公开项目 > 用PaddlePaddle实现人脸识别_人工智能导论

用PaddlePaddle实现人脸识别_人工智能导论

Fork 13 ♡ 喜欢 1

使用PaddlePaddle来实现人脸识别，通过构建CNN和VGG网络在明星数据集（章子怡、姜文、彭于晏三位明星人脸图片）上进行训练和预测，感谢「asaxam」同学的

f flyingcatty 2枚 AI Studio 经典版 1.8.0 Python3 中级 计算机视觉 深度学习 分类 2022-03-27 22:54:49

版本内容 数据集 Fork记录 评论(0)

人脸识别V1 2022-03-27 23:24:51

请选择预览文件

新版Notebook- BML CodeLab上线，fork后可修改项目版本进行体验

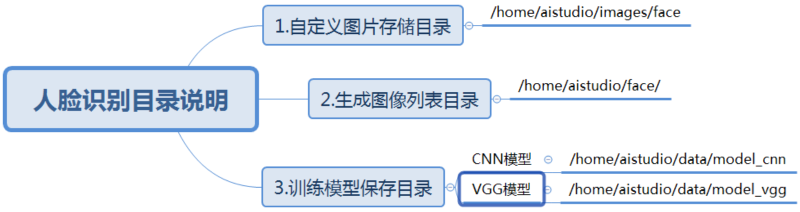
下面是代码的整个结构目录:

【1.用来存放自定义图片的目录——/home/aistudio/images/face】

【2.用来存放图像列表的目录——/home/aistudio/face/】

【3.model_vgg用来存放vgg网络训练的模型】

【4.model_cnn用来存放cnn网路训练的模型】



用%pwd查看当前所在目录

In [1]
```
%pwd
```
```
'/home/aistudio'
```

In [2]
```
#解压数据集
!unzip -qo /home/aistudio/data/data12039/images.zip -d /home/aistudio/
```

In [3]
```
!ls /home/aistudio/images/face
```
```
jiangwen  pengyuyan  zhangziyi
```

准备数据
配置网络
训练网络
模型评估
模型预测

Step1： 准备数据。

数据集介绍

数据集中章子怡、姜文、彭于晏三位明星的人脸图片。 总计317张图片，章子怡100张，姜文103张，彭于晏114张。按照9:1的比例进行划分，90%用于训练，10%用于测试。

自定义的数据集，首先要生成图像列表，把自定的图像分为测试集和训练集，并带有标签。下面的程序可以单独运行，只要把一个大类的文件夹路径传进去就可以了,该程序会把里面的每个小类别都迭代,生成固定格式的列表.比如我们把人脸类别的根目录传进去../images/face。最后会在指定目录下面生成三个文件，readme.json、trainer.list和test.list.

In [4]
```
import os
```

```python
import json

# 设置要生成文件的路径
data_root_path = '/home/aistudio/images/face'
# 所有类别的信息
class_detail = []
# 获取所有类别保存的文件夹名称，这里是['zhangziyi', 'jiangwen', 'pengyuyan']
class_dirs = os.listdir(data_root_path)
# 类别标签
class_label_dict = {'zhangziyi': 0, 'jiangwen': 1, 'pengyuyan': 2}
# 获取总类别的名称
father_paths = data_root_path.split('/')    #['', 'home', 'aistudio', 'imag
while True:
    if father_paths[father_paths.__len__() - 1] == '':
        del father_paths[father_paths.__len__() - 1]
    else:
        break
father_path = father_paths[father_paths.__len__() - 1]
# 把生产的数据列表都放在自己的总类别文件夹中
data_list_path = '/home/aistudio/%s/' % father_path
# 如果不存在这个文件夹,就创建
isexist = os.path.exists(data_list_path)
if not isexist:
    os.makedirs(data_list_path)
# 清空原来的数据
with open(data_list_path + "test.list", 'w') as f:
    pass
with open(data_list_path + "trainer.list", 'w') as f:
    pass
# 总的图像数量
all_class_images = 0
# 读取每个类别
for class_dir in class_dirs:
    # 每个类别的信息
    class_detail_list = {}
    test_sum = 0
    trainer_sum = 0
    # 统计每个类别有多少张图片
    class_sum = 0
    # 获取类别路径
    path = data_root_path + "/" + class_dir
    # 获取所有图片
    img_paths = os.listdir(path)

    for img_path in img_paths:                                  # 遍历文件夹
        name_path = path + '/' + img_path                       # 每张图片的
        if class_sum % 10 == 0:                                 # 每10张图片
            test_sum += 1                                       #test_sum测
            with open(data_list_path + "test.list", 'a') as f:
                f.write(name_path + "\t%d" % class_label_dict[class_dir] +
        else:
            trainer_sum += 1                                    #trainer_su
            with open(data_list_path + "trainer.list", 'a') as f:
                f.write(name_path + "\t%d" % class_label_dict[class_dir] +
        class_sum += 1                                          #每类图片的数
        all_class_images += 1                                   #所有类图片的

    # 说明的json文件的class_detail数据
    class_detail_list['class_name'] = class_dir             #类别名称，如jiar
    class_detail_list['class_label'] = class_label_dict[class_dir]
    class_detail_list['class_test_images'] = test_sum       #该类数据的测试集
    class_detail_list['class_trainer_images'] = trainer_sum #该类数据的训练集
    class_detail.append(class_detail_list)

# 获取类别数量
all_class_sum = class_dirs.__len__()
# 说明的json文件信息
readjson = {}
readjson['all_class_name'] = father_path                   #文件父目录
readjson['all_class_sum'] = all_class_sum                 #
readjson['all_class_images'] = all_class_images
readjson['class_detail'] = class_detail
jsons = json.dumps(readjson, sort_keys=True, indent=4, separators=(',', ':
with open(data_list_path + "readme.json",'w') as f:
    f.write(jsons)
print ('生成数据列表完成！')
print ("标签及其类别：{}".format(class_label_dict))
```

```
生成数据列表完成!
标签及其类别：{'zhangziyi': 0, 'jiangwen': 1, 'pengyuyan': 2}
```

In [5]

```
ls /home/aistudio/face/
```

```
readme.json  test.list  trainer.list
```

In [6]

```
cat /home/aistudio/face/readme.json
```

```
{
    "all_class_images": 317,
    "all_class_name": "face",
    "all_class_sum": 3,
    "class_detail": [
        {
            "class_label": 1,
            "class_name": "jiangwen",
            "class_test_images": 11,
            "class_trainer_images": 92
        },
        {
            "class_label": 2,
            "class_name": "pengyuyan",
            "class_test_images": 12,
            "class_trainer_images": 102
        },
        {
            "class_label": 0,
            "class_name": "zhangziyi",
            "class_test_images": 10,
            "class_trainer_images": 90
        }
    ]
}
```

In [7]

```
#导入要用到的模块
import paddle
import paddle.fluid as fluid
import numpy
import sys
import os
from multiprocessing import cpu_count
import matplotlib.pyplot as plt
```

```
2022-03-27 23:22:19,285-INFO: font search path ['/opt/conda/envs/python35-pa
2022-03-27 23:22:19,629-INFO: generated new fontManager
```

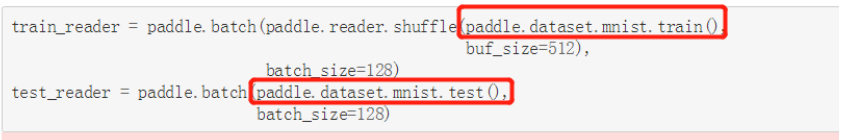
train_reader和test_reader分别用于获取训练集和测试集 paddle.reader.shuffle()表示每次缓存BUF_SIZE个数据项，并进行打乱 paddle.batch()表示每BATCH_SIZE组成一个batch

自定义数据集需要先定义自己的reader，把图像数据处理一些，并输出图片的数组和标签。

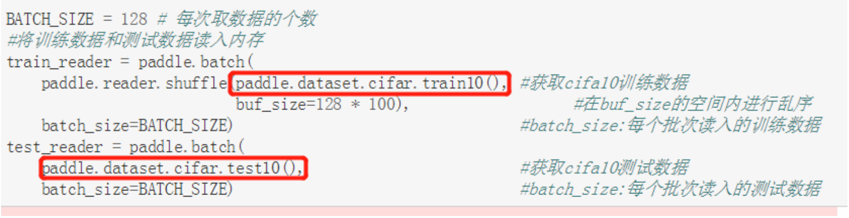
In [8]

```
# 定义训练的mapper
# train_mapper函数的作用是用来对训练集的图像进行处理修剪和数组变换，返回img数组和标签
# sample是一个python元组，里面保存着图片的地址和标签。 ('../images/face/zhangziyi
def train_mapper(sample):
    img, label = sample
    # 进行图片的读取，由于数据集的像素维度各不相同，需要进一步处理对图像进行变换
    img = paddle.dataset.image.load_image(img)
    #进行了简单的图像变换，这里对图像进行crop修剪操作，输出img的维度为(3, 100, 100)
    img = paddle.dataset.image.simple_transform(im=img,          #输入图片是
                                                resize_size=100, # 剪裁图片
                                                crop_size=100,
                                                is_color=True,  #彩色图像
                                                is_train=True)
    #将img数组进行进行归一化处理，得到0到1之间的数值
    img= img.flatten().astype('float32')/255.0
    return img, label
# 对自定义数据集创建训练集train的reader
def train_r(train_list, buffered_size=1024):
    def reader():
        with open(train_list, 'r') as f:
            # 将train.list里面的标签和图片的地址方法一个list列表里面，中间用\t隔开'
            #../images/face/jiangwen/0b1937e2-f929-11e8-8a8a-005056c00008.
            lines = [line.strip() for line in f]
            for line in lines:
                # 图像的路径和标签是以\t来分割的,所以我们在生成这个列表的时候,使用\t
                img_path, lab = line.strip().split('\t')
                yield img_path, int(lab)
    # 创建自定义数据训练集的train_reader
    return paddle.reader.xmap_readers(train_mapper, reader,cpu_count(), bu

# sample是一个python元组，里面保存着图片的地址和标签。 ('../images/face/zhangziyi
def test_mapper(sample):
    img, label = sample
    img = paddle.dataset.image.load_image(img)
    img = paddle.dataset.image.simple_transform(im=img, resize_size=100, c
    img= img.flatten().astype('float32')/255.0
    return img, label

# 对自定义数据集创建验证集test的reader
def test_r(test_list, buffered_size=1024):
    def reader():
        with open(test_list, 'r') as f:
            lines = [line.strip() for line in f]
            for line in lines:
                #图像的路径和标签是以\t来分割的,所以我们在生成这个列表的时候,使用\t就
                img_path, lab = line.strip().split('\t')
                yield img_path, int(lab)

    return paddle.reader.xmap_readers(test_mapper, reader,cpu_count(), buf
```

对比一下手写数字识别和猫狗分类创建reader的代码



```
train_reader = paddle.batch(paddle.reader.shuffle(paddle.dataset.mnist.train(),
                                                  buf_size=512),
                            batch_size=128)
test_reader = paddle.batch(paddle.dataset.mnist.test(),
                           batch_size=128)
```



```
BATCH_SIZE = 128 # 每次取数据的个数
#将训练数据和测试数据读入内存
train_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.cifar.train10(), #获取cifa10训练数据
                          buf_size=128 * 100),                #在buf_size的空间内进行乱序
    batch_size=BATCH_SIZE)                                #batch_size:每个批次读入的训练数据
test_reader = paddle.batch(
    paddle.dataset.cifar.test10(),                        #获取cifa10测试数据
    batch_size=BATCH_SIZE)                                #batch_size:每个批次读入的测试数据
```

In [9]

```
BATCH_SIZE = 32
# 把图片数据生成reader
trainer_reader = train_r(train_list="/home/aistudio/face/trainer.list")
train_reader = paddle.batch(
    paddle.reader.shuffle(
        reader=trainer_reader,buf_size=300),
    batch_size=BATCH_SIZE)

tester_reader = test_r(test_list="/home/aistudio/face/test.list")
test_reader = paddle.batch(
     tester_reader, batch_size=BATCH_SIZE)
```

打印看下数据是什么样的？PaddlePaddle接口提供的数据已经经过了归一化、居中等处理 尝试打印一下，观察一下自定义的数据集

In [10]

```
train_data = paddle.batch(trainer_reader,
                            batch_size=3)
sampledata=next(train_data())
print(sampledata)
```

```
[(array([1.        , 1.        , 1.        , ..., 0.02352941, 0.02745098,
       0.02745098], dtype=float32), 1), (array([0.8039216 , 0.80784315, 0.8
       0.4509804 ], dtype=float32), 1), (array([0.48235294, 0.43137255, 0.3
       0.03921569], dtype=float32), 1)]
```

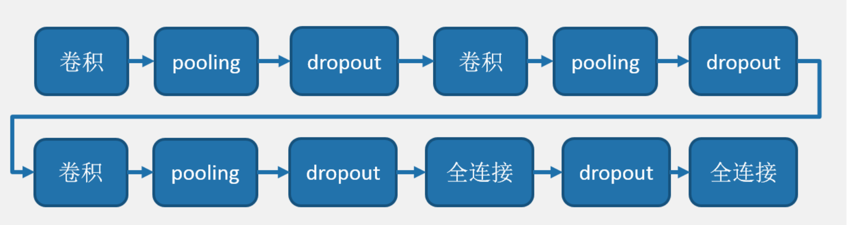
准备数据
配置网络
训练网络
模型评估
模型预测

Step2.网络配置

（1）搭建网络

配置网络主要是用来组建一个Program，主要包括三个部分：1.网络模型2.损失函数3.优化函数

搭建的CNN网络



In [11]

```python
def convolutional_neural_network(image, type_size):
    # 第一个卷积--池化层
    conv_pool_1 = fluid.nets.simple_img_conv_pool(input=image,# 输入图像
                                                       filter_size=3,# 滤波
                                                       num_filters=32,# fi
                                                       pool_size=2,# 池化层
                                                       pool_stride=2,# 池化
                                                       act='relu') # 激活类

    # Dropout主要作用是减少过拟合，随机让某些权重不更新
    # Dropout是一种正则化技术，通过在训练过程中阻止神经元节点间的联合适应性来减少过拟合
    # 根据给定的丢弃概率dropout随机将一些神经元输出设置为0，其他的仍保持不变。
    drop = fluid.layers.dropout(x=conv_pool_1, dropout_prob=0.5)

    # 第二个卷积--池化层
    conv_pool_2 = fluid.nets.simple_img_conv_pool(input=drop,
                                                       filter_size=3,
                                                       num_filters=64,
                                                       pool_size=2,
                                                       pool_stride=2,
                                                       act='relu')
    # 减少过拟合，随机让某些权重不更新
    drop = fluid.layers.dropout(x=conv_pool_2, dropout_prob=0.5)

    # 第三个卷积--池化层
    conv_pool_3 = fluid.nets.simple_img_conv_pool(input=drop,
                                                       filter_size=3,
                                                       num_filters=64,
                                                       pool_size=2,
                                                       pool_stride=2,
                                                       act='relu')
    # 减少过拟合，随机让某些权重不更新
    drop = fluid.layers.dropout(x=conv_pool_3, dropout_prob=0.5)

    # 全连接层
    fc = fluid.layers.fc(input=drop, size=512, act='relu')
    # 减少过拟合，随机让某些权重不更新
    drop =  fluid.layers.dropout(x=fc, dropout_prob=0.5)
    # 输出层 以softmax为激活函数的全连接输出层，输出层的大小为图像类别type_size个数
    predict = fluid.layers.fc(input=drop,size=type_size,act='softmax')

    return predict
```

搭建VGG网络

1.首先定义了一组卷积网络，即conv_block。卷积核大小为3x3，池化窗口大小为2x2，窗口滑动大小为2，groups决定每组VGG模块是几次连续的卷积操作，dropouts指定Dropout操作的概率。所使用的img_conv_group是在paddle.networks中预定义的模块，由若干组 Conv->BN->ReLu->Dropout 和 一组 Pooling 组成。

2.五组卷积操作，即 5个conv_block。 第一、二组采用两次连续的卷积操作。第三、四、五组采用三次连续的卷积操作。每组最后一个卷积后面Dropout概率为0，即不使用Dropout操作。

3.最后接两层512维的全连接。

4.通过上面VGG网络提取高层特征，然后经过全连接层映射到类别维度大小的向量，再通过Softmax归一化得到每个类别的概率，也可称作分类器。

In [12]:
```
def vgg_bn_drop(image, type_size):
    def conv_block(ipt, num_filter, groups, dropouts):
        return fluid.nets.img_conv_group(
            input=ipt, # 具有[N, C, H, W]格式的输入图像
            pool_size=2,
            pool_stride=2,
            conv_num_filter=[num_filter] * groups, # 过滤器个数
            conv_filter_size=3, # 过滤器大小
            conv_act='relu',
            conv_with_batchnorm=True, # 表示在 Conv2d Layer 之后是否使用 Batcl
            conv_batchnorm_drop_rate=dropouts,# 表示 BatchNorm 之后的 Dropou
            pool_type='max') # 最大池化

    conv1 = conv_block(image, 64, 2, [0.0, 0])
    conv2 = conv_block(conv1, 128, 2, [0.0, 0])
    conv3 = conv_block(conv2, 256, 3, [0.0, 0.0, 0])
    conv4 = conv_block(conv3, 512, 3, [0.0, 0.0, 0])
    conv5 = conv_block(conv4, 512, 3, [0.0, 0.0, 0])

    drop = fluid.layers.dropout(x=conv5, dropout_prob=0.5)
    fc1 = fluid.layers.fc(input=drop, size=512, act=None)

    bn = fluid.layers.batch_norm(input=fc1, act='relu')
    drop2 = fluid.layers.dropout(x=bn, dropout_prob=0.0)
    fc2 = fluid.layers.fc(input=drop2, size=512, act=None)
    predict = fluid.layers.fc(input=fc2, size=type_size, act='softmax')
    return predict
```

（2）定义数据层

image 和 label 是通过 fluid.layers.data 创建的两个输入数据层。其中 image 是 [3, 100, 100] 维度的浮点数据; label 是 [1] 维度的整数数据。

这里需要注意的是: Fluid中默认使用 -1 表示 batch size 维度，默认情况下会在 shape 的第一个维度添加 -1 。 所以 上段代码中， 我们可以接受将一个 [-1, 3, 100, 100] 的numpy array传给 image 。 Fluid中用来做类别标签的数据类型是 int64，并且标签从0开始。

In [13]:
```
image = fluid.layers.data(name='image', shape=[3, 100, 100], dtype='float32

label = fluid.layers.data(name='label', shape=[1], dtype='int64')
print('image_shape:',image.shape)
```

```
image_shape: (-1, 3, 100, 100)
```

（3）获取分类器

注：type_size要和需要分类的类别数量保持一致

In [14]:
```
# ##### 获取分类器，用cnn或者vgg网络进行分类type_size要和训练的类别一致 ########
predict = convolutional_neural_network(image=image, type_size=4)
#predict = vgg_bn_drop(image=image, type_size=4)
```

（4）定义损失函数和准确率

这次使用的是交叉熵损失函数，该函数在分类任务上比较常用。

定义了一个损失函数之后，还有对它求平均值，因为定义的是一个Batch的损失值。

同时我们还可以定义一个准确率函数，这个可以在我们训练的时候输出分类的准确率。

In [15]:
```
# 获取损失函数和准确率
cost = fluid.layers.cross_entropy(input=predict, label=label)
# 计算cost中所有元素的平均值
avg_cost = fluid.layers.mean(cost)
#计算准确率
accuracy = fluid.layers.accuracy(input=predict, label=label)
```

（5）定义优化方法

接着是定义优化方法，这次我们使用的是Adam优化方法，同时指定学习率为0.001。

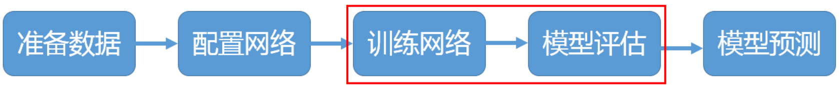
In [16]
```
# 定义优化方法
optimizer = fluid.optimizer.Adam(learning_rate=0.001)    # Adam是一阶基于梯度

optimizer.minimize(avg_cost)                             # 取局部最优化的平均
print(type(accuracy))
```

```
<class 'paddle.fluid.framework.Variable'>
```

在上述模型配置完毕后，得到两个fluid.Program：fluid.default_startup_program() 与 fluid.default_main_program() 配置完毕了。

参数初始化操作会被写入fluid.default_startup_program()

fluid.default_main_program()用于获取默认或全局main program(主程序)。该主程序用于训练和测试模型。fluid.layers 中的所有layer函数可以向 default_main_program 中添加算子和变量。default_main_program 是fluid的许多编程接口（API）的Program参数的缺省值。例如,当用户program没有传入的时候， Executor.run() 会默认执行 default_main_program 。



Step3.模型训练 and Step4.模型评估

（1）创建Executor

首先定义运算场所 fluid.CPUPlace()和 fluid.CUDAPlace(0)分别表示运算场所为CPU和GPU

Executor:接收传入的program，通过run()方法运行program。

训练分为三步：第一步配置好训练的环境，第二步用训练集进行训练，并用验证集对训练进行评估，不断优化，第三步保存好训练的模型

In [17]
```
# 使用CPU进行训练
place = fluid.CPUPlace()
# 创建一个executor
exe = fluid.Executor(place)
# 对program进行参数初始化1.网络模型2.损失函数3.优化函数
exe.run(fluid.default_startup_program())
```

```
[]
```

（2）定义数据映射器

DataFeeder负责将数据提供器（train_reader,test_reader）返回的数据转成一种特殊的数据结构，使其可以输入到Executor中。

feed_list设置向模型输入的向变量表或者变量表名

In [18]
```
# 定义输入数据的维度,DataFeeder 负责将reader(读取器)返回的数据转成一种特殊的数据结构,
feeder = fluid.DataFeeder(feed_list=[image, label], place=place)#定义输入数据
```

(3)展示模型训练曲线

In [19]

```
all_train_iter=0
all_train_iters=[]
all_train_costs=[]
all_train_accs=[]

def draw_train_process(title,iters,costs,accs,label_cost,lable_acc):
    plt.title(title, fontsize=24)
    plt.xlabel("iter", fontsize=20)
    plt.ylabel("cost/acc", fontsize=20)
    plt.plot(iters, costs,color='red',label=label_cost)
    plt.plot(iters, accs,color='green',label=lable_acc)
    plt.legend()
    plt.grid()
    plt.show()
```

（4）训练并保存模型

Executor接收传入的program,并根据feed map(输入映射表)和fetch_list(结果获取表) 向program中添加feed operators(数据输入算子)和fetch operators（结果获取算子)。

feed map为该program提供输入数据。fetch_list提供program训练结束后用户预期的变量。

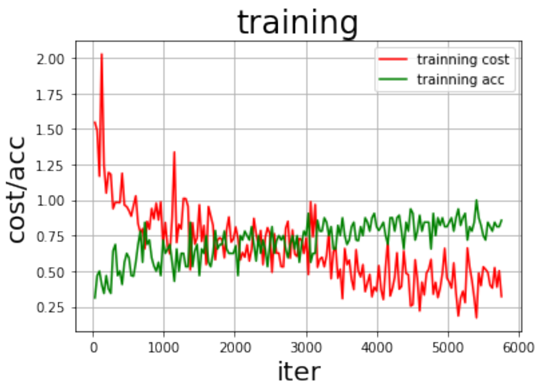
这次训练5个Pass。每一个Pass训练结束之后，再使用验证集进行验证，并求出相应的损失值Cost和准确率acc。

In [20]

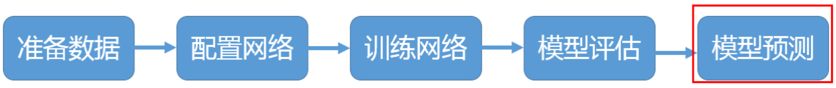
```
# 训练的轮数
EPOCH_NUM = 20
print('开始训练...')
#两种方法，用两个不同的路径分别保存训练的模型
#model_save_dir = "/home/aistudio/data/model_vgg"
model_save_dir = "/home/aistudio/data/model_cnn"
for pass_id in range(EPOCH_NUM):
    train_cost = 0
    for batch_id, data in enumerate(train_reader()):
        train_cost, train_acc = exe.run(
            program=fluid.default_main_program(),
            feed=feeder.feed(data),
            fetch_list=[avg_cost, accuracy])

        all_train_iter=all_train_iter+BATCH_SIZE
        all_train_iters.append(all_train_iter)
        all_train_costs.append(train_cost[0])
        all_train_accs.append(train_acc[0])


        if batch_id % 10 == 0:
```

```
    test_accs = []
    test_costs = []
    # 每训练一轮 进行一次测试
    for batch_id, data in enumerate(test_reader()):
         test_cost, test_acc = exe.run(program=fluid.default_main_program(
                                       feed=feeder.feed(data),
                                       fetch_list=[avg_cost, accuracy])
         test_accs.append(test_acc[0])
         test_costs.append(test_cost[0])

   # 求测试结果的平均值
    test_cost = (sum(test_costs) / len(test_costs))
    test_acc = (sum(test_accs) / len(test_accs))
    print('Test:%d, Cost:%0.5f, ACC:%0.5f' % (pass_id, test_cost, test_acc


    # 如果保存路径不存在就创建
    if not os.path.exists(model_save_dir):
        os.makedirs(model_save_dir)
    # 保存训练的模型，executor 把所有相关参数保存到 dirname 中
    fluid.io.save_inference_model(dirname=model_save_dir,
                                    feeded_var_names=["image"],
                                    target_vars=[predict],
                                        executor=exe)

 draw_train_process("training",all_train_iters,all_train_costs,all_tr

 print('训练模型保存完成！')
```

```
开始训练...

Pass 0, Step 0, Cost 1.545719, Acc 0.312500
Test:0, Cost:0.71614, ACC:0.70312

Pass 1, Step 0, Cost 0.984149, Acc 0.687500
Test:1, Cost:1.65988, ACC:0.29688

Pass 2, Step 0, Cost 1.028732, Acc 0.562500
Test:2, Cost:0.48792, ACC:0.81250

Pass 3, Step 0, Cost 0.978104, Acc 0.500000
Test:3, Cost:0.54789, ACC:0.79688

Pass 4, Step 0, Cost 0.700373, Acc 0.656250
Test:4, Cost:0.52407, ACC:0.87500

Pass 5, Step 0, Cost 0.735128, Acc 0.687500
Test:5, Cost:0.41733, ACC:0.87500

Pass 6, Step 0, Cost 0.719776, Acc 0.656250
Test:6, Cost:0.35984, ACC:0.81250

Pass 7, Step 0, Cost 0.729553, Acc 0.468750
Test:7, Cost:0.50601, ACC:0.84375

Pass 8, Step 0, Cost 0.679281, Acc 0.750000
Test:8, Cost:0.81121, ACC:0.82812

Pass 9, Step 0, Cost 0.627250, Acc 0.750000
Test:9, Cost:0.67843, ACC:0.78125

Pass 10, Step 0, Cost 0.730999, Acc 0.562500
Test:10, Cost:0.28707, ACC:0.89062

Pass 11, Step 0, Cost 0.584491, Acc 0.781250
Test:11, Cost:0.28830, ACC:0.90625

Pass 12, Step 0, Cost 0.514081, Acc 0.750000
Test:12, Cost:0.24950, ACC:0.92188

Pass 13, Step 0, Cost 0.459601, Acc 0.812500
Test:13, Cost:0.24520, ACC:0.90625

Pass 14, Step 0, Cost 0.370637, Acc 0.812500
Test:14, Cost:0.44119, ACC:0.89062

Pass 15, Step 0, Cost 0.392226, Acc 0.781250
Test:15, Cost:0.21446, ACC:0.90625

Pass 16, Step 0, Cost 0.422558, Acc 0.781250
Test:16, Cost:0.26926, ACC:0.89062

Pass 17, Step 0, Cost 0.484142, Acc 0.875000
Test:17, Cost:0.22030, ACC:0.89062

Pass 18, Step 0, Cost 0.358580, Acc 0.875000
Test:18, Cost:0.19693, ACC:0.92188

Pass 19, Step 0, Cost 0.530168, Acc 0.750000
Test:19, Cost:0.65898, ACC:0.40625
```



```
<Figure size 432x288 with 1 Axes>
训练模型保存完成!
```



准备数据 → 配置网络 → 训练网络 → 模型评估 → 模型预测

Step5.模型预测

下面是预测程序，直接单独运行In[*]就可以。预测主要有四步：第一步配置好预测的环境，第二步准备好要预测的图片，第三步加载预测的模型，把要预测的图片放到模型里进行预测，第四步输出预测的结果

In [21]

```
# coding:utf-8
import paddle.fluid as fluid
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import paddle

# 使用CPU进行训练
place = fluid.CPUPlace()
# 定义一个executor
infer_exe = fluid.Executor(place)
inference_scope = fluid.core.Scope()#要想运行一个网络，需要指明它运行所在的域，确
#选择保存不同的训练模型
params_dirname ="/home/aistudio/data/model_cnn"
#params_dirname ='/home/aistudio/data/model_vgg'

#  (1) 图片预处理
def load_image(path):
    img = paddle.dataset.image.load_and_transform(path,100,100, False).asty
    img = img / 255.0
    return img

infer_imgs = []
infer_path = []
zzy = '/home/aistudio/images/face/zhangziyi/20181206144436.png'
jw = '/home/aistudio/images/face/pengyuyan/20181206161115.png'
pyy = '/home/aistudio/images/face/jiangwen/0acb8d12-f929-11e8-ac67-005056c
infer_path.append((Image.open(zzy), load_image(zzy)))
infer_path.append((Image.open(jw), load_image(jw)))
infer_path.append((Image.open(pyy), load_image(pyy)))
#infer_path.append((Image.open(zzy), load_image(zzy)))

print('infer_imgs的维度: ',np.array(infer_path[0][1]).shape)

#fluid.scope_guard修改全局/默认作用域（scope），运行时中的所有变量都将分配给新的scop
with fluid.scope_guard(inference_scope):
     #获取训练好的模型
    #从指定目录中加载 推理model(inference model)
    [inference_program,# 预测用的program
     feed_target_names,# 是一个str列表，它包含需要在推理 Program 中提供数据的变量
     fetch_targets] = fluid.io.load_inference_model(params_dirname, infer_
    image_and_path = infer_path[2]
    plt.imshow(image_and_path[0])   #根据数组绘制图像
    plt.show()          #显示图像

    #  开始预测
    results = infer_exe.run(
        inference_program,                          #运行预测程序
        feed={feed_target_names[0]: np.array([image_and_path[1]])},#喂入要预
        fetch_list=fetch_targets)                   #得到推测结果
    print('results:',np.argmax(results[0]))

    #  训练数据的标签
    label_list = ["zhangziyi","jiangwen","pengyuyan"]
    print(results)
    print("infer results: %s" % label_list[np.argmax(results[0])])
```

```
infer_imgs的维度:  (3, 100, 100)
results: 1
[array([[0.0281146 , 0.8147827 , 0.14639573, 0.01070699]], dtype=float32)]
infer results: jiangwen
```

```
<Figure size 432x288 with 1 Axes>
```

教师开课申请

了解:

友情链接： 飞桨官网 | 飞桨源码 | 百度开发者中心 | 百度云智学院 | 百度技术学院 | 百度效率云 | 百度点石 | 用户协议 | © 使用百