Deep Learning

[部署： 2](#_Toc15576)

[onnx： 2](#_Toc14371)

[torchscript： 2](#_Toc15739)

[openvino： 2](#_Toc26236)

[coco： 3](#_Toc9296)

[mediapipe： 3](#_Toc21628)

[模型solutions： 3](#_Toc2667)

[mindspore： 4](#_Toc19194)

[环境context： 4](#_Toc299)

[数据dataset： 4](#_Toc23084)

[网络nn： 4](#_Toc29409)

[optuna： 5](#_Toc14923)

[torch： 5](#_Toc31070)

[梯度autograd： 7](#_Toc24158)

[函数functional： 8](#_Toc25127)

[显卡cuda： 8](#_Toc12831)

[混合amp： 8](#_Toc5418)

[分布distributed： 8](#_Toc1123)

[激活gradcam： 9](#_Toc31261)

[通用utils： 9](#_Toc26989)

[网络nn： 9](#_Toc6651)

[函数functional： 11](#_Toc32006)

[初始化init： 12](#_Toc25910)

[剪枝utils.prune： 12](#_Toc27236)

[优化optim： 12](#_Toc3216)

[学习率lr\_scheduler： 12](#_Toc4121)

[通用utils： 12](#_Toc2499)

[关卡checkpoint： 12](#_Toc2211)

[数据data： 13](#_Toc7772)

[张量板tensorboard： 13](#_Toc25684)

[视觉torchvision： 13](#_Toc628)

[模型models： 13](#_Toc1964)

[数据集datasets： 13](#_Toc2604)

[运算ops： 13](#_Toc31072)

[转换transforms： 13](#_Toc27)

[拓展： 14](#_Toc1430)

[timm： 14](#_Toc9227)

[引擎mmengine： 14](#_Toc2094)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| float | 半精度 | exponent (5)、fraction (10)：CPU不支持 | | |
| 单精度 | exponent (8)、fraction (23) | | |
| 计算 | 正常 |  | |
| 指数全0 |  | |
| 指数全1 | 小数全0 |  |
| 小数不全0 |  |

visio图像转角 - {x: 76.1, y: 345.5}

部署：

onnx：

netron.start('f.onnx')

|  |
| --- |
| 对shape / size使用int转换 |
| 上采样使用scale\_factor指定参数 (而不是size) |
| reshape、view操作的batch维度为-1 |
| export的dynamic\_axes只指定batch维度 |

|  |  |  |
| --- | --- | --- |
| 模型导出torch.onnx： | | |
| export(model, args, f, input\_names, output\_names) | | 导出模型的onnx文件 |
| args：模型的输入 | f：导出文件名称 |  |
| torch.onnx.is\_in\_onnx\_export() | | 检查是否在导出模型 |

|  |  |  |
| --- | --- | --- |
| 模型调用onnxruntime： | | |
| get\_available\_providers() | | 获取可用设备 |
| InferenceSession(path\_or\_bytes) | | 实例化onnx文件的推断模型 |
| 实例方法 | get\_inputs() | 返回输入变量列表 |
| get\_outputs() | 返回输出变量列表 |
| run(ouput\_names, input\_feed) | 返回推理结果 |

torchscript：

|  |  |  |
| --- | --- | --- |
| 模型导出torch.jit： | | |
| load(file) | | 加载模型 |
| trace(model, args, strict) | | 创建模型 |
| 实例方法 | save(file) | 导出模型文件 |

openvino：

|  |  |
| --- | --- |
| mo命令： | |
| -w <input-model> | 指定模型文件 |
| -o <output-dir> | 指定输出目录 |
| --compress\_to\_fp16 | 指定半精度 |
| --input\_shape <shape> | 指定输入尺寸 |

**推理runtime：**

|  |  |  |
| --- | --- | --- |
| runtime.Core() | | 实例化内核 |
| 实例属性 | available\_devices | 可用设备 |
| 实例方法 | read\_model(xml) | 加载并返回模型 |
| compile\_model(model, device\_name) | 返回编译模型 |

|  |  |  |
| --- | --- | --- |
| runtime.CompiledModel(other) | | 实例化编译模型 |
| 实例方法 | input(idx) / output(idx) | 返回io结点 |
| \_\_call\_\_(\*arrays) | 前向传播 |

|  |  |  |
| --- | --- | --- |
| 结点： | | |
| 实例属性 | any\_name | 名称 |
| shape | 尺寸 |
| element\_type | 数据类型 |

coco：

from pycocotools.coco import COCO

|  |  |  |
| --- | --- | --- |
| COCO(annotation\_file) | | 初始化数据集 |
| 实例属性 | imgs | 图片信息 |
| anns | 标注实例 |
| cats | 类别信息 |
| 实例方法 | getAnnIds(imgIds) | 获取标注实例Id |

mediapipe：

模型solutions：

|  |  |  |
| --- | --- | --- |
| 检测器基类 | close() | 关闭检测器 |
| reset() | 重启检测器 |
| 关键点信息 | landmark | 返回关键点坐标 |

|  |  |  |  |
| --- | --- | --- | --- |
| 绘制： | | | |
| drawing\_utils | draw\_landmarks(image, landmark, connections,  landmark\_style, connection\_style) | OpenCV | 绘制关键点 |
| draw\_detection(image, detection) | 绘制检测框 |
| plot\_landmarks(landmark, connections) | Matplotlib | 绘制关键点 |
| drawing\_styles | get\_default\_hand\_landmarks\_style | 手掌 | 关键点样式 |
| get\_default\_hand\_connections\_style | 连接样式 |

|  |  |  |
| --- | --- | --- |
| 手掌hands： | | |
| HAND\_CONNECTIONS | | 关键点连接方式 |
| Hands(static\_image\_mode, max\_num\_hands,  model\_complexity, min\_detection\_confidence,  min\_tracking\_confidence) | | 实例化手掌关键点检测器 |
| 实例方法 | process(image) | 返回检测结果Output |
| Output  实例属性 | multi\_hand\_landmarks | 返回关键点绝对坐标列表 |
| multi\_hand\_world\_landmarks | 返回关键点相对坐标列表 |
| multi\_handedness | 返回手性检测结果 |
|  | | |

|  |  |  |
| --- | --- | --- |
| 人脸face\_detection： | | |
| get\_key\_point(Box, idx) | | 返回关键点坐标 |
| FaceDetection(min\_detection\_confidence, model\_selction) | | 实例化人脸关键点检测器 |
| 实例方法 | process(image).detections | 返回检测结果Box |

mindspore：

环境context：

|  |  |  |
| --- | --- | --- |
| context.setcontext(mode, devicd\_target="CPU") | | 配置运行环境 |
| mode：模式 | context.GRAPH\_MODE | 图模式 |
| context.PYNATIVE\_MODE |  |

数据dataset：

|  |  |  |
| --- | --- | --- |
| ds.GeneratorDataset(seq, column\_names=seq) | | 返回seq转换的数据集GD |
| 实例方法 | get\_dataset\_size() | 返回数据集大小 |
| batch(batch\_size) | 将指定数量数据组合成一批 |
| repeat(repeat\_size) | 数据集数量倍增 |

网络nn：

|  |  |
| --- | --- |
| 基类： | |
| Cell() | 神经网络基类 |
| GraphKernel() | 图形内核基类 |
| CellList() | 功能如list |
| SequentialCell() | MindSpore有序列表 |

|  |  |  |
| --- | --- | --- |
| 卷积网络： | | |
| Conv1d(in\_channels, out\_channels, kernel\_size, stride, padding, groups) | | 一维卷积层 |
| Conv1dTranspose(in\_channels, out\_channels, kernel\_size, stride, padding, groups) | | 一维反卷积 |
| Conv2d(in\_channels, out\_channels, kernel\_size, stride, padding, groups) | | 二维卷积层 |
| Conv2dTranspose(in\_channels, out\_channels, kernel\_size, stride, padding, groups) | | 二维反卷积 |
| in\_channals：输入通道数 | kernel\_size：卷积核尺寸 |  |
| out\_channels：输出通道数 | stride：步长 |  |

optuna：

|  |  |  |
| --- | --- | --- |
| 试验： | | |
| Trial(study, trial\_id) | | 初始化试验对象 |
| 实例属性 | params | 建议后的参数 |
| state | 状态 |
| 实例方法 | suggest\_int(name, low, high, step, log) | 获取建议的int |
| suggest\_float(name, low, high, step, log) | 获取建议的float |
| suggest\_categorical(name, choices) | 获取建议的类别 |

|  |  |  |
| --- | --- | --- |
| 研究： | | |
| create\_study(study\_name, direction) | | 初始化研究对象 |
| 形参 | direction: str = 'maximize' | 优化方向 |
| 实例属性 | best\_value | 最优函数值 |
| best\_params | 最优超参数 |
| best\_trial | 最优试验 |
| 实例方法 | ask() | 获取试验对象 |
| tell(trial, value) | 反馈试验结果 |
| optimize(func, n\_trials, show\_progress\_bar)  - func(Trial) -> float | 优化目标函数 |
| enqueue\_trial(dict) | 指定下次试验的参数 |
| add\_trials(trials) | 添加试验 |
| get\_trials(deepcopy) | 获取试验 |
| trials\_dataframe(attrs) | 获取试验数据表 |

|  |  |  |
| --- | --- | --- |
| 可视化visualization： | | |
| plot\_optimization\_history(study) | | 返回适应度曲线对象 |
| 实例方法 | show() | 通过端口展示图像 |

torch：

|  |  |
| --- | --- |
| pip安装： | |
| 1.9.0 | pip install torch==1.9.0+cu102 torchvision==0.10.0+cu102 -f https://download.pytorch.org/whl/torch\_stable.html |

flops, params = thop.profile(model, inputs, verbose=False)

# 测试浮点运算量、模型参数量

|  |  |  |
| --- | --- | --- |
| 张量属性访问： | | |
| x.shape | 尺寸 | |
| x.size(i) | 指定维度信息 | |
| x.min() / x.max() | 最小/大值 |  |
| x.argmin() / x.argmax() | 索引 |
| x.sum(dim) | 元素和tensor | |
| x.numel() | 元素数 | |
| x.dim() | 维度数 | |
| x.item() | 单标量 | |
| x.data | 数据 | |
| x.grad.data | 梯度数据 | |
| x.zero\_() | 梯度置0 | |

|  |  |
| --- | --- |
| 设备： | |
| device('cuda:0') | 实例化CPU设备 |
| x.to(device|type, non\_blocking=True) | 将实例在设备上运行/修改数据类型 |
| x.cpu() | 将实例在CPU上运行 |
| x.cuda() | 将实例在GPU上运行 |

|  |  |  |
| --- | --- | --- |
| 张量转换： | | |
| tensor(x, dtype=torch.float32) | 返回序列对应的tensor | |
| from\_numpy(x) |
| x.numpy() | 返回tensor对应的ARRAY | |
| x.tolist() | 返回tensor对应的列表 | |
| x.resize(\*sizes) | 返回尺寸转换后的tensor | |
| x.resize\_as(tensor) |
| x.view(\*shape) |
| x.float() | 数据类型转为 | float32 |
| x.byte() | uint8 |
| x.round\_() | 对tensor取最近整数 | |

|  |  |
| --- | --- |
| 张量操作： | |
| x.clone() | 深拷贝张量所有属性 |
| x.detach() | 张量脱离计算图 |
| x.contiguous() | 使张量在内存上连续 |

|  |  |  |
| --- | --- | --- |
| 张量维度变换： | | |
| x.flatten(start\_dim, end\_dim) | 将张量展平 | |
| x.unflatten(dim, sizes) | 将张量展开 | |
| narrow(x, dim, start, length) | 截取矩阵块 | |
| flip(x, dims) | 按维度翻转 | |
| x.unsqueeze(dim) | 添加新维度 | |
| x.squeeze(dim) | 删除冗余维度 | |
| x.repeat(\*shape\_blow) | 各维度数据复制扩增 | |
| x.transpose(dim1, dim2) | 交换对应维度 | |
| x.permute(\*dims) | 重排维度 | |
| cat(xs, dim) | 在原维度 | 合并tensor |
| stack(xs, dim) | 创建维度 |
| x.split(lengths, dim) | 按长度 | 分割张量 |
| x.chunk(chunks, dim) | 按块 |

|  |  |  |
| --- | --- | --- |
| 张量创建： | | |
| zeros(shape) | 返回指定  形状tensor | cells = 0 |
| empty(shape) | cells -> 0 |
| ones(shape) | cells = 1 |
| eye(shape) | 单位矩阵 |
| full(shape, element) | cells = element |
| arange(min, max, h) | 返回行向量 | [min, max) 步长h |
| linspace(min, max, steps) | [min, max] 长度steps |

|  |  |  |
| --- | --- | --- |
| 随机： | | |
| manual\_seed(idx) | 选定指定的随机数初态 | |
| rand(\*size) | 返回均匀分布 |  |
| randint(low, high, size) |  |
| x.uniform\_(low, high) |
| randn(\*size) | 返回正态分布 |  |
| normal(mu, sigma, size) |  |
| randperm(n) | [0, n) 整数，顺序打散 | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 矩阵公式： | | | | | | | |
| sqrt(x) |  | log(x) |  | log2(x) |  | log10(x) |  |
| abs(x) | |x| | sin(x) |  | cos(x) |  | tan(x) |  |
| exp(x) |  |  |  |  |  | erf(x) |  |

|  |  |
| --- | --- |
| 张量运算： | |
| x1 @ x2 | 返回矩阵乘法结果 |
| matmul(x1, x2) |
| x.add\_() | 就地运算 |
| x.mul\_() |
| clamp(x, min, max) | 返回数值收缩到 [min, max] 的tensor |
| norm(x, dim) | 返回二范数 |
| cosine\_similarity(x1, x2, dim) | 返回余弦相似度 |

|  |  |
| --- | --- |
| 功能： | |
| nonzero(x) | 筛选出非零元素 |
| scatter(x, dim, index, value) | 按位置填充值 |
| equal(x1, x2) | 判断相等 |
| where(bool, true\_out, false\_out) | 返回布尔值对应tensor |
| x.unique(sorted, dim) | 返回不重复元素的张量 |
| sort(x, dim, descending) | 返回排序结果 |
| x.backward() | 反向传播梯度 |
| bernoulli(x, p) | 生成伯努利分布 |
| meshgrid(tensors) | 返回多维坐标网格点矩阵 |

|  |  |
| --- | --- |
| 通用： | |
| save(data, file) | 将data存储到file |
| load(file, map\_location) | 读取file并返回存储的数据 |
| no\_grad() | 实例化零梯度上下文管理器 |
| set\_printoptions(precision, sci\_mode) | 设置输出格式 |

梯度autograd：

|  |  |
| --- | --- |
| set\_detect\_anomaly(True) | 设置梯度异常检测器 |

|  |  |  |
| --- | --- | --- |
| ctx | | 上下文管理器 |
| 实例属性 | saved\_tensors | 获取用于反向传播的张量 |
| 实例方法 | save\_for\_backward(\*args) | 保存用于反向传播的张量 |

|  |  |  |
| --- | --- | --- |
| Function() | | 实例化传播函数 |
| 静态方法  (重写) | forward(ctx, \*args) | 前向传播 |
| backward(ctx, \*grad\_outputs) | 反向传播 |
| 类方法 | apply(x) | 调用 |

函数functional：

import torch.autograd.functional as aF

|  |  |
| --- | --- |
| jacobian(func, inputs) | 生成函数的雅可比矩阵 |

显卡cuda：

|  |  |
| --- | --- |
| 显卡操作： | |
| empty\_cache() | 释放显卡缓存 |
| synchronize() | 等待所有协程完成 |

|  |  |
| --- | --- |
| 信息获取： | |
| is\_available() | 显卡是否可用 |
| device\_count() | 显卡数量 |
| memory\_reserved(i) | 返回当前的GPU占用量 |

|  |  |  |
| --- | --- | --- |
| 显卡实例： | | |
| get\_device\_properties(i) | | 返回显卡实例 |
| 实例属性 | name | 显卡名称 |
| total\_memory | 显存大小 |

混合amp：

|  |  |
| --- | --- |
| autocast(enabled) | 实例化混合精度训练装饰器 |

|  |  |  |
| --- | --- | --- |
| GradScaler() | | 实例化梯度缩放器 |
| 实例方法 | scale(loss) | 缩放损失值 |
| step(optim) | 装饰优化器，检查非有限梯度 |
| update() | 启动迭代 |

分布distributed：

|  |  |  |
| --- | --- | --- |
| 环境变量： | | |
| CUDA\_VISIBLE\_DEVICES | cuda可见设备 | '0,1,2'：多 cuda |
| ''：GPU |
| RANK | 进程在所有进程中的级别 | |
| LOCAL\_RANK | 进程在本地进程中的级别 | |
| WORLD\_SIZE | 进程组中的进程数量 | |

|  |  |
| --- | --- |
| init\_process\_group('nccl', world\_size, rank) | 初始化线程池 |

from torch.utils.data.distributed import DistributedSampler

|  |  |  |
| --- | --- | --- |
| DistributedSampler(dataset, num\_replicas, rank) | | 实例化分布式数据集 |
| 实例方法 | set\_epoch(epoch) | 设置训练轮次 |

from torch.nn.parallel import DistributedDataParallel as DDP

|  |  |  |
| --- | --- | --- |
| DDP(module, device\_ids, output\_device) | | 实例化分布式数据并行器 |
| 实例属性 | module | 网络模型 |
| 实例方法 | train(mode) |  |

激活gradcam：

import pytorch\_grad\_cam as gradcam

|  |  |  |
| --- | --- | --- |
| EigenGradCAM(model, target\_layers, use\_cuda) | | 返回Grad-CAM绘制器 |
| 实例方法 | module | 网络模型 |

通用utils：

|  |  |
| --- | --- |
| 目标model\_targets： | |
| ClassifierOutputTarget(category).\_\_call\_\_(output) | 返回对应类别得分 |

网络nn：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 通用： | | | | |
| Parameter(x, requires\_grad) | | 实例化模型参数 | | |
| Sequential(\*modules) /  ModuleList(list) / ModuleDict(dict\_or\_items) | | 实例化网络模块容器 | | |
| Module() | | 网络模块基类 | | |
| 实例属性 | parameters() | 参数生成器 | | |
| named\_parameters | 参数字典生成器 | | |
| \_modules | 子模块字典 | | |
| modules() | 子模块生成器 | | |
| state\_dict() | 模型参数生成器 | | |
| 实例方法  (重写) | forward(\*args) | 前向传播方法 | | |
| extra\_repr() | 额外信息 | | |
| 实例方法 | load\_state\_dict(dict, strict) | 加载模型参数 | | |
| apply(fn: Callable[module]) | 递归地将函数应用于模块 | | |
| train() / eval() | 运算模式切换 | | |
| half() / float() / fuse() / type(dtype) | 数据类型切换 | | |
| add\_module(name, module) | 添加模块到实例属性 | | |
| register\_buffer(name, tensor) | 注册网络成员变量 | | |
| register\_forward\_hook  (hook: Callable[module, x, output]) | 注册挂钩  返回handle | .remove()  释放 | 前传挂钩 |
| register\_full\_backward\_hook  (hook: Callable[module, gx, gy]) | 反传挂钩 |

|  |  |
| --- | --- |
| 基本模块 shape = [B, D]： | |
| Dropout(proba, inplace) | 随机置零输入值 |
| Identity() | 恒等映射 |
| Linear(in\_features, out\_features) | 线性变换 |

|  |  |
| --- | --- |
| timm.models.layers： | |
| to\_2tuple(x) | 转化成二元组 |
| DropPath(drop\_prob) | 随机置零样本 |
| DropBlock2d(drop\_prob, block\_size, batchwise) | 随机置零连续区域 |

|  |  |  |
| --- | --- | --- |
| CNN模块 shape = [B, C, H, W]： | | |
| Conv1d(in\_ch, out\_ch, kernel\_size, stride, padding, dilation) | | 卷积 |
| Conv2d(in\_ch, out\_ch, kernel\_size, stride, padding, dilation) | |
| ConvTranspose2d(in\_ch, out\_ch, kernel\_size, stride, padding) | | 反卷积 |
| Upsample(scale\_factor, mode='nearest') | | 上采样 |
| AdaptiveAvgPool2d(ouput\_size) | | 平均池化 |
| AvgPool2d(kernel\_size, stride, padding) | |
| MaxPool2d(kernel\_size, stride, padding) | | 最大池化 |
| BatchNorm2d(channels, eps, momentum) | | 批标准化 |
| GroupNorm(groups, channels) | | 组标准化 |
| in\_channels：输入通道数 | kernel\_size：卷积核尺寸 |  |
| out\_channels：卷积核通道数 | stride：步长 |  |
| padding：填充规模 | dilation：膨胀尺寸 |  |



|  |  |
| --- | --- |
| Transformer： | |
| MultiheadAttention(embed\_dim, num\_heads,  dropout, bias=False) | 多头注意力网络 (dropout在softmax之后) |
| head\_dim = embed\_dim / num\_heads |
| 单头注意力： |
| TransformerEncoderLayer(d\_model, nhead,  dim\_feedforward, dropout, activation) | 编码器 |
| TransformerDecoderLayer(d\_model, nhead,  dim\_feedforward, dropout, activation) | 解码器 (比编码器多了对memory的MSA) |
| LayerNorm(normalized\_shape) | 层标准化 |
| Transformer(d\_model, nhead,  num\_encoder\_layers, num\_decoder\_layers,  dim\_feedforward, dropout, activation) | 编码器+解码器 |

|  |  |
| --- | --- |
| NLP模块 shape = [L, B, D]： | |
| Embedding(num\_embeddings, embedding\_dim) | 数据降维 |
| RNN(input\_size, hidden\_size, num\_layers, batch\_first) | 循环网络 |
| RNNCell((input\_size, hidden\_size) | 循环网络单元 |
| LSTM(input\_size, hidden\_size, num\_layers, batch\_first) | 循环网络 |

函数functional：

import torch.nn.functional as F

|  |  |
| --- | --- |
| 功能函数： | |
| one\_hot(x, num\_classes) | 返回one hot矩阵 |
| normalize(x, p, dim) | 向量规范化 |

Logit变换：

|  |  |  |
| --- | --- | --- |
|  | 激活函数 | 值域 |
| x.sigmoid(x) |  | [0, 1] |
| tanh(x) |  | [-1, 1] |
| hardtanh(x, min, max) |  | [min, max] |
| softplus(x, beta) |  | [0, ∞] |
| leaky\_relu(x, negetive\_slope) |  | [-∞, ∞] |
| mish(x) |  |  |
| relu(x) |  | [0, ∞] |
| silu(x) |  | [0, ∞] |
| gelu(x) |  |  |

|  |  |
| --- | --- |
| 损失函数： | |
| softmax(x, dim) | 相对概率： |
| log\_softmax(x, dim) | 相对概率： |
| mse\_loss(x, y, reduction='sum') | 均方差： |
| nll\_loss(x, y) | 负对数似然： |
| cross\_entropy(x, y, weight) | 交叉熵：softmax → nll\_loss |
| binary\_cross\_entropy(x, y) | 二元交叉熵： |
| binary\_cross\_entropy\_with\_logits(x, y) | 二元交叉熵： |

|  |  |  |
| --- | --- | --- |
| 图像函数： | | |
| interpolate(input, scale\_factor, mode="nearest") | | 向上/向下采样指定倍数 |
| adaptive\_avg\_pool2d(input, shape) | | 自适应平均池化，得到目标尺寸图像 |
| 形参 | scale\_factor | 比例因子 |

初始化init：

|  |  |
| --- | --- |
| trunc\_normal\_(tensor, mean, std) | 截断正态分布，初始化weight |
| constant\_(tensor, val) | 填充定值，初始化bias |

剪枝utils.prune：

|  |  |
| --- | --- |
| global\_unstructured(param, method=L1Unstructured, \*\*kwargs) | 全局非结构化剪枝 |
| l1\_unstructured(module, name, amount) | 非结构化剪枝 |
| ln\_structured(module, name, amount, n, dim) | 结构化剪枝 |

优化optim：

|  |  |  |
| --- | --- | --- |
| Optimizer(params, defaults) | | 优化器基类，重写 step |
| [SGD](https://pytorch.org/docs/stable/generated/torch.optim.SGD.html)(params, lr, momentum, weight\_decay) | | 实例化SGD算法优化器 |
| [Adam](https://pytorch.org/docs/stable/generated/torch.optim.Adam.html)(params, lr, betas, eps, weight\_decay) | | 实例化Adam算法优化器 (在梯度上权重衰减) |
| [AdamW](https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html)(params, lr, betas, eps, weight\_decay) | | 实例化AdamW算法优化器 (直接权重衰减) |
| 实例方法 | add\_param\_group(dict) | 添加参数组 |
| step() | 使用优化器优化变量 |
| zero\_grad() | 梯度信息置零 |
| lr：学习率 | momentum：动量 (0.78) | weight\_decay：L2 范数正则项系数 |

学习率lr\_scheduler：

|  |  |  |
| --- | --- | --- |
| ReduceLROnPlateau(optimizer, factor, patience, threshold, min\_lr) | | 实例化损失监听器 |
| 实例方法 | step(loss) | 若loss稳定则更新学习率 |

|  |  |  |  |
| --- | --- | --- | --- |
| StepLR(optimizer, step\_size, gamma) | | 实例化按步更新器 | step\_size：优化次数 |
| gamma：学习率缩放 |
| LambdaLR(optimizer, lr\_lambda, last\_epoch) | | 实例化自定义更新器 | |
| 实例方法 | step() | 满足条件则更新学习率 | |

通用utils：

关卡checkpoint：

|  |  |
| --- | --- |
| checkpoint(func, \*args) | 执行运算，不保留中间激活值，节约内存空间 |

数据data：

|  |  |
| --- | --- |
| Dataset() | 数据集基类，重写\_\_init\_\_、\_\_getitem\_\_、\_\_len\_\_ |

|  |  |  |
| --- | --- | --- |
| DataLoader(dataset, batch\_size, shuffle, num\_workers, drop\_last) | | 返回分批加载器 |
| batch\_size：批大小 | shuffle：是否打乱 |  |
| num\_workers：线程数 | drop\_last：是否舍去余数 |  |

|  |  |
| --- | --- |
| random\_split(DS, [\*int]) | 随机拆分数据集，返回数据集元组 |

张量板tensorboard：

|  |  |  |
| --- | --- | --- |
| SummaryWriter(log\_dir) | | 实例化SW编辑器 |
| 实例方法 | add\_image(str, img, step, dataformats="HWC") | 添加图像 |
| add\_scalar(str, y, x) | 添加标量 |
| close() | 关闭编辑器 |

视觉torchvision：

模型models：

ResNet, VGG, ShuffleNet, DenseNet, MnasNet, MobileNet, SqueezeNet

数据集datasets：

|  |  |  |
| --- | --- | --- |
| CIFAR10(root, train, transform, download) | | 实例化CIFAR数据集 |
| 形参 | train | 是否为训练集 |
| transform | 转换器实例 |
| download | 是否下载 |
| 实例属性 | classes | 种类列表 |
| 实例方法 | \_\_getitem\_\_(item) | 返回 [img, target] |

运算ops：

|  |  |
| --- | --- |
| box\_iou(xyxy1, xyxy2) | 边界框IoU |
| nms(boxes, scores, iou\_threshold) | 不区分类别非极大抑制 |
| batched\_nms(boxes, scores, idxs, iou\_threshold) | 分类别非极大抑制 |

转换transforms：

|  |  |  |  |
| --- | --- | --- | --- |
| Compose(trans\_list) | 实例化变换器实例的组合变换器 | | |
| ToTensor() | 实例化tensor转化器，并归一化 | | |
| Normalize(mean, std) | 实例化归一化器： | | |
| Resize([height, width]) | 实例化尺寸转换器 | | |
| RandomApply(transforms, p) | 实例化随机应用器 | | |
| RandomHorizontalFlip(proba) | 实例化随机镜像器 | 水平方向 | proba：概率 |
| RandomVerticalFlip(proba) | 竖直方向 |
| RandomRotation([min, max], fill\_color) | 实例化随机旋转器 | | |
| RandomCrop(shape) | 实例化随机裁剪器 | | |
| RandomAffine(rotate, translate, scale, fill\_color) | 实例化随机仿射变换器 | | |
| ColorJitter(brightness, contrast, saturation, hue) | 实例化色彩变化器 | | |
| Grayscale(num\_out\_channels) | 实例化灰度转换器 | | |
| GaussianBlur(kernel\_size, sigma) | 实例化高斯模糊器 | | |

拓展：

mmengine：

实例可使用字典表示，type特指父类，其它key表关键字参数

|  |  |  |
| --- | --- | --- |
| 配置Config： | | |
| 形参 | cfg\_dict | 配置字典 |
| 类方法 | from\_file(file) | 文件 -> 配置工具 |
| 实例方法 | dump(file) | 配置工具 -> 文件 |

评估evaluator：

|  |  |  |
| --- | --- | --- |
| 度量BaseMetric： | | |
| 实例方法  (重写) | process(self, data\_batch, data\_samples) | 处理过程数据，并存储在self.results |
| compute\_metrics(self, results) -> dict | 根据过程数据，计算并返回性能度量 |

模型model：

|  |  |  |
| --- | --- | --- |
| 模型BaseModel： | | |
| 实例方法  (重写) | \_\_init\_\_(self, \*args, \*\*kwargs) | 设置实例属性 |
| forward(\*\*kwargs, mode) | mode='loss'：dict  mode='predict'：any |

执行器runner：

|  |  |  |
| --- | --- | --- |
| 训练循环： | | |
| EpochBasedTrainLoop | | 基于epoch |
| IterBasedTrainLoop | | 基于iter |
| 形参 | max\_epochs / max\_iters: int | 训练总轮次 |
| val\_begin: int | 开始验证的时间 |
| val\_interval: int | 验证轮次的间隔 |

|  |  |  |
| --- | --- | --- |
| 执行器Runner： | | |
| 形参 | model: nn.Module | 模型对象 |
| workdir: str | 工作目录，保存参数、日志 |
| load\_from: str | 检查点文件 |
| experiment\_name: str |  |
| train\_dataloader /  val\_dataloader: Dataloader | 数据集加载器 |
| train\_cfg: dict | 'by\_epoch' 指定训练TrainLoop |
| val\_evaluator: Evaluator / dict / list | 验证评价指标 |
| optim\_wrapper: dict | 'optimizer' 配置迭代器 |
| param\_scheduler: dict | 学习率步进包装 |
| auto\_scale\_lr: dict | 学习率缩放  {'base\_batch\_size': None, 'enable': True} |
| resume: bool | 还原训练进程 |
| 类方法 | from\_cfg(cfg) | 通过配置文件创建 |
| 实例方法 | train() -> nn.Module | 启动训练进程 |
| val() -> dict | 启动验证进程，返回度量 |

timm：

层models.layers：

|  |  |
| --- | --- |
| to\_2tuple(arg) / to\_ntuple(n)(arg) | 返回参数的n元组 |
| DropPath(drop\_prob, block\_size) | 批随机置零 |