```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


#导入需要的包
import numpy as np
import paddle as paddle
import paddle.fluid as fluid
from PIL import Image
import matplotlib.pyplot as plt
import os
import math


# In[2]:


BUF_SIZE=256
BATCH_SIZE=64
#用于训练的数据提供器，每次从缓存中随机读取批次大小的数据
train_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.mnist.train(),
                          buf_size=BUF_SIZE),
    batch_size=BATCH_SIZE)
#用于训练的数据提供器，每次从缓存中随机读取批次大小的数据
test_reader = paddle.batch(
    paddle.reader.shuffle(paddle.dataset.mnist.test(),
                          buf_size=BUF_SIZE),
    batch_size=BATCH_SIZE)

#用于打印，查看mnist数据
train_data=paddle.dataset.mnist.train()


# In[3]:


class DistResNet():
    def __init__(self, is_train=True):

        self.is_train = is_train
        self.weight_decay = 1e-4

    def net(self, input, class_dim=10):

        depth = [3, 3, 3]
        num_filters = [16, 32, 32]
```

```
        conv = self.conv_bn_layer(
            input=input, num_filters=16, filter_size=3, act='relu')
        conv = fluid.layers.pool2d(
            input=conv,
            pool_size=3,
            pool_stride=2,
            pool_padding=1,
            pool_type='max')

        for block in range(len(depth)):
            for i in range(depth[block]):
                conv = self.bottleneck_block(
                    input=conv,
                    num_filters=num_filters[block],
                    stride=2 if i == 0 and block != 0 else 1)
                conv = fluid.layers.batch_norm(input=conv, act='relu')
        print(conv.shape)
        pool = fluid.layers.pool2d(
            input=conv, pool_size=4, pool_type='avg', global_pooling=True)
        stdv = 1.0 / math.sqrt(pool.shape[1] * 1.0)
        out = fluid.layers.fc(input=pool,
                              size=class_dim,
                              act="softmax",
                              param_attr=fluid.param_attr.ParamAttr(
                                  initializer=fluid.initializer.Uniform(-stdv,
                                                                        stdv),

regularizer=fluid.regularizer.L2Decay(self.weight_decay)),
                              bias_attr=fluid.ParamAttr(

regularizer=fluid.regularizer.L2Decay(self.weight_decay))
                              )
        return out

    def conv_bn_layer(self,
                      input,
                      num_filters,
                      filter_size,
                      stride=1,
                      groups=1,
                      act=None,
                      bn_init_value=1.0):
        conv = fluid.layers.conv2d(
            input=input,
            num_filters=num_filters,
            filter_size=filter_size,
            stride=stride,
            padding=(filter_size - 1) // 2,
            groups=groups,
            act=None,
            bias_attr=False,

param_attr=fluid.ParamAttr(regularizer=fluid.regularizer.L2Decay(self.weight_decay)))
```

```
        return fluid.layers.batch_norm(
                input=conv, act=act, is_test=not self.is_train,
                param_attr=fluid.ParamAttr(
                    initializer=fluid.initializer.Constant(bn_init_value),
                    regularizer=None))

    def shortcut(self, input, ch_out, stride):
        ch_in = input.shape[1]
        if ch_in != ch_out or stride != 1:
            return self.conv_bn_layer(input, ch_out, 1, stride)
        else:
            return input

    def bottleneck_block(self, input, num_filters, stride):
        conv0 = self.conv_bn_layer(
            input=input, num_filters=num_filters, filter_size=1, act='relu')
        conv1 = self.conv_bn_layer(
            input=conv0,
            num_filters=num_filters,
            filter_size=3,
            stride=stride,
            act='relu')
        conv2 = self.conv_bn_layer(
            input=conv1, num_filters=num_filters * 4, filter_size=1, act=None,
bn_init_value=0.0)

        short = self.shortcut(input, num_filters * 4, stride)

        return fluid.layers.elementwise_add(x=short, y=conv2, act='relu')



# In[4]:


# 定义输入输出层
image = fluid.layers.data(name='image', shape=[1, 28, 28], dtype='float32')#单通道,
28*28像素值
label = fluid.layers.data(name='label', shape=[1], dtype='int64')           #图片标签


# In[5]:


# 获取分类器
model = DistResNet()
out = model.net(input=image, class_dim=10)


# In[6]:


# 获取损失函数和准确率函数
```

```
cost = fluid.layers.cross_entropy(input=out, label=label)  #使用交叉熵损失函数,描述真实样本标签和预测概率之间的差值
avg_cost = fluid.layers.mean(cost)
acc = fluid.layers.accuracy(input=out, label=label)


# In[7]:


# 定义优化方法
optimizer = fluid.optimizer.AdamOptimizer(learning_rate=2e-4)   #使用Adam算法进行优化
opts = optimizer.minimize(avg_cost)


# In[8]:


# 定义一个使用CPU的解析器
place = fluid.CPUPlace()
exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())

feeder = fluid.DataFeeder(place=place, feed_list=[image, label])


# In[9]:


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()


# In[10]:


EPOCH_NUM=4  # 调参 训练轮数
model_save_dir = "/home/aistudio/data/hand.inference.model"
for pass_id in range(EPOCH_NUM):
    # 进行训练
    for batch_id, data in enumerate(train_reader()):                           #遍历train_reader
```

```
        train_cost, train_acc = exe.run(program=fluid.default_main_program(),#运行主程序
                                        feed=feeder.feed(data),                 #给模型喂入
数据
                                        fetch_list=[avg_cost, acc])             #fetch 误
差、准确率

        all_train_iter=all_train_iter+1
        all_train_iters.append(all_train_iter)
        all_train_costs.append(train_cost[0])
        all_train_accs.append(train_acc[0])

        # 每100个batch打印一次信息  误差、准确率
        if batch_id % 100 == 0:
            print('Pass:%d, Batch:%d, Cost:%0.5f, Accuracy:%0.5f' %
                  (pass_id, batch_id, train_cost[0], train_acc[0]))

    # 进行测试
    test_accs = []
    test_costs = []
    #每训练一轮 进行一次测试
    for batch_id, data in enumerate(test_reader()):                             #遍历
test_reader
        test_cost, test_acc = exe.run(program=fluid.default_main_program(), #执行训练程序
                                      feed=feeder.feed(data),                   #喂入数据
                                      fetch_list=[avg_cost, acc])               #fetch 误
差、准确率
        test_accs.append(test_acc[0])                                           #每个batch的
准确率
        test_costs.append(test_cost[0])                                         #每个batch的
误差



    # 求测试结果的平均值
    test_cost = (sum(test_costs) / len(test_costs))                             #每轮的平均误
差
    test_acc = (sum(test_accs) / len(test_accs))                                #每轮的平均准
确率
    print('Test:%d, Cost:%0.5f, Accuracy:%0.5f' % (pass_id, test_cost, test_acc))

    #保存模型
    # 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)
    print ('save models to %s' % (model_save_dir))
    fluid.io.save_inference_model(model_save_dir,  #保存推理model的路径
                                  ['image'],       #推理 (inference) 需要 feed 的数据
                                  [out],       #保存推理 (inference) 结果的 Variables
                                  exe)             #executor 保存 inference model
draw_train_process("training",all_train_iters,all_train_costs,all_train_accs,"trainning
cost","trainning acc")
```

```
#

# In[11]:


def load_image(file):
    im = Image.open(file).convert('L')                          #将RGB转化为灰度图像，L代表灰
度图像，像素值在0~255之间
    im = im.resize((28, 28), Image.ANTIALIAS)                   #resize image with high-
quality 图像大小为28*28

    im = np.array(im).reshape(1, 1, 28, 28).astype(np.float32)#返回新形状的数组,把它变成一个
numpy 数组以匹配数据馈送格式。
    #print(im)
    im = im / 255.0 * 2.0 - 1.0                                 #归一化到【-1~1】之间
    return im


# In[12]:


infer_exe = fluid.Executor(place)
#声明一个新的作用域
inference_scope = fluid.core.Scope()


# In[13]:


#运行时中的所有变量都将分配给新的scope
with fluid.scope_guard(inference_scope):
    #获取训练好的模型
    #从指定目录中加载模型
    [inference_program,                                                 #推理Program
     feed_target_names,                                                 #是一个str列表，它包含
需要在推理 Program 中提供数据的变量的名称。
     fetch_targets] = fluid.io.load_inference_model(model_save_dir,#fetch_targets: 是一
个列表，从中我们可以得到推断结果。model_save_dir：模型保存的路径
                                                    infer_exe)      #infer_exe: 运行
inference model的 executor
    infer_path='/home/aistudio/work/data5286/infer_3.png'
    raw = Image.open(infer_path)
    img = load_image(infer_path)

    results = infer_exe.run(program=inference_program,          #运行推测程序
                    feed={feed_target_names[0]: img},           #喂入要预测的img
                    fetch_list=fetch_targets)                   #得到推测结果，
    # 获取概率最大的label
    lab = np.argsort(results)                                   #argsort函数返回的是result
数组值从小到大的索引值
    print("该图片的预测结果的label为: %d" % lab[0][0][-1])      #-1代表读取数组中倒数第一列
    plt.imshow(raw)    #根据数组绘制图像
```

```
    plt.title('pred: '+str(lab[0][0][-1]))
    plt.show()        #显示图像


# In[ ]:
```