

目录

目录	1
Pipeline	2
Pipeline中的连接	2
Pipeline中的节点	2
Notebook	2
启动Notebook	2
停止Notebook	2
使用Notebook	2
算法开发	2
environment.yml	2
environment.yml > version	3
environment.yml > image	3
environment.yml > train{test, infer, deploy, compile}	3
parameter.yml	3
Helper Package	3
Dataset Package	3
写数据	3
读数据	3
代码改写	4
QA:	5
语义分割算法	5
上传数据	5
处理数据	5
训练等	7
分类识别算法	7
上传数据	7
处理数据	8
训练等	9
检测算法	10
上传数据	10
处理数据	10
训练等	12

Pipeline

在KPL运行算法等，有两种方式，第一种是通过构造Pipeline，第二种是使用JupyterLab。在这一小节介绍Pipeline的用法。Pipeline的优点是，对于成熟的业务场景，可以通过一次构造Pipeline，以此为模板，多次快速高效使用。

Pipeline中的连接

节点之间的连接，均为数据文件，数据文件主要为：数据集和模型文件。通过容器技术将连接确定的依赖解析为将数据集或模型挂载到目标路径。

Pipeline中的节点

节点包括几种类型的节点：数据集节点，模型节点，算法节点，处理器节点。

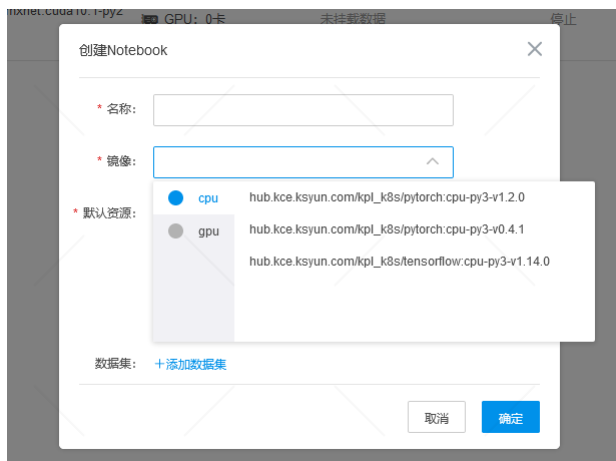
1. 数据集节点：也就是用户上传的数据集，数据集均为通过kpl-dataset包序列化后的数据集（可类比为TensorFlow的TFRecords）
2. 模型节点：用户上传的模型文件，可能为预训练模型或可直接用于生产的模型
3. 算法节点：用于训练，测试，批量推理，提供API服务的算法节点
4. 处理器节点：对数据进行加工处理的节点，如解析数据集的XML文件

每个节点代表一次对输入数据文件的处理，然后输出相应的结果，传递给下一个节点再次进行加工处理。算法节点支持自定义开发，具体方法参考算法开发小节。

Notebook

系统使用JupyterLab作为Web端在线代码开发环境。您可以在线即用即开，随时启停，弹性利用计算资源，workspace中的所有文件都会持久化到分布式存储中，再次启动时能快速恢复环境进行算法开发。

启动Notebook



1. 选择您需要的镜像
申请GPU资源而使用GPU镜像，可能会导致import 框架包时失败。
2. 按需申请您需要的计算资源 表单中填写的计算资源均为使用上限，CPU和GPU资源如果在使用过程中达到上限会导致算法运行变慢卡顿，内存超过使用上限可能会发生OOMKilled错误导致失败。另外资源的占用为排他性占用，特别是GPU资源，因此最好合理根据算法的需要申请资源，避免申请过少影响性能，申请太多浪费资源。

不同的镜像中包含了不同的深度学习框架，以及相应的Python软件依赖，注意如果没有

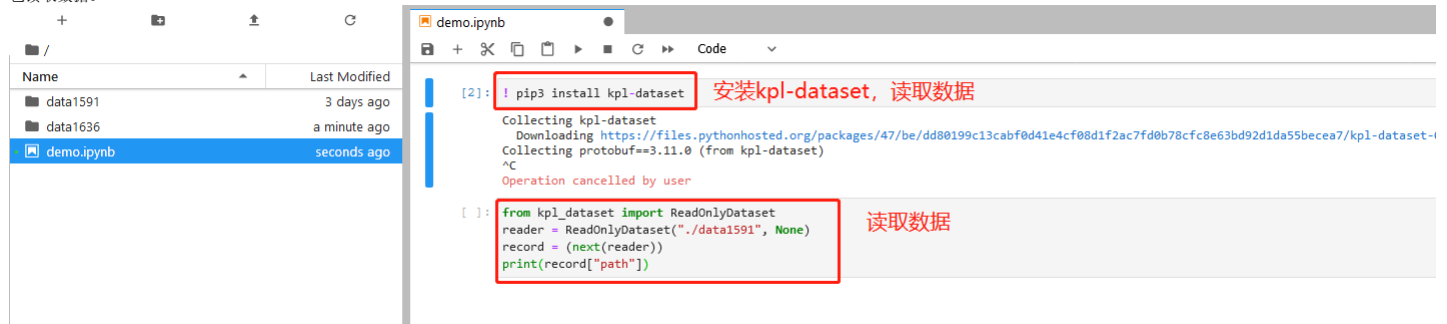
停止Notebook

为避免干预您使用Notebook，系统默认不会自动停止您的Notebook，Notebook在运行期间一直会锁定计算资源，其他任务均不可用。如果不再需要运行，请手动停止Notebook。

使用Notebook

最常用的功能为Terminal和Notebook，启动方式如下图：

在Notebook中使用数据，可以直接在Notebook中上传自己的数据。也可以挂载平台中已有的数据集，但是平台中的数据集均以kpl-dataset提供的序列化方式进行存储，您可以使用kpl-dataset Python包读取数据。



算法开发

平台中的算法运行依赖于容器运行环境，理论上能制作成Docker镜像的算法都能在平台上运行。要开发出能在平台上安装运行的算法，需要对算法进行以下几项适配：

1. 将算法所需的运行环境（如package依赖），输入输出数据等以environment.yml文件进行描述。
2. 如算法有需要经常调整的超参数，将超参数以parameter.yml文件进行描述。
3. 为使用以上两个配置文件，需对代码进行少量改写，届时会用到两个Python工具包：kpl-helper和kpl-dataset。

environment.yml

完整的environment.yml示例

```
version: 1
image:
  from_image: hub.kce.ksyun.com/kpl_k8s/pytorch:cuda10.0-py3-v1.2.0
  runs:
    - pip3 install kpl-dataset kpl-helper matplotlib==2.2.4 Pillow==5.0.0
      terminaltables==3.1.0 -i https://pypi.tuna.tsinghua.edu.cn/simple
train:
  resource:
  limits:
```

```

cpu: 4
gpu: 1
memory: 10240 # 单位MB
shm: 1024 # 共享内存, 单位MB
requests: # requests选项
cpu: 4
gpu: 1
memory: 10240
shm: 1024
inputs:
- key: data
  name: 数据
- key: model
  name: 预训练模型
outputs:
- key: save
  name: 模型
cmd: python3 train.py

infer:
resource:
limits:
cpu: 4
gpu: 1
memory: 10240
inputs:
- key: data
  name: 数据
- key: model
  name: 模型
outputs:
- key: save
  name: 推理输出
output: /output
cmd: python3 infer.py

test:
格式与以上一致

deploy:
格式与以上一致

compile:
格式与以上一致

```

environment.yml > version

environment.yml文件配置格式的版本号，以上格式配置的版本为1

environment.yml > image

image下的参数用于描述运行算法所需的Docker镜像，from_image含义等同于Dockerfile中的FROM，runs下的每一条配置等同于Dockerfile中的RUN。

environment.yml > train{test, infer, deploy, compile}

一般算法可以分为5个功能：训练，测试，批量推理，API服务，编译SDK。train, test, infer, deploy, compile分别与之对应，用于描述不同功能参数配置。5个功能的参数配置都是几乎相同的，以train为例进行说明。train下包含共包含4部分：运行所需资源描述resource，输入描述inputs，输出描述outputs，启动训练命令cmd。

parameter.yml

只要符合yaml文件的格式即可，字段与字段的值完全由用户自定义，在下述Helper包中会说明parameter.yml的使用。

Helper Package

kpl-helper包提供了用于在平台中运行算法的工具函数，如：获取数据的路径，传递和解析超参数等。安装方法：pip3 install kpl-helper 使用方法会在下面代码改写小节中以例子说明。

Dataset Package

在平台中，数据的序列化格式是由平台来定义的（类似TensorFlow的TFRecords，但是会更简单），该序列化方式既方便使用者脱离平台使用又能友好的在平台上由系统解析在Web上呈现内容。kpl-dataset就是用于创建和读取序列化数据集的工具包（可脱离平台使用）。安装方法：pip3 install kpl-dataset

写数据

```

from kpl_dataset import ReadOnlyDataset, WriteOnlyDataset, BasicType
# 定义要写入的字段结构和类型，结构支持key: value的层次化结构（推荐结构不要太深），key必须为字符串。
record_type = {
    "height": BasicType.Int,
    "width": BasicType.Int,
    "image": BasicType.ByteArray,
    "object": [{
        "name": BasicType.String,
        "xmin": BasicType.Int,
        "xmax": BasicType.Int,
        "ymin": BasicType.Int,
        "ymax": BasicType.Int
    }]
}
writer = WriteOnlyDataset("/tmp/", data_name="data_name", record_type=record_type)
for _ in range(10):
    writer.write({
        "height": 200,
        "width": 200,
        "image": b"image file",
        "object": [
            {
                "name": "dog",
                "xmin": 100,
                "ymin": 100,
                "xmax": 200,
                "ymax": 200,
            },
            {
                "name": "dog",
                "xmin": 100,
                "ymin": 100,
                "xmax": 200,
                "ymax": 200,
            }
        ]
    })
# 最后一定不要忘了关闭数据
writer.close()

```

读数据

```

from kpl_dataset import ReadOnlyDataset, WriteOnlyDataset, BasicType
reader = ReadOnlyDataset("/tmp", data_name=None)
# 迭代
record = next(reader)
print(record)

for record in reader:
    print(record)

```

```
# 以index索引
index = random.randint(0, len(reader))
record = reader[index]
print(record)

# 获取数据中record的数量
print(len(reader))

# 迭代结束后重置数据
reader.reset()

# 关闭数据
reader.close()
```

代码改写

如果你已经有一个可以在本地运行的算法，那么让该算法能在平台上运行，你需要对代码进行如下改写。

1. 编写environment.yml和parameter.yml文件，并存放于代码根路径下。
2. 在算法代码执行前后调用ready()和done()（必要）

```
import kpl_helper as helper
if __name__ == "__main__":
    helper.ready()
    # your algorithm code
    helper.done()
```

3. 获取超参数和使用超参数（必要）

```
'''
如parameter.yml文件内容如下:
KEY1:
KEY2: VALUE
'''

import kpl_helper as helper
parameter = helper.get_parameter(default="./parameter.yml")
print(parameter.KEY1.KEY2) # 输出结果为VALUE
```

4. 获取输入路径（必要）

你将使用到的数据和模型，会由系统挂载到运行时的容器中去，对应的路径需要使用helper包来获取。

```
'''
如environment.yml文件内容如下:
...
train:
  inputs:
    - key: data
      name: 输入数据
    - key: pretrain_model
      name: 预训练模型
...
'''

import kpl_helper as helper
data_path = helper.io.get_input_path(key="data", default="your local path")
pretrain_model_path = helper.io.get_input_path(key="model", default="your local path")
```

5. 获取输出路径（必要）

```
'''
如environment.yml文件内容如下:
...
train:
  outputs:
    - key: save_model
      name: 训练出的模型
...
'''

import kpl_helper as helper
save_path = helper.io.get_output_path(key="save_model", default="your local path")
```

6. 使用kpl-dataset读取输入数据（必要）

除了模型之外，所有的数据都需要使用kpl_dataset包进行读取。

```
from kpl_dataset import ReadOnlyDataset, WriteOnlyDataset, BasicType
import kpl_helper as helper
data_path = helper.io.get_input_path(key="data")
reader = ReadOnlyDataset(data_path, data_name=None)
# method 1. iteration
for record in reader:
    print(record)

# method 2
record = next(reader)
print(record)

# method 3. read by index
index = random.randint(0, len(reader))
record = reader[index]
print(record)
```

7. 使用kpl-dataset保存输出数据（必要）

在批量推理保存数据时，数据的保存格式一定要使用kpl_dataset包进行序列化保存。

```
from kpl_dataset import ReadOnlyDataset, WriteOnlyDataset, BasicType
from kpl_dataset.common import RectangleBoxObjectDetectionDatasetDefine
import kpl_helper as helper
save_path = helper.io.get_output_path(key="save_model")
# 以一般矩形框检测数据的组织方式为例
writer = WriteOnlyDataset(save_path, data_name="data_name",
                          record_type=RectangleBoxObjectDetectionDatasetDefine)

for _ in range(10):
    writer.write({
        "path": "file path (String)",
        "content": b"I am image file (Bytes)",
        "contentType": "image/jpeg",
        "object": [
            {
                "name": "dog",
                "xmin": 100,
                "ymin": 100,
                "xmax": 200,
                "ymax": 200,
                "difficulty": 0,
                "score": 0.9
            }
        ]
    })
writer.close()
```

8. 发送Metric数据（可选）

如果希望在Web端可视化Loss等曲线，可以通过helper包提供的MetricFigure来实现，如：

```
from kpl_helper import metric
figure = metric.MetricFigure("title", y1_name="loss") # 默认横坐标为迭代次数
iter_num = 100
```

```
for no in range(1, iter_num):
    ce_loss = 10.0 / no
    figure.push_metric(x=no, y1=ce_loss)
```

9. 发送进度信息（可选）

如果希望系统能对剩余时长进行更好的估计，可以将内部算法当前进行进度通过send_progress函数发出来，如

```
from kpl_helper.progress import send_progress
iter_num = 100
for no in range(1, iter_num):
    send_progress(float(no) / iter_num)
```

QA:

语义分割算法

上传数据

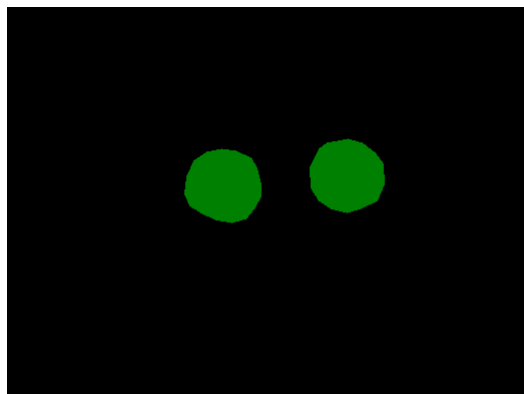
以下处理过程中，基于如下组织的原始数据（非必须）

```
.
├── JPEGImages
│   ├── IMG_20180129_163826.jpg
│   ├── IMG_20180129_163839.jpg
│   ├── IMG_20180129_163843.jpg
│   └── IMG_20180129_163849.jpg
└── SegmentationClass
    ├── IMG_20180129_163826.png
    ├── IMG_20180129_163839.png
    ├── IMG_20180129_163843.png
    └── IMG_20180129_163849.png
```

即以PascalVOC开源数据的方式进行组织。JPEGImages文件夹中全为图片，SegmentationClass文件夹中全为Mask语义分割标注图片。关系在于图片的文件名称是一一对应的（图片如果是IMG_20180129_163826.jpg，那么标注文件的名称是IMG_20180129_163826.png）。如： 图片IMG_20180129_163826.jpg:



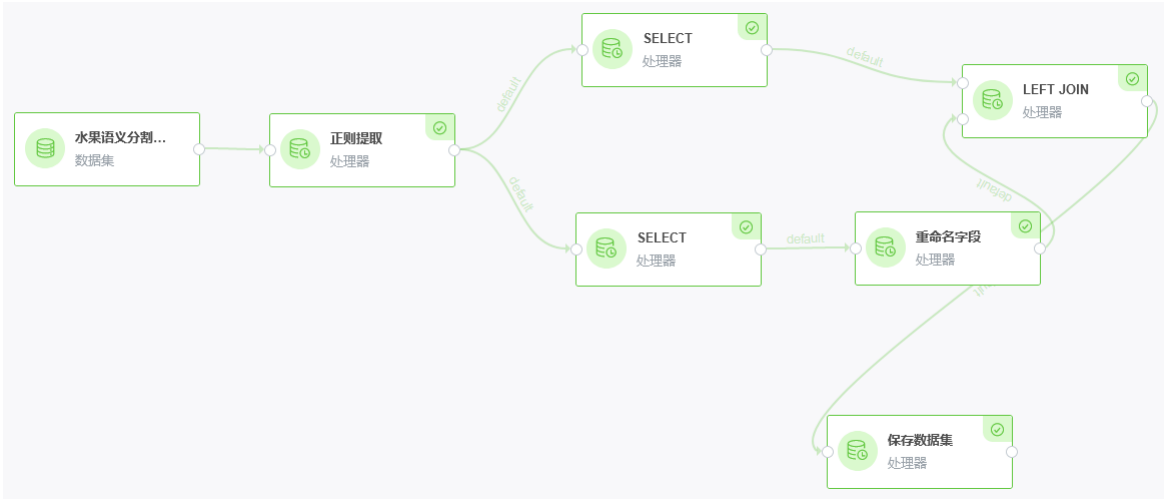
标注图片IMG_20180129_163826.png:



最后使用zip压缩后上传即可（注意：根目录不能有中文，只能是大小写字母，数字和中划线下划线）

处理数据

构建处理上述语义分割数据的Pipeline



1. 节点：水果语义分割数据集 数据节点。该数据即为上述上传的数据。 数据为：

path(String)	contentType(String)	content(ByteArray)
Fruit/SegmentationClass/IMG_20180129_182401.png	image/png	
Fruit/SegmentationClass/IMG_20180129_182722.png	image/png	

2. 节点：正则提取 对数据进行处理，将数据的path路径进行正则提取，创建两个新字段：type和id 例：如数据图片path为root/JPEGImages/IMG_20180129_163826.jpg，解析完成后，type=JPEGImages，id=IMG_20180129_163826。如此解析的目的为使用type区分是数据图片还是mask图片，使用id将数据图片和mask图片进行JOIN一一对应起来。 节点参数

正则提取

[高级 >](#)

基于正则选取功能，从已有String属性中提取新属性，正则匹配失败时丢弃Record

来源属性

正则表达式

输出数据

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)
Fruit/SegmentationClass/IMG_20180129_182...	image/png		SegmentationClass	IMG_20180129_182401
Fruit/SegmentationClass/IMG_20180129_182...	image/png		SegmentationClass	IMG_20180129_182722

3. 节点：SELECT 对正则解析后的数据进行过滤操作，将数据图片和mask图片分开。 节点参数 选取数据图片的SELECT

SELECT

[高级 >](#)

SELECT

来源属性

反向

运算符

目标字符串

选取Mask图片的SELECT

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)
Fruit/JPEGImages/IMG_20180129_172813.jpg	image/jpeg		JPEGImages	IMG_20180129_172813
Fruit/JPEGImages/IMG_20180129_173315.jpg	image/jpeg		JPEGImages	IMG_20180129_173315

输出数据 选取数据图片的SELECT

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)
Fruit/JPEGImages/IMG_20180129_172813.jpg	image/jpeg		JPEGImages	IMG_20180129_172813
Fruit/JPEGImages/IMG_20180129_173315.jpg	image/jpeg		JPEGImages	IMG_20180129_173315

选取Mask图片的SELECT

重命名字段



高级 >

将选定的字段重命名为新字段名，原字段名将会被丢弃

原有字段名称

新的字段名称

4. 节点: 重命名字段 为了避免在JOIN过程中，有两个相同的content字段冲突，先对Mask数据所在的字段进行重命名。 节点参数

path(String)	contentType(String)	mask(ByteArray)	type(String)	id(String)
Fruit/SegmentationClass/IMG_20180129_182...	image/png		SegmentationClass	IMG_20180129_182401
Fruit/SegmentationClass/IMG_20180129_182...	image/png		SegmentationClass	IMG_20180129_182722

输出数据

LEFT JOIN







高级 >

LEFT JOIN

属性名1

属性名2

5. 节点: LEFT JOIN 根据第一步正则提取出来的id字段，将数据图片和Mask图片进行一一对应，两者构成一个完整的训练样本。 节点参数

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)	mask(ByteArray)
Fruit/JPEGImages/IMG...	image/jpeg		JPEGImages	IMG_20180129_172813	
Fruit/JPEGImages/IMG...	image/jpeg		JPEGImages	IMG_20180129_173315	
Fruit/JPEGImages/IMG...	image/jpeg		JPEGImages	IMG_20180129_170432	

输出数据

! [检测15.png] (<http://fe-frame.ks3-cn-beijing.ksyun.com/project/cms/968f88ec3483eb606d2183790d068bbe>)

6. 节点: 保存数据集 将最后得到的输出持久化为数据集，便于在后续流程中使用。

训练等

和分类算法的使用过程类似。 构建一个Pipeline，使用上述处理好的数据和预训练模型作为训练的输入，将训练输出的模型进行保存，同时也部署一个API对结果进行可视化查看。



1. 通过API查看效果 在API部署中找到刚刚部署的API。上传测试数据。



Response

```
{
  "code": "Success",
  "data": {
    "depth": 3,
    "height": 389,
    "objects": [
      {
        "name": "apple",
        "score": 0.99917834997
      }
    ]
  },
  "xmax": 254,
  "xmin": 179,
  17712,
}
```

分类识别算法

上传数据

```

├── apple
│   ├── 0.jpg
│   ├── 100.jpg
│   ├── 101.jpg
│   └── 102.jpg
├── dragon
│   ├── 97.jpg
│   ├── 98.jpg
│   ├── 99.jpg
│   └── 9.jpg
├── kiwi
│   ├── 97.jpg
│   ├── 98.jpg
│   ├── 99.jpg
│   └── 9.jpg
└── orange
    ├── 0.jpg
    ├── 100.jpg
    ├── 101.jpg
    └── 102.jpg

```

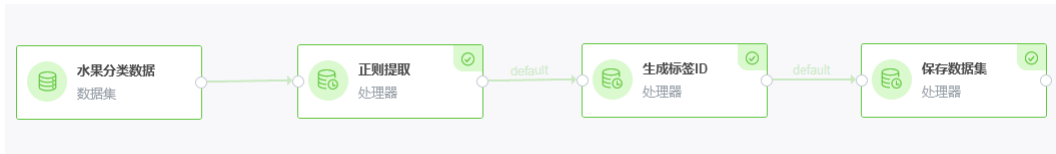
以下处理过程中，基于如下组织的原始数据（非必须）

每一类的图片以不同的文件夹存放。 图片示例：



最后使用zip压缩后上传即可（注意：根目录不能有中文，只能是大小写字母，数字和中划线下划线）

处理数据



构建处理上述数据的Pipeline

- 节点：水果分类数据集 数据节点。该数据即为上述上传的数据。 数据为：

path(String)	contentType(String)	content(ByteArray)
Fruit/kiwi/265.jpg	image/jpeg	
Fruit/kiwi/120.jpg	image/jpeg	

- 节点：正则提取 对数据进行处理，将数据的path路径进行正则提取，创建两个新字段：type和id 例：如数据图片path为root/JPEGImages/IMG_20180129_163826.jpg，解析完成后，type=JPEGImages, id=IMG_20180129_163826。如此解析的目的为使用type区分是数据图片还是mask图片，使用id将数据图片和mask图片进行JOIN一一对应起来。 节点参数：

正则提取 高级 >

基于正则选取功能，从已有String属性中提取新属性，正则匹配失败时丢弃Record

来源属性:

正则表达式:

输出数据:

path(String)	contentType(String)	content(ByteArray)	class_name(String)	file_name(String)
Fruit/kiwi/74.jpg	image/jpeg		kiwi	74
Fruit/kiwi/360.jpg	image/jpeg		kiwi	360

生成标签ID 高级 >

生成标签ID

标签属性:

输出属性名:

- 节点：生成标签ID 根据正则提取出来的文件夹名称（也就是类别名称）生成ID（从0开始生成） 节点参数：

输出数据:

path(String)	contentType(String)	content(ByteArray)	class_name(String)	file_name(String)	class_id(Int)
Fruit/apple/256.jpg	image/jpeg		apple	256	0
Fruit/kiwi/370.jpg	image/jpeg		kiwi	370	2

4. 节点：保存数据集 将最后得到的输出持久化为数据集，便于在后续流程中使用。

训练等



可以构建如下Pipeline，使用上述处理完保存的数据集作为输入

1. 节点：算法训练 点击训练节点，可设置算法训练参数，如迭代次数，GPU数量等。参数种类和命名均与具体算法开发者提供的算法相关，以下供参考。

Gluon 分类算法

高级 >

Gluon 分类算法

环境参数 超参数

编辑

```

version : 1
image
  from_image : hub.kce.ksyun.com/kpl_k8s/mxn
  et.cuda10.1-py2-py3-v1.5.1
  copy_first : false
  runs
    [0] : pip3 install numpy opencv-python
    -i https://pypi.tuna.tsinghua.edu.cn
    /simple/
    [1] : pip3 install kpl-helper-test==0.0.5
    kpl-dataset-test==0.0.5
  train
  resource
  
```

分类算法重点review参数：

- 类别数量（与选择的数据集有关）

环境参数 超参数

编辑

```

...
last_gamma : false
num_epochs : 20
hard_weight : 0.5
mixup_alpha : 0.2
num_classes : 4
num_workers : 4

```

- 迭代次数或epoch大小

2. 节点：算法API部署 类似训练

3. 节点：保存模型 为了下次再次使用该训练输出的模型进行测试或其他，最好将模型进行保存，以便进行管理。

4. 通过API查看效果 在API部署中找到刚刚部署的API。上传测试数据。

API测试

The screenshot shows an API testing tool interface. On the left, there is a search bar and a button to upload a local image. In the center, a red apple is displayed. On the right, the JSON response is shown:

```

{
  "code": "Success",
  "data": {
    "class_id": 0,
    "class_name": "apple",
    "depth": 3,
    "height": 256,
    "type": "classification",
    "width": 256
  },
  "msg": "",
  "text": ""
}
    
```

检测算法

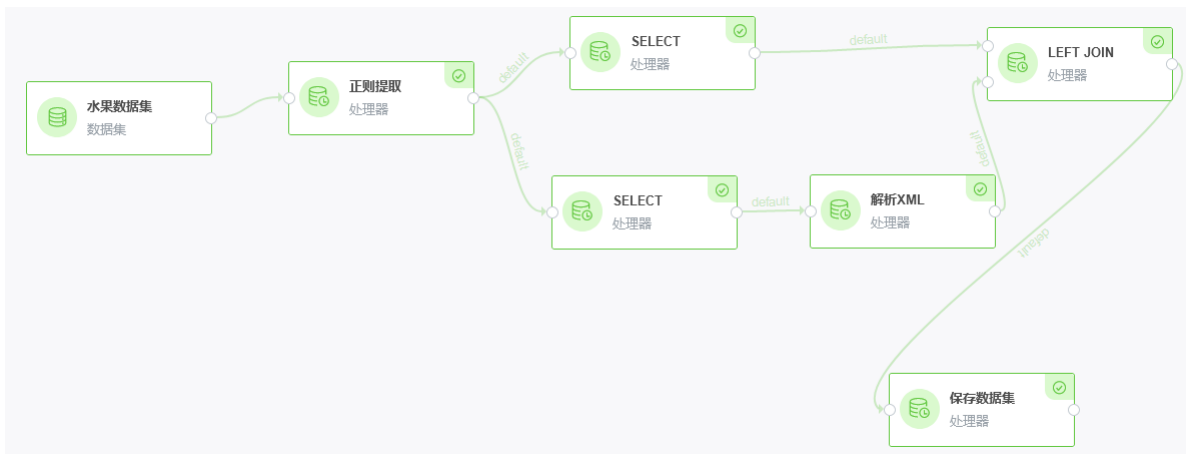
上传数据

```

Annotations
├── IMG_20180129_163416.xml
├── IMG_20180129_163419.xml
├── IMG_20180129_163422.xml
├── IMG_20180129_163424.xml
├── IMG_20180129_190246.xml
└── JPEGImages
    ├── IMG_20180129_163416.jpg
    ├── IMG_20180129_163419.jpg
    ├── IMG_20180129_163422.jpg
    ├── IMG_20180129_163424.jpg
    └── IMG_20180129_190246.jpg
    
```

以下处理过程中，基于如下组织的原始数据（非必须）中全为图片，Annotations文件夹中全为XML标注文件。关系在于图片和XML文件名称是一一对应的（图片如果是IMG_20180129_163826.jpg，那么对应的标注文件的名称是IMG_20180129_163826.xml）。最后使用zip压缩后上传即可（注意：根目录不能有中文，只能是大小写字母，数字和中划线下划线）

处理数据



构建处理上述数据的Pipeline

1. 节点：水果检测数据集 数据节点。该数据即为上述上传的数据。 数据为：

path(String)	contentType(String)	content(ByteArray)
fruit/JPEGImages/IMG_20180129_172432.jpg	image/jpeg	
fruit/JPEGImages/IMG_20180129_190159.jpg	image/jpeg	

2. 节点：正则提取 对数据进行处理，将数据的path路径进行正则提取，创建两个新字段：type和id 例：如数据图片path为root/JPEGImages/IMG_20180129_163826.jpg，解析完成后，type=JPEGImages, id=IMG_20180129_163826。如此解析的目的为使用type区分是数据图片还是标注XML，最后使用id将数据图片和标注XML进行JOIN一一对应起来。 节点参数：

正则提取

[高级 >](#)

基于正则选取功能，从已有String属性中提取新属性，正则匹配失败时丢弃Record

来源属性

path

正则表达式

^.*\?(?P<type>.*)(?P<id>.*).*?\$

输出数据：

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)
fruit/Annotations/IMG_20180129_170422.xml	application/xml	<annotation> <folder>baseball</folder> <filena...	Annotations	IMG_20180129_170422
fruit/Annotations/IMG_20180129_183559.xml	application/xml	<annotation> <folder>baseball</folder> <filena...	Annotations	IMG_20180129_183559

3. 节点: SELECT 对正则解析后的数据进行过滤操作, 将数据图片和标注XML分开。 节点参数: 选取数据图片的SELECT

SELECT 高级 >

SELECT

来源属性: type

反向: false

运算符: Equal

目标字符串: JPEGImages

选取标注XML的

SELECT 高级 >

SELECT



来源属性: type

反向: false

运算符: Equal

目标字符串: Annotations

输出数据: 选取数据图片的SELECT

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)
Fruit/JPEGImages/IMG_20180129_172813.jpg	image/jpeg		JPEGImages	IMG_20180129_172813
Fruit/JPEGImages/IMG_20180129_173315.jpg	image/jpeg		JPEGImages	IMG_20180129_173315

选取标注XML的SELECT

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)
fruit/Annotations/IMG_20180129_170422.xml	application/xml	<annotation> <folder>baseball</folder> <filena...	Annotations	IMG_20180129_170422
fruit/Annotations/IMG_20180129_183559.xml	application/xml	<annotation> <folder>baseball</folder> <filena...	Annotations	IMG_20180129_183559

解析XML 高级 >

从XML中选取有用的字段

来源属性: content

字段提取

item 1 X item

属性类型: List

XML路径: annotation/object[*]

目标路径: object

子元素

item 1 X item

属性类型: String

XML路径: name

目标路径: name

item 2 X item

属性类型: Int

XML路径: difficult

目标路径: difficult

item 3 X item

属性类型: Int

XML路径:

4. 节点: 解析XML 解析XML中记录的Bounding Box信息。 节点参数

输出数据

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)	object(List)
fruit/Annotations/IMG_20180129_182...	application/xml	<annotation> <folder>baseball</folder>...	Annotations	IMG_20180129_182731	[{"difficult":0,"name":"drago
fruit/Annotations/IMG_20180129_173...	application/xml	<annotation> <folder>baseball</folder>...	Annotations	IMG_20180129_173427	[{"difficult":0,"name":"apple
fruit/Annotations/IMG_20180129_170...	application/xml	<annotation> <folder>baseball</folder>...	Annotations	IMG_20180129_170422	[{"difficult":0,"name":"orang

5. 节点: LEFT JOIN 根据第一步正则提取出来的id字段, 将数据图片和解析出来的object一一对应, 两者构成一个完整的训练样本。 节点参数:

LEFT JOIN 高级 >

LEFT JOIN

属性名1

属性名2

输出数据:

path(String)	contentType(String)	content(ByteArray)	type(String)	id(String)	object(List)
fruit/JPEGImages/IMG_20180129_19...	image/jpeg		JPEGImages	IMG_20180129_190054	[{"difficult":0,"name"
fruit/JPEGImages/IMG_20180129_16...	image/jpeg		JPEGImages	IMG_20180129_163856	[{"difficult":0,"name"
fruit/JPEGImages/IMG_20180129_17...	image/jpeg		JPEGImages	IMG_20180129_172432	[{"difficult":0,"name"

6. 节点: 保存数据集 将最后得到的输出持久化为数据集, 便于在后续流程中使用。

训练等



可以构建如下Pipeline, 使用上述处理完保存的数据集作为输入

1. 节点: 算法训练 点击训练节点, 可设置算法训练参数, 如迭代次数, GPU数量等。参数种类和命名均与具体算法开发者提供的算法相关, 以下供参考。

Gluon SSD检测算法 [高级 >](#)

Gluon SSD检测算法

环境参数 [超参数](#)

```
lr : 0.001
wd : 0.0005
gpus : 0
seed : 233
epochs : 240
resume :
syncbn : false
classes
  [0] : orange
  [1] : apple
  [2] : dragon
  [3] : kiwi
```

分类算法重点review参数:

- 类别名称列表 (与选择的数据集有关), 在训练、测试、API部署等中, 注意不要更改

- 迭代次数或epoch大小

2. 节点: 算法API部署 类似训练
3. 节点: 保存模型 为了下次再次使用该训练输出的模型进行测试或其他, 最好将模型进行保存, 以便进行管理。
4. 通过API查看效果 在API部署中找到刚刚部署的API。上传测试数据。

只能上传 .jpg 和 .png 格式的图片, 文件大小不超过5MB

```
Response
{
  "code": "Success",
  "data": {
    "depth": 3,
    "height": 389,
    "objects": [
      {
        "name": "apple",
        "score": 0.99917834997
      }
    ],
    "width": 17712,
    "xmax": 254,
    "xmin": 179,
  }
}
```