# MNIST数字识别-在线部署

## 教程说明

本教程主要是使用AI Studio部署数字识别模型，模型的训练使用了[PaddlePaddle的数字识别教程](http://paddlepaddle.org/documentation/docs/zh/1.2/beginners_guide/quick_start/recognize_digits/README.cn.html)，AI Studio的使用参考了[AI Studio的教程——项目大厅-在线部署及预测](http://ai.baidu.com/docs#/AIStudio_Project_Forecast/top)，AI Studio Notebook的使用参考教程[项目大厅-Notebook使用说明](http://ai.baidu.com/docs#/AIStudio_Project_Notebook/top)

## 创建项目

打开[AI Studio](https://aistudio.baidu.com/aistudio/#/projectoverview)并**登陆**（不支持IE浏览器）

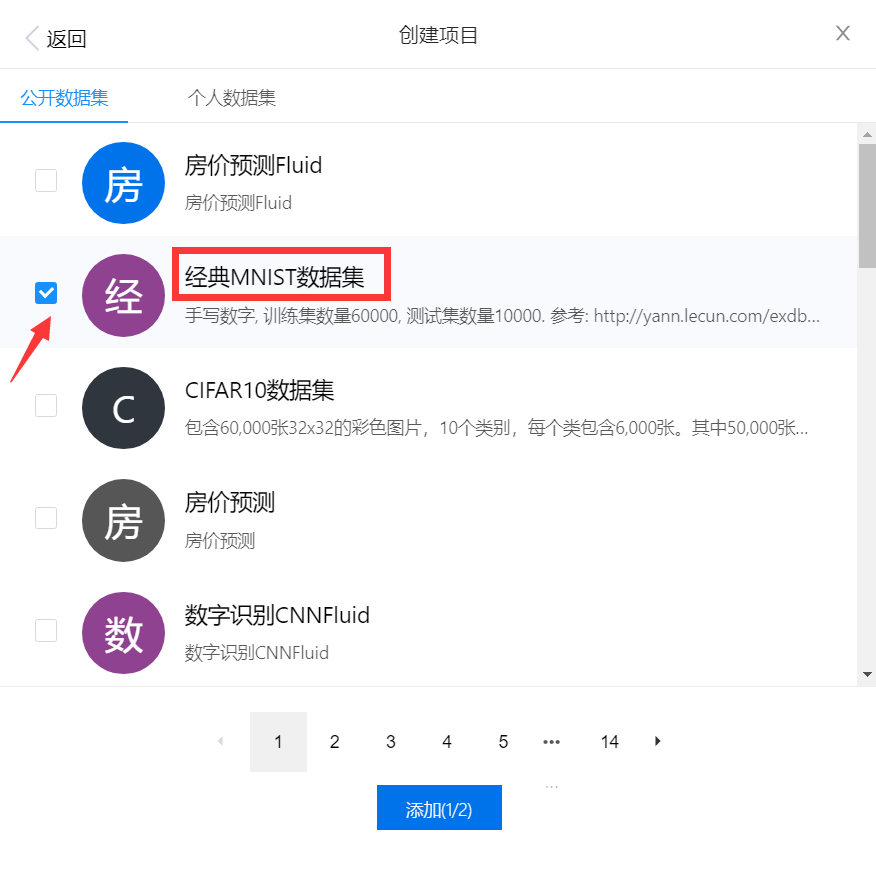


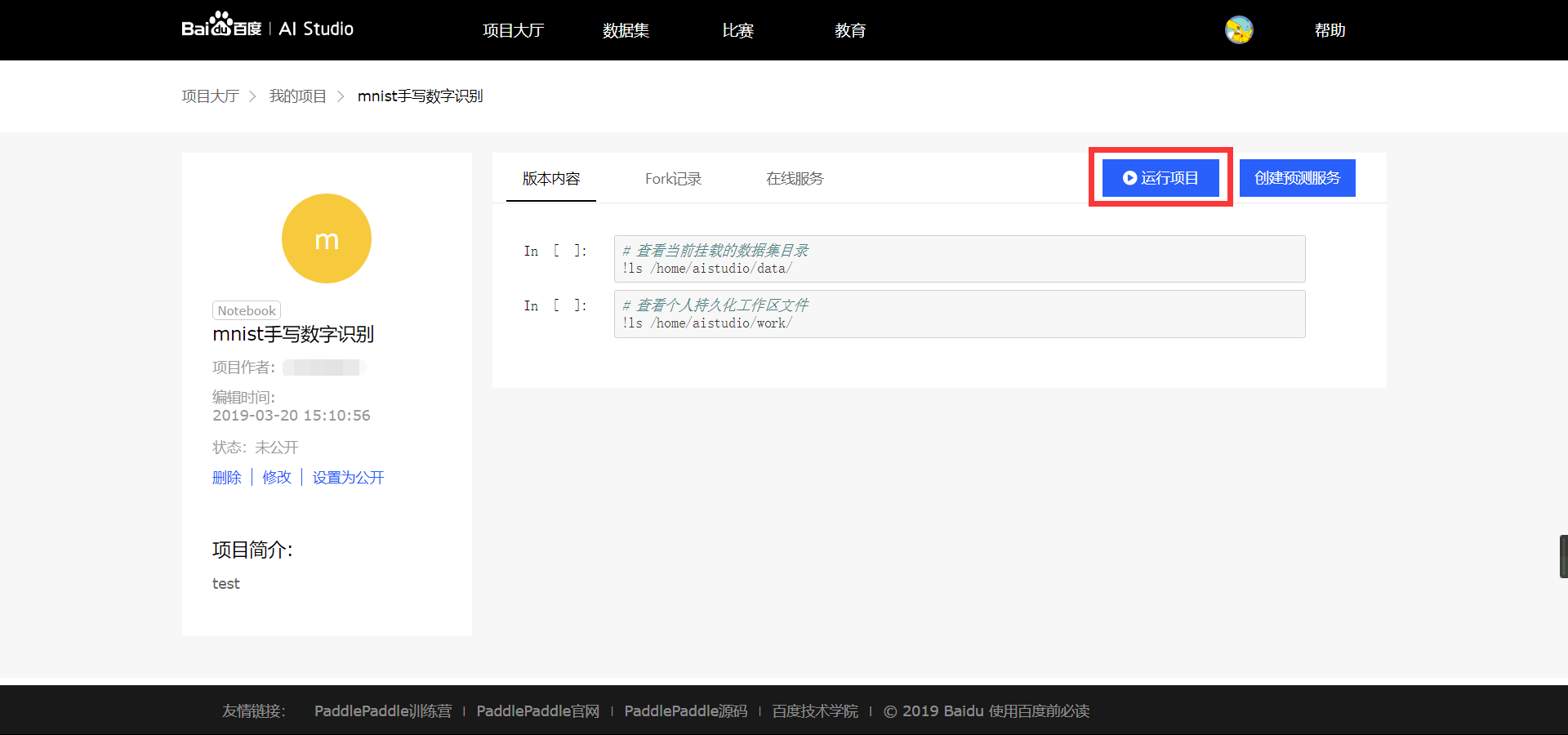
然后点击【项目】、【创建项目】



输入【项目名称】和【项目描述】，然后点击【添加数据集】  


找到【经典MNIST数据集】并勾选，然后点击【添加】，最后点击【创建】



创建完成后，点击【运行项目】  


## 通过训练任务生成模型文件

通过【Cell操作】下的【新增cell】创建新的cell，并将代码拷贝到其中



使用[PaddlePaddle数字识别教程](http://paddlepaddle.org/documentation/docs/zh/1.2/beginners_guide/quick_start/recognize_digits/README.cn.html)中的代码进行训练，在训练任务过程中, 通过调用[paddle.fluid.io.save\_inference\_model](http://paddlepaddle.org/documentation/docs/zh/1.2/api_cn/io_cn.html#permalink-5-save_inference_model)实现模型的保存，具体代码如下

**import** os

**from** PIL **import** Image # 导入图像处理模块

**import** matplotlib.pyplot **as** plt

**import** numpy

**import** paddle # 导入paddle模块

**import** paddle.fluid **as** fluid

**from** \_\_future\_\_ **import** print\_function # 将python3中的print特性导入当前版本

**def** **softmax\_regression**():

"""

定义softmax分类器：

一个以softmax为激活函数的全连接层

Return:

predict\_image -- 分类的结果

"""

# 输入的原始图像数据，大小为28\*28\*1

img = fluid.layers.data(name='img', shape=[1, 28, 28], dtype='float32')

# 以softmax为激活函数的全连接层，输出层的大小必须为数字的个数10

predict = fluid.layers.fc(

input=img, size=10, act='softmax')

**return** predict

**def** **multilayer\_perceptron**():

"""

定义多层感知机分类器：

含有两个隐藏层（全连接层）的多层感知器

其中前两个隐藏层的激活函数采用 ReLU，输出层的激活函数用 Softmax

Return:

predict\_image -- 分类的结果

"""

# 输入的原始图像数据，大小为28\*28\*1

img = fluid.layers.data(name='img', shape=[1, 28, 28], dtype='float32')

# 第一个全连接层，激活函数为ReLU

hidden = fluid.layers.fc(input=img, size=200, act='relu')

# 第二个全连接层，激活函数为ReLU

hidden = fluid.layers.fc(input=hidden, size=200, act='relu')

# 以softmax为激活函数的全连接输出层，输出层的大小必须为数字的个数10

prediction = fluid.layers.fc(input=hidden, size=10, act='softmax')

**return** prediction

**def** **convolutional\_neural\_network**():

"""

定义卷积神经网络分类器：

输入的二维图像，经过两个卷积-池化层，使用以softmax为激活函数的全连接层作为输出层

Return:

predict -- 分类的结果

"""

# 输入的原始图像数据，大小为28\*28\*1

img = fluid.layers.data(name='img', shape=[1, 28, 28], dtype='float32')

# 第一个卷积-池化层

# 使用20个5\*5的滤波器，池化大小为2，池化步长为2，激活函数为Relu

conv\_pool\_1 = fluid.nets.simple\_img\_conv\_pool(

input=img,

filter\_size=5,

num\_filters=20,

pool\_size=2,

pool\_stride=2,

act="relu")

conv\_pool\_1 = fluid.layers.batch\_norm(conv\_pool\_1)

# 第二个卷积-池化层

# 使用50个5\*5的滤波器，池化大小为2，池化步长为2，激活函数为Relu

conv\_pool\_2 = fluid.nets.simple\_img\_conv\_pool(

input=conv\_pool\_1,

filter\_size=5,

num\_filters=50,

pool\_size=2,

pool\_stride=2,

act="relu")

# 以softmax为激活函数的全连接输出层，输出层的大小必须为数字的个数10

prediction = fluid.layers.fc(input=conv\_pool\_2, size=10, act='softmax')

**return** prediction

**def** **train\_program**():

"""

配置train\_program

Return:

predict -- 分类的结果

avg\_cost -- 平均损失

acc -- 分类的准确率

"""

# 标签层，名称为label,对应输入图片的类别标签

label = fluid.layers.data(name='label', shape=[1], dtype='int64')

# predict = softmax\_regression() # 取消注释将使用 Softmax回归

# predict = multilayer\_perceptron() # 取消注释将使用 多层感知器

predict = convolutional\_neural\_network() # 取消注释将使用 LeNet5卷积神经网络

# 使用类交叉熵函数计算predict和label之间的损失函数

cost = fluid.layers.cross\_entropy(input=predict, label=label)

# 计算平均损失

avg\_cost = fluid.layers.mean(cost)

# 计算分类准确率

acc = fluid.layers.accuracy(input=predict, label=label)

**return** predict, [avg\_cost, acc]

**def** **reader\_creator**(image\_filename,label\_filename,buffer\_size):

**def** **reader**():

#调用命令读取文件，Linux下使用zcat

# zcat\_cmd = 'zcat'

**if** platform.system()=='Linux':

zcat\_cmd = 'zcat'

**elif** paltform.system()=='Windows':

zcat\_cmd = 'gzcat'

**else**:

**raise** NotImplementedError("This program is suported on Windows or Linux,\

but your platform is" + platform.system())

#create a subprocess to read the images

sub\_img = subprocess.Popen([zcat\_cmd, image\_filename], stdout = subprocess.PIPE)

sub\_img.stdout.read(16) #skip some magic bytes 这里我们已经知道，所以我们不在需要前16字节

#create a subprocess to read the labels

sub\_lab = subprocess.Popen([zcat\_cmd, label\_filename], stdout = subprocess.PIPE)

sub\_lab.stdout.read(8) #skip some magic bytes 同理

**try**:

**while** **True**: #前面使用try,故若再读取过程中遇到结束则会退出

#label is a pixel repersented by a unsigned byte,so just read a byte

labels = numpy.fromfile(

sub\_lab.stdout,'ubyte',count=buffer\_size).astype("int")

**if** labels.size != buffer\_size:

**break**

#read 28\*28 byte as array,and then resize it

images = numpy.fromfile(

sub\_img.stdout,'ubyte',

count=buffer\_size \* 28 \* 28).reshape(buffer\_size, 28, 28).astype("float32")

#mapping each pixel into (-1,1)

images = images / 255.0 \* 2.0 - 1.0;

**for** i **in** xrange(buffer\_size):

**yield** images[i,:],int(labels[i]) #将图像与标签抛出，循序与feed\_order对应！

**finally**:

**try**:

#terminate the reader subprocess

sub\_img.terminate()

**except**:

**pass**

**try**:

#terminate the reader subprocess

sub\_lable.terminate()

**except**:

**pass**

**return** reader

**def** **optimizer\_program**():

**return** fluid.optimizer.Adam(learning\_rate=0.001)

# 一个minibatch中有64个数据

BATCH\_SIZE = 64

# 每次读取训练集中的500个数据并随机打乱，传入batched reader中，batched reader 每次 yield 64个数据

train\_reader = paddle.batch(

paddle.reader.shuffle(

paddle.dataset.mnist.train(), buf\_size=500),

batch\_size=BATCH\_SIZE)

# 读取测试集的数据，每次 yield 64个数据

test\_reader = paddle.batch(

paddle.dataset.mnist.test(), batch\_size=BATCH\_SIZE)

**def** **event\_handler**(pass\_id, batch\_id, cost):

# 打印训练的中间结果，训练轮次，batch数，损失函数

print("Pass %d, Batch %d, Cost %f" % (pass\_id,batch\_id, cost))

**from** paddle.utils.plot **import** Ploter

train\_prompt = "Train cost"

test\_prompt = "Test cost"

cost\_ploter = Ploter(train\_prompt, test\_prompt)

# 将训练过程绘图表示

**def** **event\_handler\_plot**(ploter\_title, step, cost):

cost\_ploter.append(ploter\_title, step, cost)

cost\_ploter.plot()

# 该模型运行在单个CPU上

use\_cuda = **False**

place = fluid.CUDAPlace(0) **if** use\_cuda **else** fluid.CPUPlace()

# 调用train\_program 获取预测值，损失值，

prediction, [avg\_loss, acc] = train\_program()

# 输入的原始图像数据，大小为28\*28\*1

img = fluid.layers.data(name='img', shape=[1, 28, 28], dtype='float32')

# 标签层，名称为label,对应输入图片的类别标签

label = fluid.layers.data(name='label', shape=[1], dtype='int64')

# 告知网络传入的数据分为两部分，第一部分是img值，第二部分是label值

feeder = fluid.DataFeeder(feed\_list=[img, label], place=place)

# 选择Adam优化器

optimizer = fluid.optimizer.Adam(learning\_rate=0.001)

optimizer.minimize(avg\_loss)

PASS\_NUM = 5 #训练5轮

epochs = [epoch\_id **for** epoch\_id **in** range(PASS\_NUM)]

# 将模型参数存储在名为 save\_dirname 的文件中

save\_dirname = "recognize\_digits.inference.model"

def train\_**test**(train\_**test**\_program,

train\_**test**\_feed, train\_**test**\_reader):

# 将分类准确率存储在acc\_set中

acc\_**set** = []

# 将平均损失存储在avg\_loss\_set中

avg\_loss\_**set** = []

# 将测试 reader yield 出的每一个数据传入网络中进行训练

**for** **test**\_data **in** train\_**test**\_reader():

acc\_np, avg\_loss\_np = exe.run(

program=train\_**test**\_program,

feed=train\_**test**\_feed.feed(**test**\_data),

fetch\_list=[acc, avg\_loss])

acc\_set.append(**float**(acc\_np))

avg\_loss\_set.append(**float**(avg\_loss\_np))

# 获得测试数据上的准确率和损失值

acc\_val\_mean = numpy.array(acc\_**set**).mean()

avg\_loss\_val\_mean = numpy.array(avg\_loss\_**set**).mean()

# 返回平均损失值，平均准确率

**return** avg\_loss\_val\_mean, acc\_val\_mean

exe = fluid.Executor(place)

exe.run(fluid.default\_startup\_program())

main\_program = fluid.default\_main\_program()

**test**\_program = fluid.default\_main\_program().clone(**for**\_**test**=True)

lists = []

step = 0

**for** epoch\_id **in** epochs:

**for** step\_id, data **in** enumerate(train\_reader()):

metrics = exe.run(main\_program,

feed=feeder.feed(data),

fetch\_list=[avg\_loss, acc])

**if** step % 100 == 0: #每训练100次 打印一次log

**print**("Pass %d, Batch %d, Cost %f" % (step, epoch\_id, metrics[0]))

event\_handler\_plot(train\_prompt, step, metrics[0])

step += 1

# 测试每个epoch的分类效果

avg\_loss\_val, acc\_val = train\_**test**(train\_**test**\_program=**test**\_program,

train\_**test**\_reader=**test**\_reader,

train\_**test**\_feed=feeder)

**print**("Test with Epoch %d, avg\_cost: %s, acc: %s" %(epoch\_id, avg\_loss\_val, acc\_val))

event\_handler\_plot(**test**\_prompt, step, metrics[0])

lists.append((epoch\_id, avg\_loss\_val, acc\_val))

# 保存训练好的模型参数用于预测

**if** save\_dirname is not None:

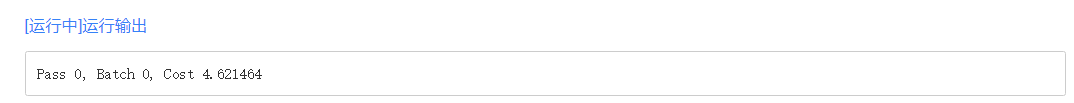
fluid.io.save\_inference\_model(save\_dirname,

["img"], [prediction], exe,

model\_filename=None,

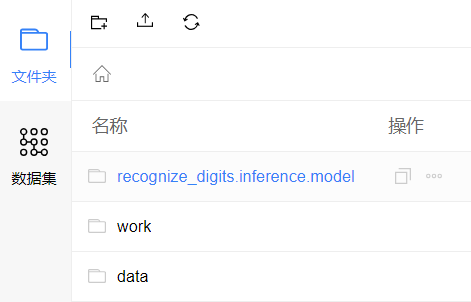
params\_filename=None)

点击【Notebook操作】，然后点击【运行所有】，程序便开始运行，大约需要十几分钟（5000pass），请耐心等待  

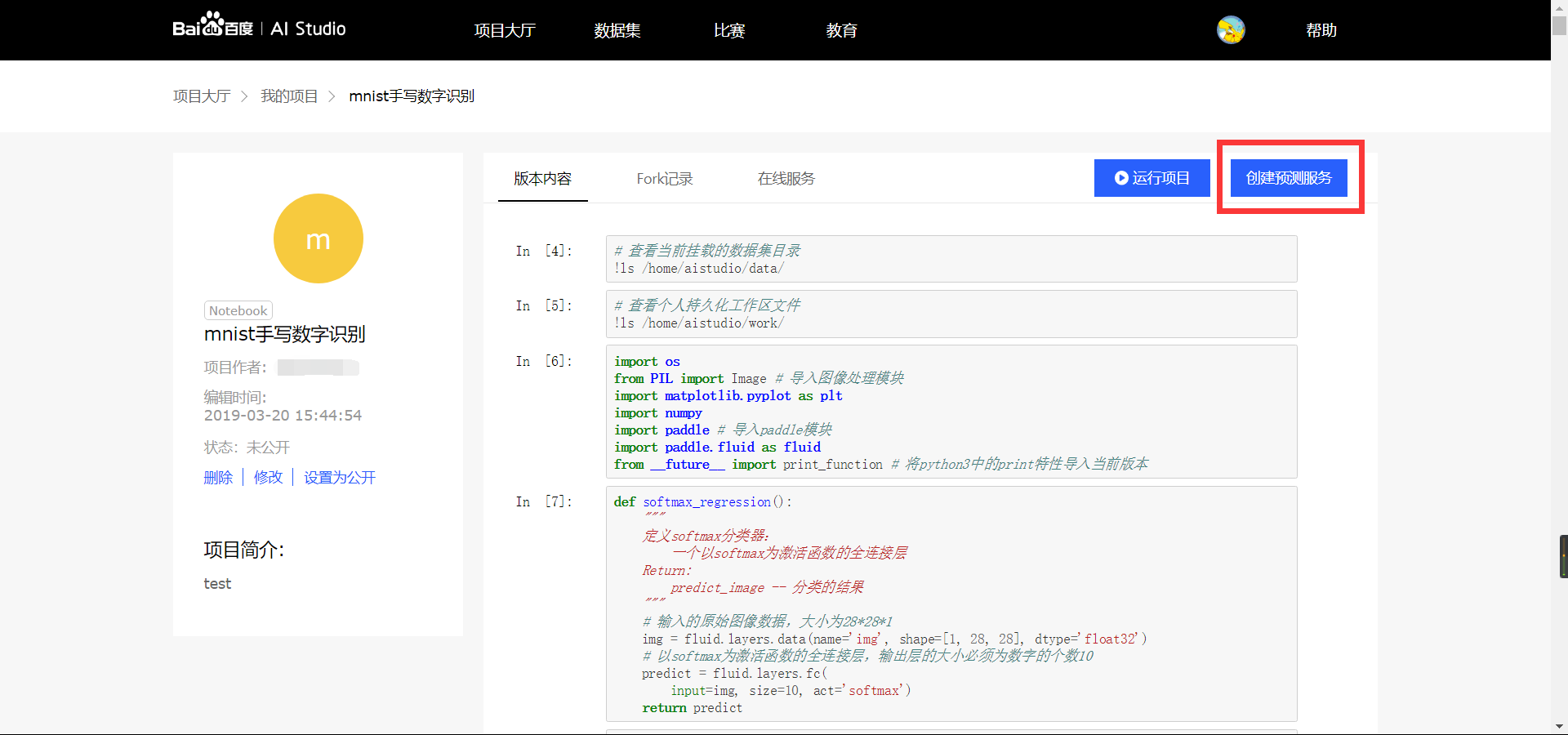

最后一个训练的cell打印信息时会卡住  
  
（此步可忽略：点击【清空输出】后将会打印新的信息，但仍会卡住，可以不管它）

  
等到最后一个cell右下角出现绿色的【运行耗时：xx】时，说明训练结束  

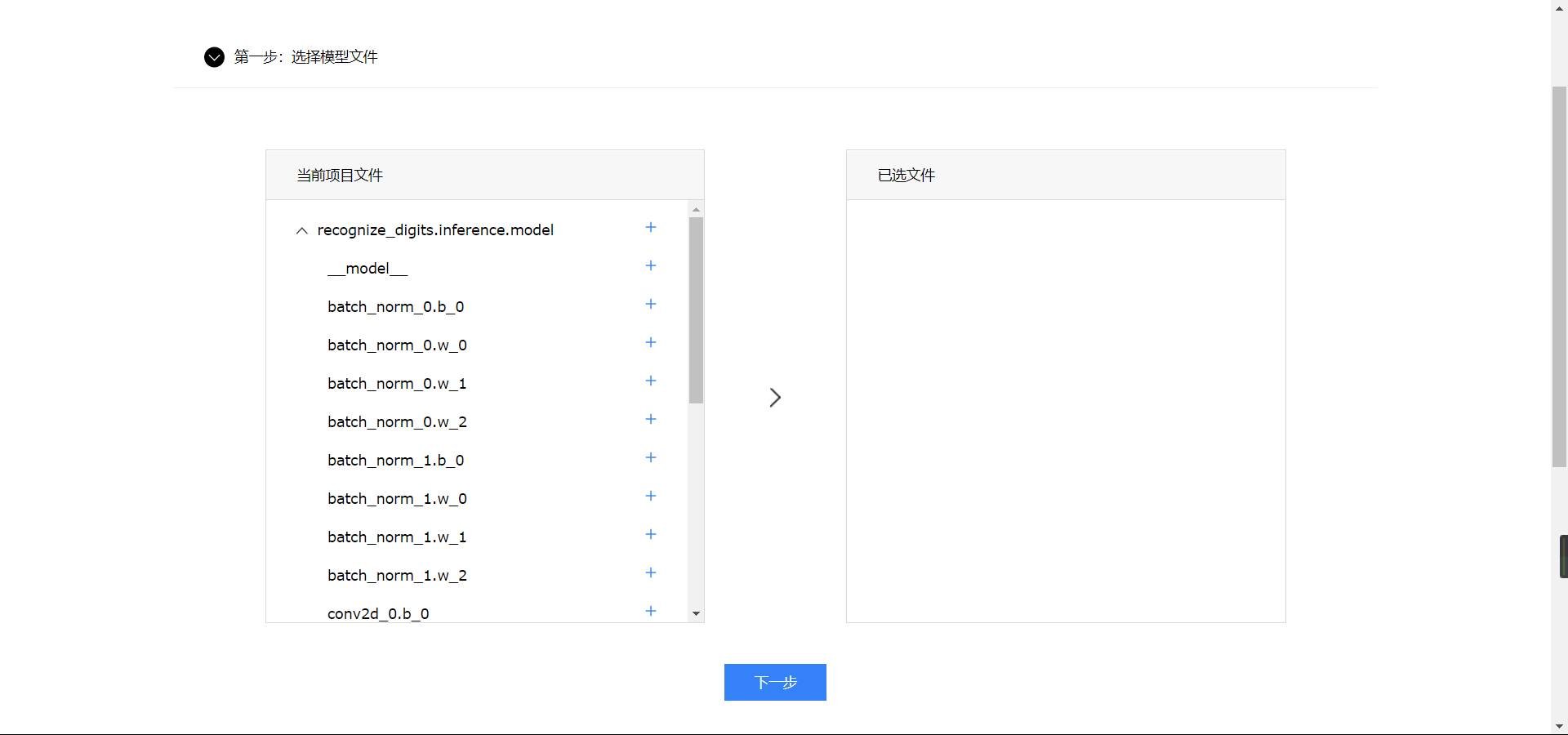

结束后在左侧【文件夹】中会看到生成的模型文件（若没有可以尝试点击【刷新】）

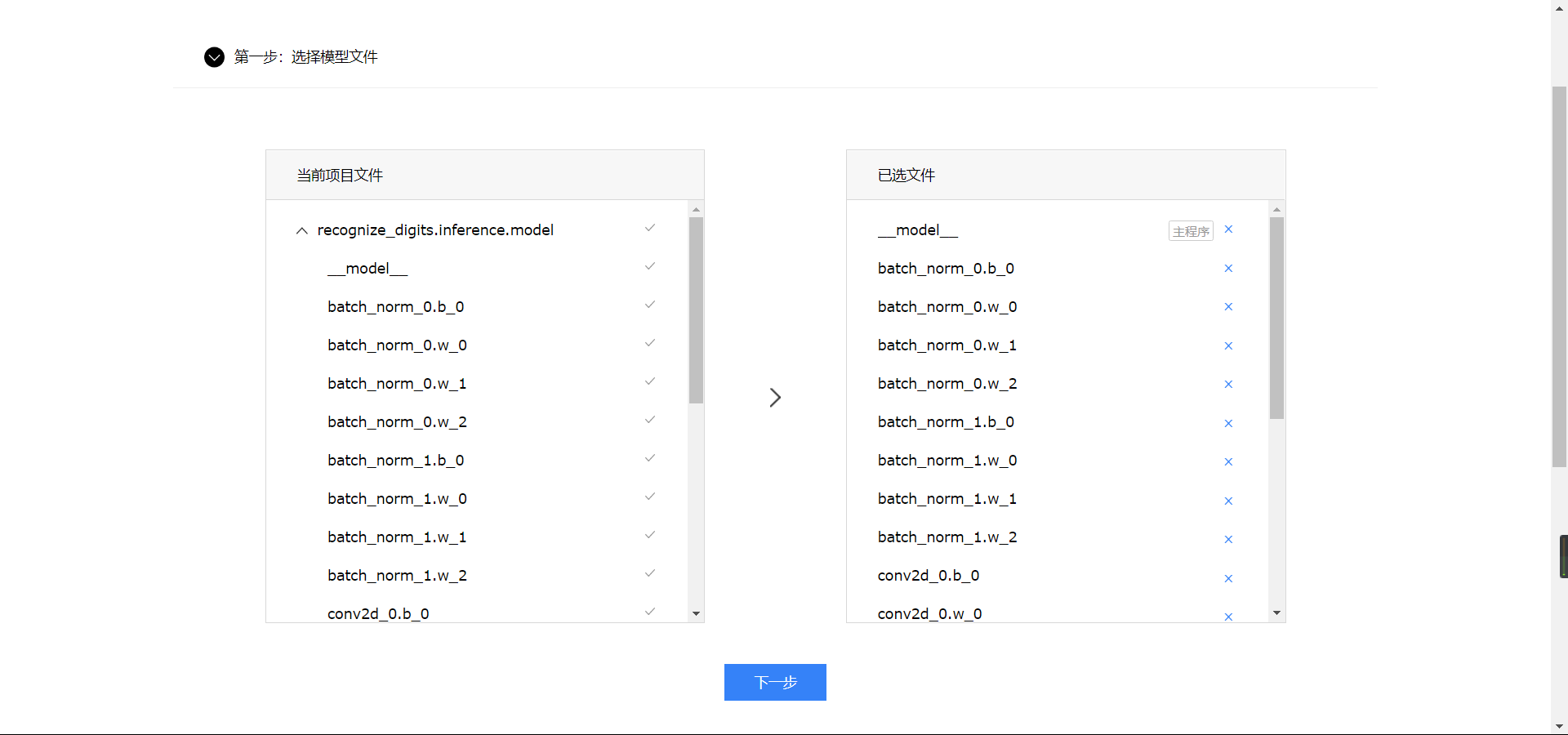


## 创建在线预测服务

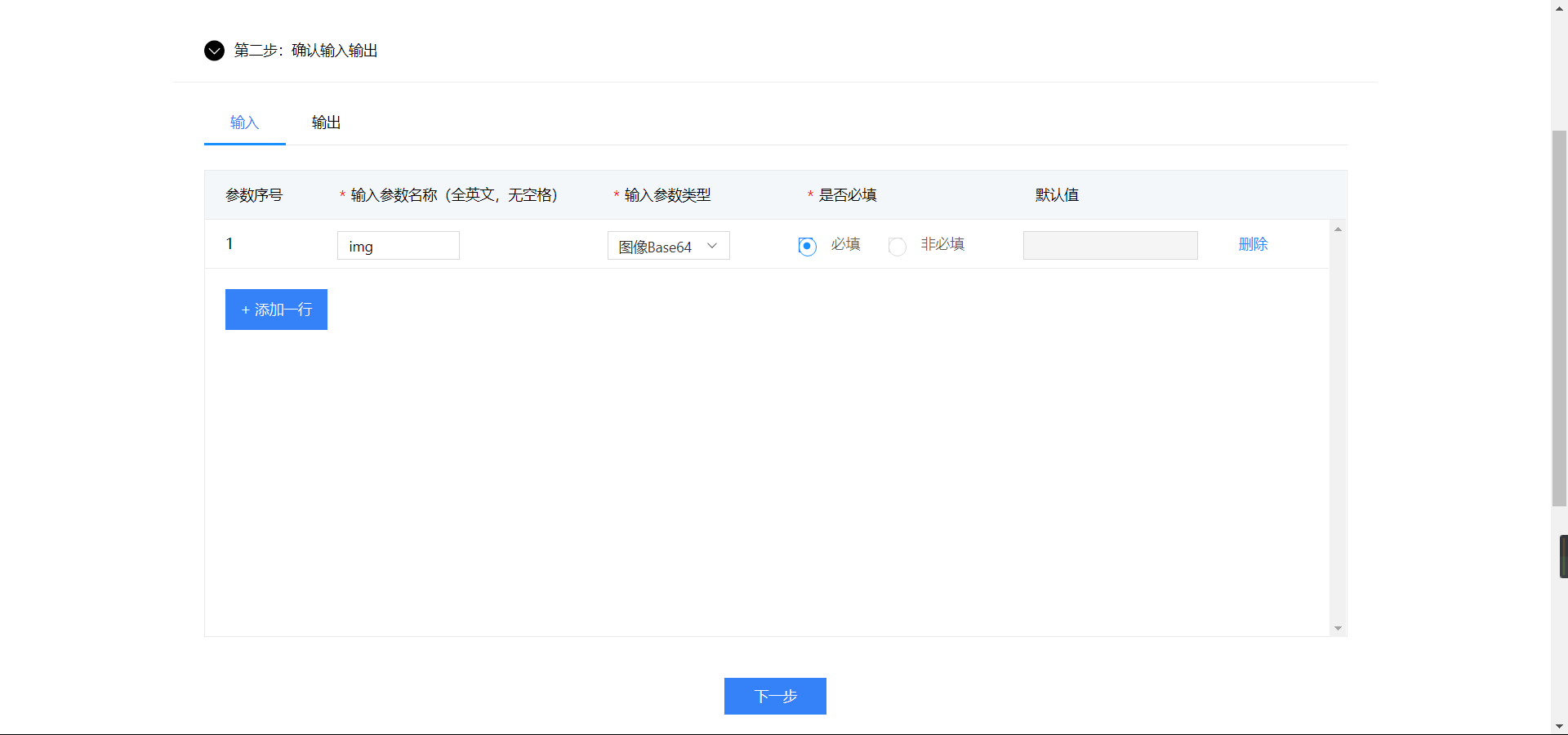
完成模型训练后, 在单机项目页面点击【创建预测服务】  


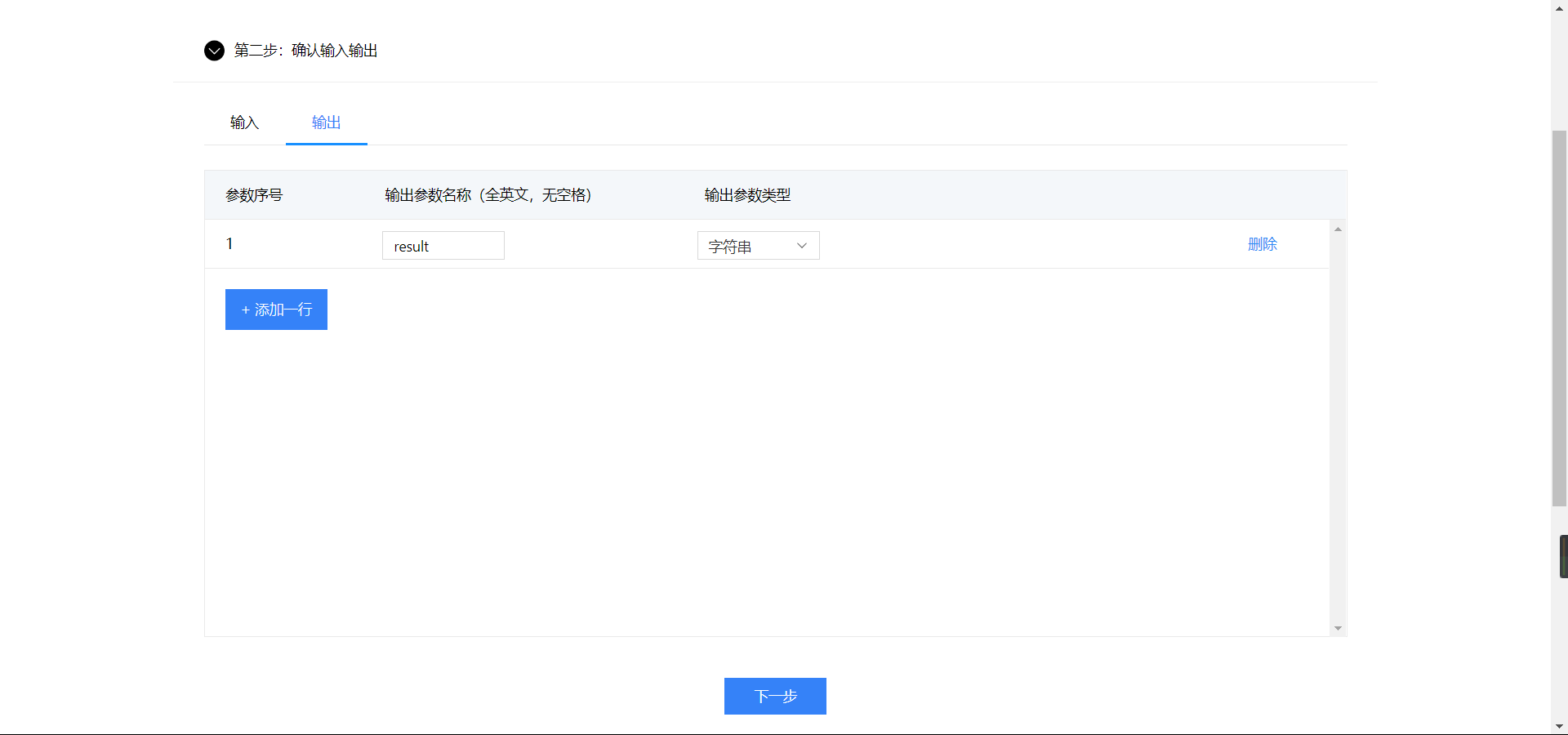
### 第一步 选择模型文件

点击recognize\_digits.inference.model左侧的箭头展开模型文件  


点击recognize\_digits.inference.model右侧的“+”选择该文件夹下的所有文件  


### 第二步 确认输入输出





### 第三步 制作参数转换器

我们使用自定义转换器

输入参数  
这里的变量img要和第二步输入参数设置的参数名相同  
对image的操作是将它变成一个 numpy 数组以匹配数据feed格式

# -\*- coding: utf-8 -\*-

"""

Hook

"""

import os

import sys

sys.path.append("..")

from PIL import Image

import numpy as np

import paddle.fluid as fluid

from home.utility import base64\_to\_image

def reader\_infer(data\_args):

"""

reader\_infer

:param data\_args: 接口传入的数据，以k-v形式

:return [[]], feeder

"""

def reader():

"""

reader

:return:

"""

img = fluid.layers.data(name='img', shape=[1,28,28], dtype='float32')

feeder = fluid.DataFeeder(place=fluid.CPUPlace(), feed\_list=[img])

image = base64\_to\_image(data\_args["img"])

image = image.convert("L")

image = image.resize((28, 28), Image.ANTIALIAS)

image = np.array(image).reshape(1, 1, 28, 28).astype(np.float32)

image = image / 255.0 \* 2.0 - 1.0

return [[image]], feeder

return reader

输出参数

# -\*- coding: utf-8 -\*-

"""

Hook

"""

import numpy as np

def output(results, data\_args):

"""

模型识别结果是各数字的概率

使用numpy的argmax方法选出概率最大的作为识别结果

:param results 模型预测结果

:param data\_args: 接口传入的数据，以k-v形式

:return predict 需要能被json\_encode的数据格式

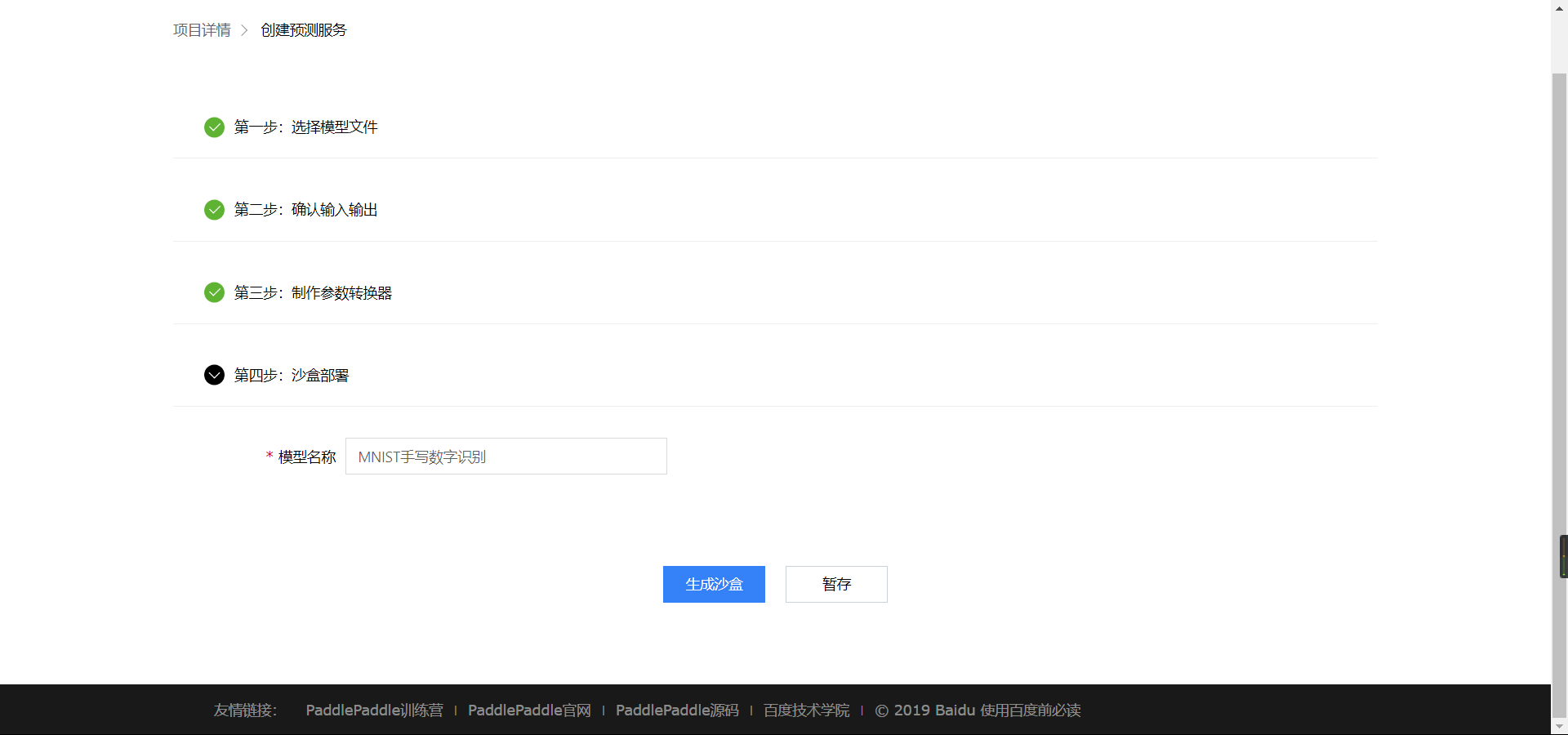
"""

result = np.argmax(results)

predict = {'predict':result}

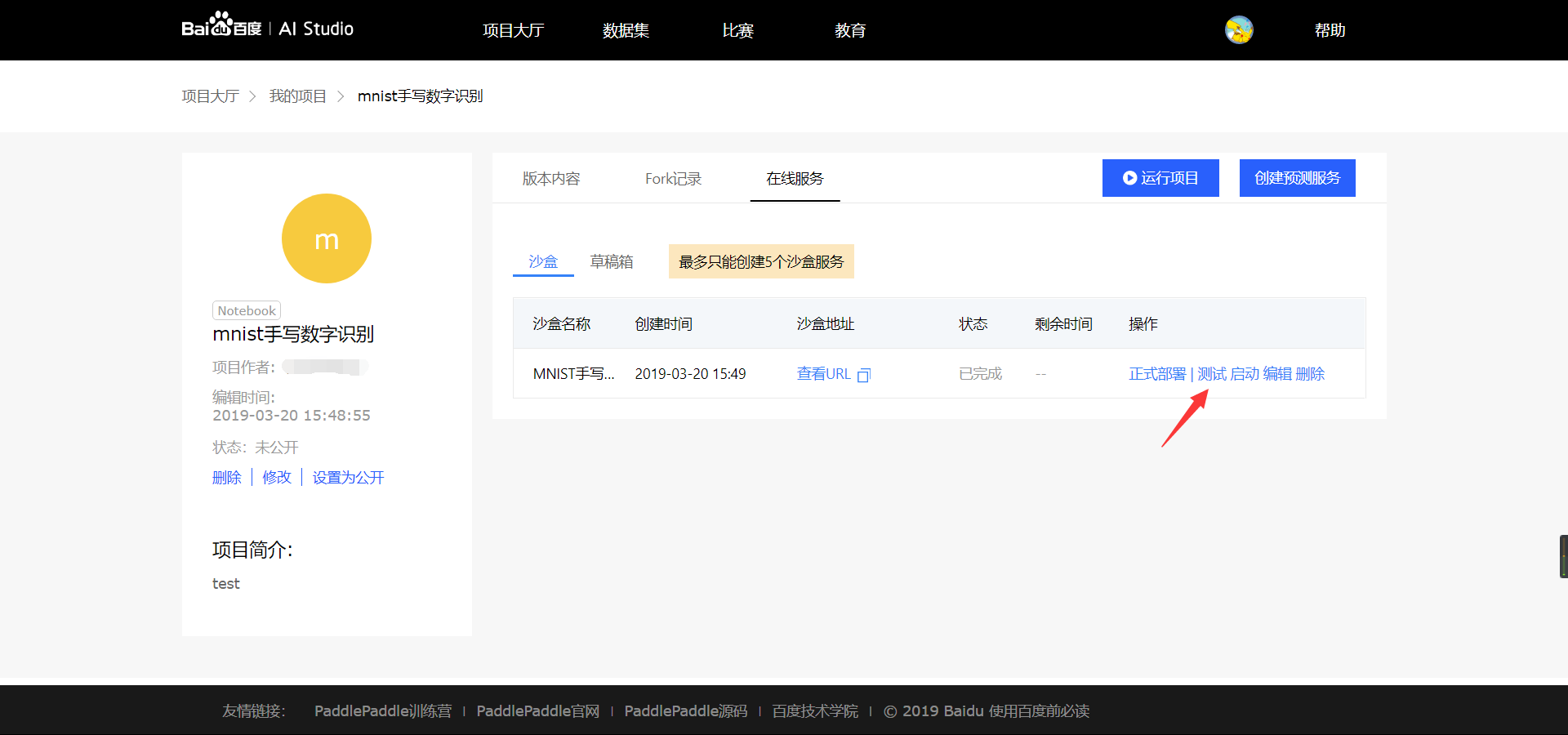
return predict

### 第四步 沙盒部署

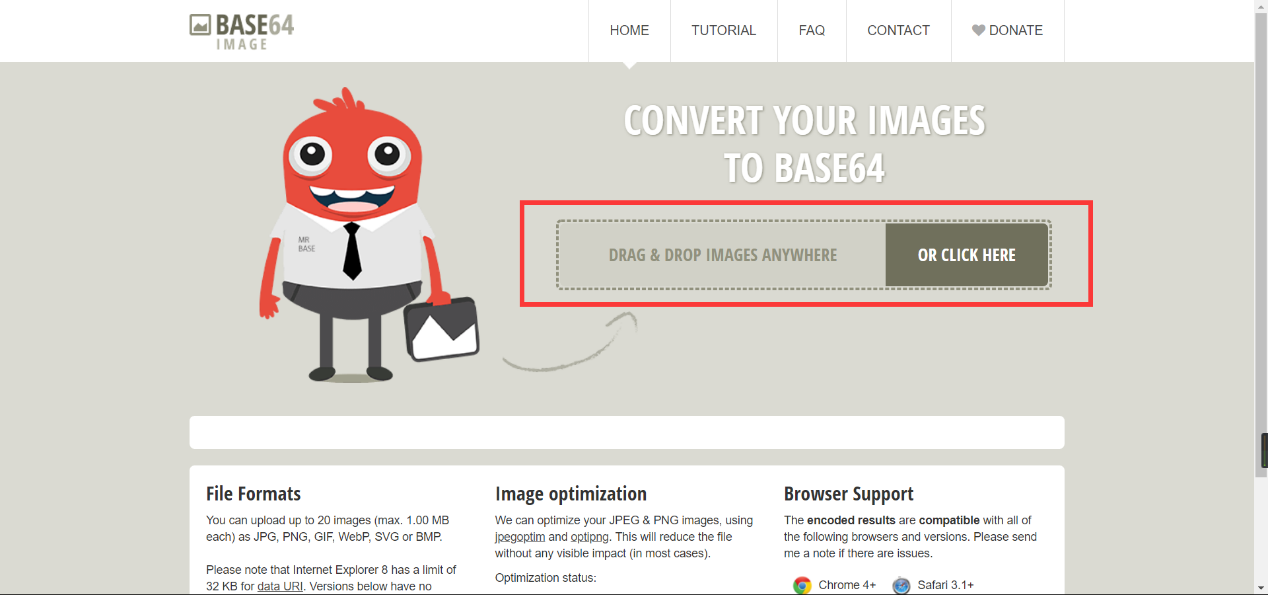
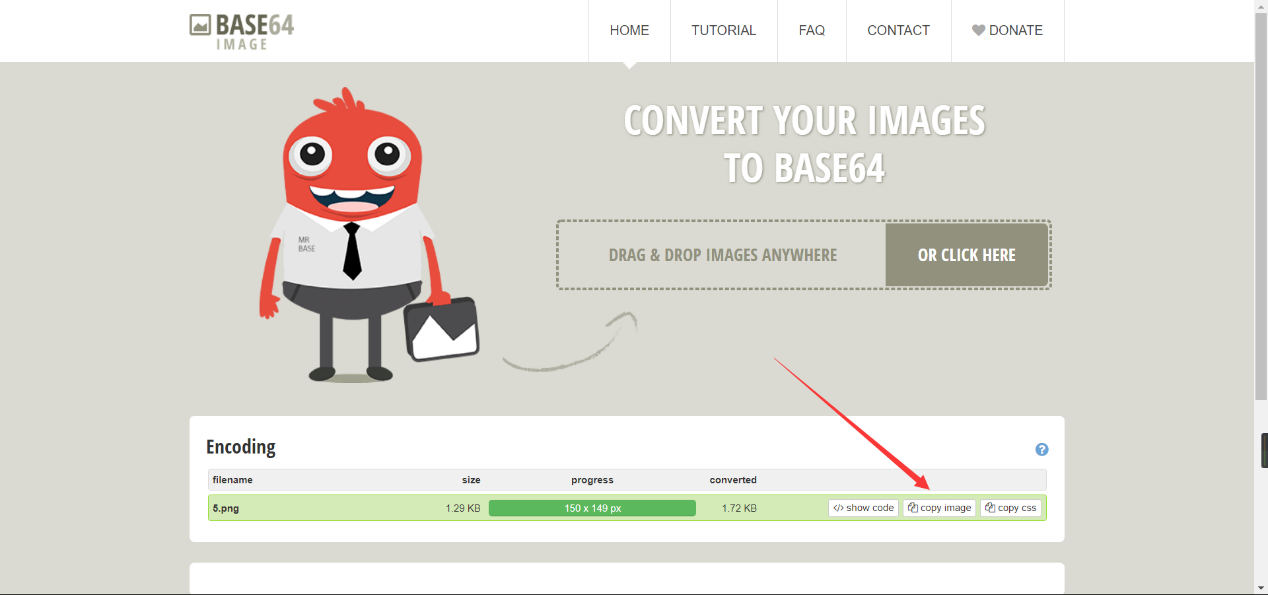
输入名称，点击【生成沙盒】  


## 测试沙盒服务

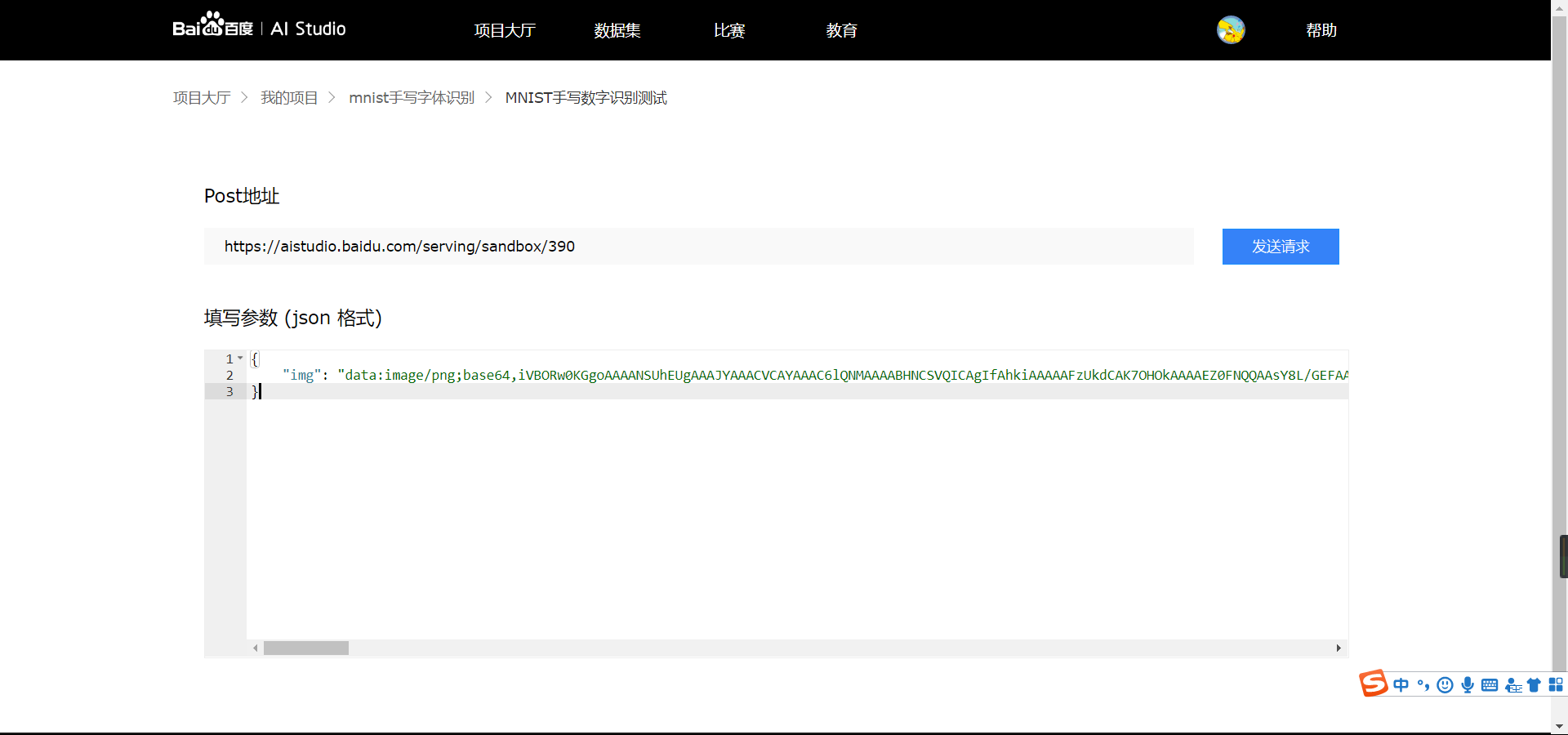
### 第一步 点击【测试】进入测试页面



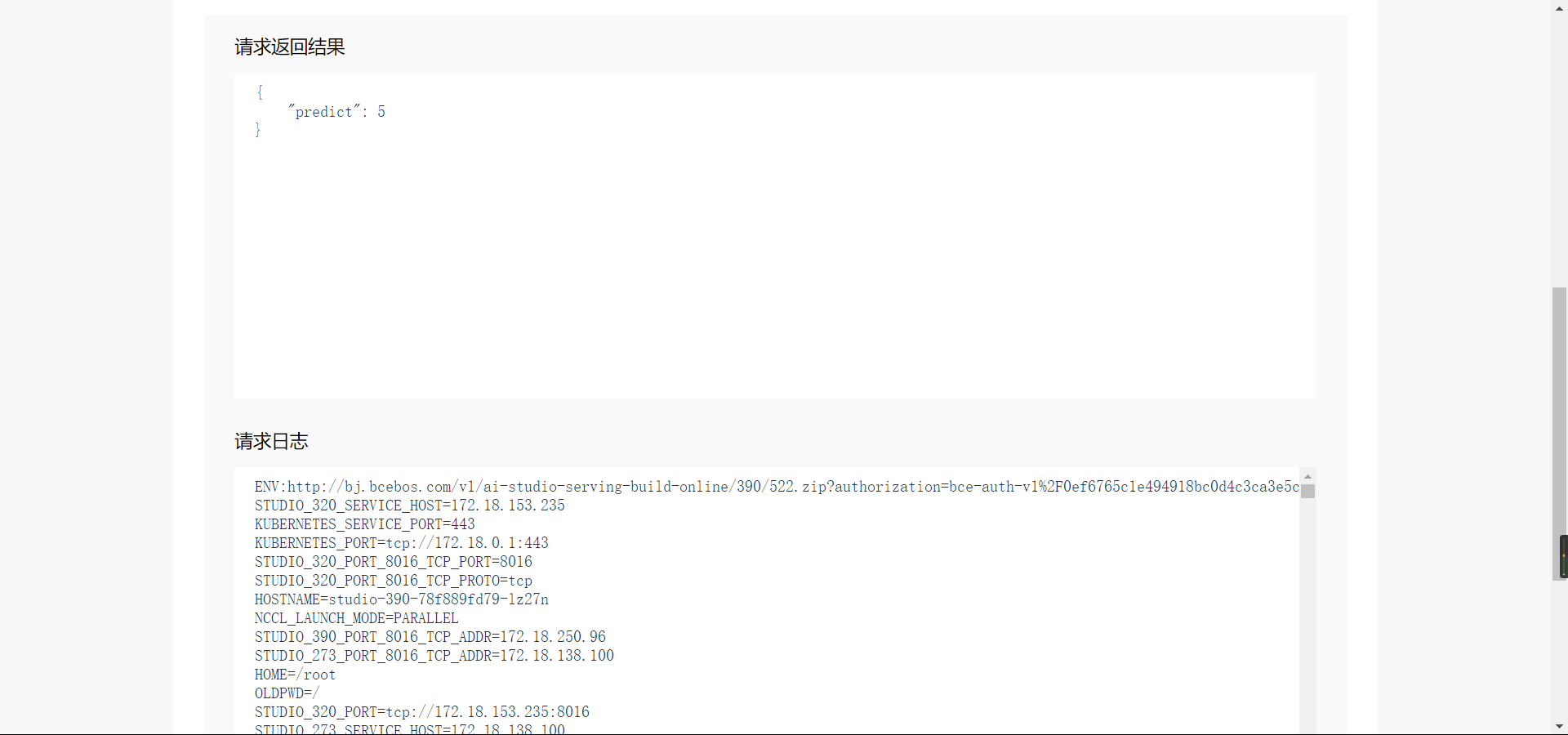
### 第二步 将测试图片转为BASE64格式

打开<https://www.base64-image.de/>，将测试图片拖入指定区域转换格式  
  
转换成功后点击【copy image】复制转换后的代码  


### 第三步 填写json格式请求参数

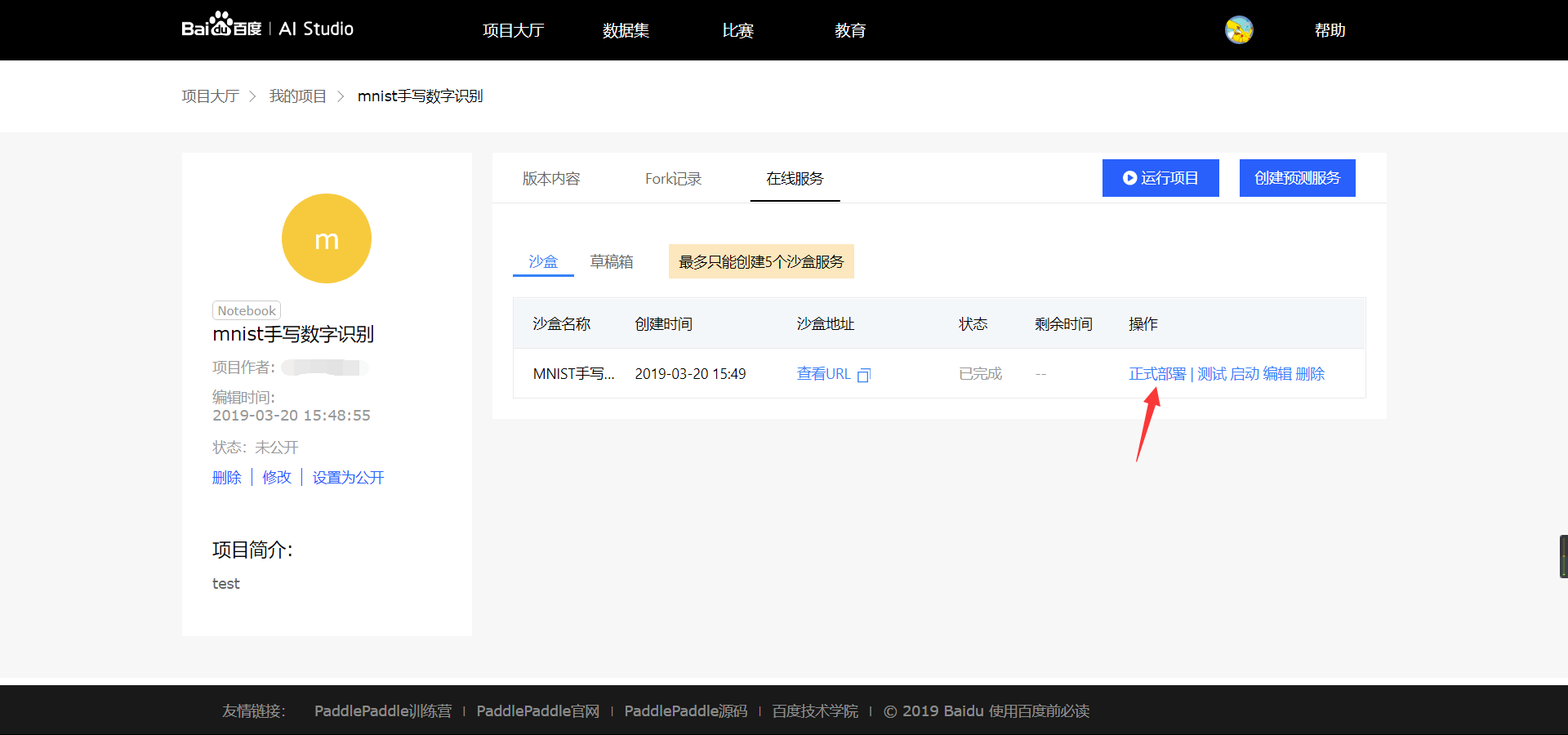
使用刚才复制的图片代码替换img的value  


### 第四步 点击【发送请求】校验返回结果

点击后片刻便会看到【请求结果】和【请求日志】  


如果【请求结果】中为错误信息，可在【请求日志】中查看更具体的报错，之后点击最下方的【进入编辑页面】对之前的预测服务进行修改即可。

## 部署在线服务

点击【返回沙盒列表】后，点击沙盒后的【正式部署】即可部署线上API  


## 调用在线服务

依据API key、服务地址和用户自定义参数, 实现对服务的调用

**请求URL在Android调用API中会使用**

### 请求方式

* HTTP请求URL: [服务地址] [?] [apiKey=xxx]
* HTTP请求方法: POST
* HTTP Body: 用户自定义参数  
  