

2. 图像预处理  
为保证输入图像与模型输入尺寸匹配，采用letterBox方法对图像进行缩放，同时保持纵横比。多余区域填充灰色，记录缩放比例与填充偏移用于后处理。

3. 推理执行

* 构造输入张量，将预处理后的图像加载到模型推理引擎中。
* 通过推理引擎执行前向传播，并提取推理结果。

4. 后处理与结果解析

* 将模型输出解析为二维张量，提取目标检测框及类别置信度，筛选满足置信度阈值的目标。
* 使用非极大值抑制（NMS）过滤重叠检测框，保留最佳候选。
* 对检测结果还原至原图比例，记录目标的边界框、类别、置信度及关键点信息。

5. 关键功能设计

* Letterbox缩放：在保持图像比例的同时调整尺寸，填充多余部分以适配模型输入。
* 非极大值抑制：通过置信度和NMS阈值过滤冗余框，提高检测结果的准确性。



图4.2.1 流程图

**提出推理请求**

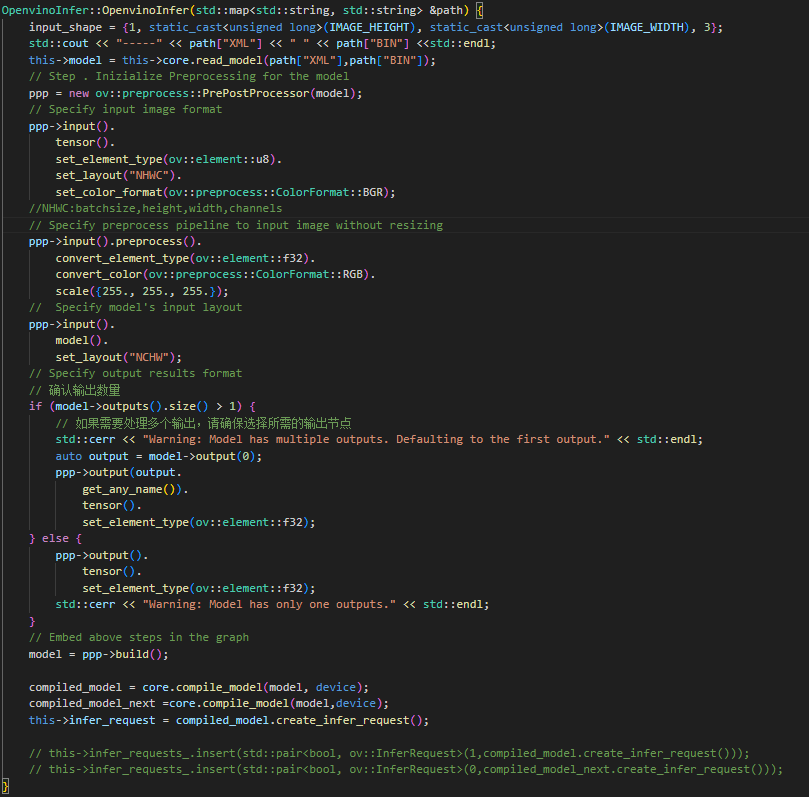


图4.2.2 推理请求

为了避免图像在resize过程中，由于图像像素比例不等，我采用letterbox方法来修改图像大小

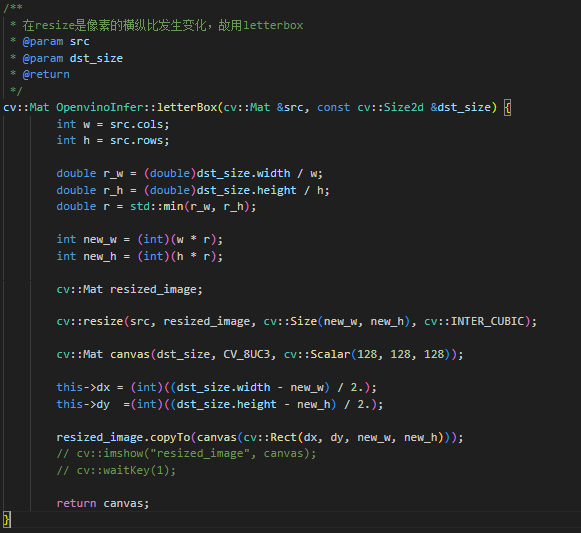
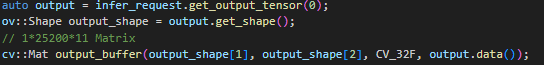


图4.2.3 letterbox修正图像

然后将图像降序成一个tensor输入，得到输出，通过设置置信度对softmax过的confidence经行筛选，通过判断概率决定类别，最后进过极大值抑制（NMS）得到最后的结果



**五、目标跟随**

受限于传感器选择了单目相机，且识别的物体没有固定大小，无法通过PNP（Perspective-n-Point）求解物体的位姿，.



所以笔者才用了最简单的控制方法来控制目标的跟随，那就是pid



只需让物体尽量处于图像的正中间遍可以有一个很好的效果



图5.1 pid代码实现

为了方便调参，写了一个工具filetoolkit实时读取参数文件，便可以做到实时调参的效果。



图5.2 实时调参代码实现

六、代码架构

