西 南 石 油 大 学 实 验 报 告

课 程	神经网络与深度学习导论	**实验项目**	实验二　典型的深度神经网络		**成 绩**	
专业年级	**计算机科学与技术 2022 级**	**学 号**	202231060435	**指导教师**	**郑 津**	
姓 名	**王磊**	**同组姓名**			**日 期**	

一、实验目的

本项目的目标为：学生在理解深度学习基本理论及方法的基础上，具有实现不同任务目标的深度学习模型，并对其进行性能分析的能力。通过本实验环节的训练，使学生进一步熟悉并掌握基于飞桨平台实现典型深度学习模型的方法，加深对深度学习基础理论的理解，强化编程技能，培养较为熟练的动手能力。

支撑专业实践能力培养目标中的专业核心能力 5：具备大数据分析与挖掘能力，掌握各种数据分析技术和手段，对数据进行初步建模，并能利用统计建模和机器学习的基本理论、方法，对数据进行深度分析和产品化开发。

二、实验内容

1. 基于 CNN 的图像分类任务
2. 深度残差网络
3. 基于 RNN 的文本分类任务

三、实验过程

1. 基于 CNN 的图像分类实验代码

```
#导入需要的包


import  paddle
import  paddle.fluid  as  fluid
import  numpy  as  np
from  PIL  import  Image
import  sys
```

```
from  multiprocessing  import  cpu_count
import  matplotlib.pyplot  as  plt
import  os
paddle.enable_static()

def  unpickle(file):
        import  pickle
        with  open(file,  'rb')  as  fo:
                dict  =  pickle.load(fo,  encoding='bytes')
        return  dict

print(unpickle("cifar-10-batches-py/data_batch_1").keys())
print(unpickle("cifar-10-batches-py/test_batch").keys())



def  test_mapper(sample):
        img,  label  =  sample
        #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




def  train_mapper(sample):
        img,  label  =  sample
        #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



def  train_r(  buffered_size=1024):
```

```
        def  reader():
                xs=[]
                ys=[]
                for  i  in  range(1,6):
                        train_dict=unpickle("cifar-10-batches-py/data_batch_%d"
  %  (i,))
                        xs.append(train_dict[b'data'])
                        ys.append(train_dict[b'labels'])

                Xtr  =  np.concatenate(xs)
                Ytr  =  np.concatenate(ys)

                for  (x,y)  in  zip(Xtr,Ytr):
                        yield  x,  int(y)
        return  paddle.reader.xmap_readers(train_mapper,  reader,cpu_count(),
buffered_size)


#  对自定义数据集创建训练集 train 的 reader

def  test_r(  buffered_size=1024):
        def  reader():
                test_dict=unpickle("cifar-10-batches-py/test_batch")
                X=test_dict[b'data']
                Y=test_dict[b'labels']
                for  (x,y)  in  zip(X,Y):
                        yield  x,  int(y)
        return  paddle.reader.xmap_readers(test_mapper,  reader,cpu_count(),  b
uffered_size)
```

```
BATCH_SIZE  =  128
#用于训练的数据提供器
train_reader  =  train_r()
train_reader  =  paddle.batch(
        paddle.reader.shuffle(
                reader=train_reader,buf_size=128*100),
        batch_size=BATCH_SIZE)
#用于测试的数据提供器
test_reader  =  test_r()
test_reader  =  paddle.batch(
        paddle.reader.shuffle(
                reader=test_reader,buf_size=300),
        batch_size=BATCH_SIZE)



def  convolutional_neural_network(img):
        #  第一个卷积-池化层
        conv1=fluid.layers.conv2d(input=img,                #输入图像
                                  num_filters=20,
#卷积核大小
                                  filter_size=5,
  #卷积核数量，它与输出的通道相同
                                  act="relu")
    #激活函数
        pool1  =  fluid.layers.pool2d(
                            input=conv1,
    #输入
                            pool_size=2,
    #池化核大小
                            pool_type='max',                                #
池化类型
```

```
                            pool_stride=2)
  #池化步长
        conv_pool_1  =  fluid.layers.batch_norm(pool1)


        #  第二个卷积-池化层
        conv2=fluid.layers.conv2d(input=conv_pool_1,
                                                        num_filters=50,
                                                        filter_size=5,
                                                        act="relu")
        pool2  =  fluid.layers.pool2d(
                            input=conv2,
                            pool_size=2,
                            pool_type='max',
                            pool_stride=2,
                            global_pooling=False)
        conv_pool_2  =  fluid.layers.batch_norm(pool2)
        #  第三个卷积-池化层
        conv3=fluid.layers.conv2d(input=conv_pool_2,  num_filters=50,  filter_s
ize=5,  act="relu")
        pool3  =  fluid.layers.pool2d(
                            input=conv3,
                            pool_size=2,
                            pool_type='max',
                            pool_stride=2,
                            global_pooling=False)
        #  以 softmax 为激活函数的全连接输出层，10 类数据输出 10 个数字
        prediction  =  fluid.layers.fc(input=pool3,
                                                                    size=
10,
                                                                    act='
softmax')
```

```
        return  prediction



#定义输入数据
data_shape  =  [3,  32,  32]
images  =  fluid.layers.data(name='images',  shape=data_shape,  dtype='float32'
)
label  =  fluid.layers.data(name='label',  shape=[1],  dtype='int64')



#  获取分类器，用cnn进行分类
predict  =    convolutional_neural_network(images)



#  获取损失函数和准确率
cost  =  fluid.layers.cross_entropy(input=predict,  label=label)  #  交叉熵
avg_cost  =  fluid.layers.mean(cost)
             #  计算cost中所有元素的平均值
acc  =  fluid.layers.accuracy(input=predict,  label=label)              #使用输
入和标签计算准确率



#  获取测试程序
test_program  =  fluid.default_main_program().clone(for_test=True)
#  定义优化方法
optimizer  =fluid.optimizer.Adam(learning_rate=0.00001)
optimizer.minimize(avg_cost)
print("完成")



#  定义使用CPU还是GPU,使用CPU时use_cuda  =  False,使用GPU时use_cuda  =  True
```

```
use_cuda = True
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()

exe = fluid.Executor(place)
exe.run(fluid.default_startup_program())



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()



EPOCH_NUM = 20
model_save_dir = "/home/aistudio/work/catdog.inference.model"

for pass_id in range(EPOCH_NUM):
        # 开始训练
        for batch_id, data in enumerate(train_reader()):
                                #遍历 train_reader 的迭代器，并为数据加上索引
batch_id
```

```
                train_cost,train_acc  =  exe.run(program=fluid.default_main_pro
gram(),#运行主程序
                                                        feed=feeder.feed(data)
,                                                   #喂入一个 batch 的数据
                                                        fetch_list=[avg_cost,
 acc])                                              #fetch 均方误差和准确率


                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])

                #每 100 次 batch 打印一次训练、进行一次测试
                if  batch_id  %  20  ==  0:

                        print('Pass:%d,  Batch:%d,  Cost:%0.5f,  Accuracy:%0.5f
'  %
                        (pass_id,  batch_id,  train_cost[0],  train_acc[0]))


        #  开始测试
        test_costs  =  []
                                                        #测试的损失值
        test_accs  =  []
                                                          #测试的准确率
        for  batch_id,  data  in  enumerate(test_reader()):
                test_cost,  test_acc  =  exe.run(program=test_program,
                        #执行训练程序
                                                                fee
d=feeder.feed(data),                                #喂入数据
```

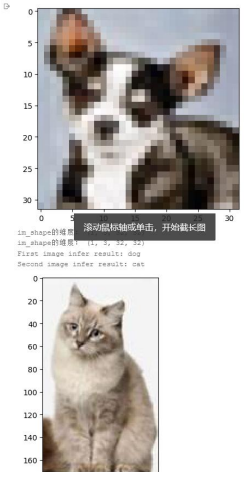
```
                                                                    fet
ch_list=[avg_cost,  acc])                          #fetch  误差、准确率
                test_costs.append(test_cost[0])
                                                #记录每个 batch 的误差
                test_accs.append(test_acc[0])
                                                  #记录每个 batch 的准确率

        #  求测试结果的平均值
        test_cost  =  (sum(test_costs)  /  len(test_costs))
                                #计算误差平均值（误差和/误差的个数）
        test_acc  =  (sum(test_accs)  /  len(test_accs))
                                    #计算准确率平均值（  准确率的和/准确率的个数）
        print('Test:%d,  Cost:%0.5f,  ACC:%0.5f'  %  (pass_id,  test_cost,  tes
t_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,
                                                        ['images'],
                                                        [predict],
                                                        exe)
print('训练模型保存完成！')
draw_train_process("training",all_train_iters,all_train_costs,all_train_accs,"t
rainning  cost","trainning  acc")

infer_exe  =  fluid.Executor(place)
inference_scope  =  fluid.core.Scope()
```

```
def  load_image(file):
                #打开图片
                im  =  Image.open(file)
                #将图片调整为跟训练数据一样的大
小    32*32,                                    设定 ANTIALIAS，即抗锯
齿.resize 是缩放
                im  =  im.resize((32,  32),  Image.ANTIALIAS)
                #建立图片矩阵  类型为 float32
                im  =  np.array(im).astype(np.float32)
                #矩阵转置
                im  =  im.transpose((2,  0,  1))

                #将像素值从【0-255】转换为【0-1】
                im  =  im  /  255.0
                #print(im)
                im  =  np.expand_dims(im,  axis=0)
                #  保持和之前输入 image 维度一致
                print('im_shape 的维度：',im.shape)
                return  im



with  fluid.scope_guard(inference_scope):
        #  从指定目录中加载推理 model(inference  model)
        [inference_program,    #  预测用的  program
          feed_target_names,    #  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  Variable  列表，用于获取推断结果

                          infer_exe)              #  infer_exe:  运
行  inference  model  的  executor
```

```
        #  加载两张图片
        infer_path  =  'work/dog.png'
        infer1_path  =  'work/cat.jpg'

        img  =  Image.open(infer_path)    #  打开第一张图片

        #  展示第一张图片
        plt.imshow(img)
        plt.show()

        #  加载两张图片的  numpy  数组形式
        img  =  load_image(infer_path)      #  加载第一张图片
        img1  =  load_image(infer1_path)    #  加载第二张图片

        #  运行推理程序
        results1  =  infer_exe.run(inference_program,
          #  运行预测程序
                                                    feed={feed_target_nam
es[0]:  img},      #  喂入要预测的第一张  img
                                                    fetch_list=fetch_targ
ets)                      #  得到第一张图片的推断结果

        results2  =  infer_exe.run(inference_program,
          #  运行预测程序
                                                    feed={feed_target_nam
es[0]:  img1},    #  喂入要预测的第二张  img1
                                                    fetch_list=fetch_targ
ets)                      #  得到第二张图片的推断结果

        #  打印推理结果
```

```
        label_list = [
                "airplane", "automobile", "bird", "cat", "deer", "dog", "
frog", "horse",
                "ship", "truck"
        ]

        # 输出第一张图片的推理结果
        print("First image infer result: %s" % label_list[np.argmax(resul
ts1[0])])

        img1 = Image.open(infer1_path)    # 打开第二张图片

        # 展示第二张图片
        plt.imshow(img1)
        plt.show()

        # 输出第二张图片的推理结果
        print("Second image infer result: %s" % label_list[np.argmax(resu
lts2[0])])
```

2. 基于 CNN 的图像分类实验结果（截图）



3. 深度残差网络实验代码

```
#一、导入相关库
import  os
import  paddle
import  paddle.vision.transforms  as  T
import  numpy  as  np
from  PIL  import  Image
import  paddle
import  paddle.nn.functional  as  F
import  cv2
from  sklearn.utils  import  shuffle

#二、读取数据
import  io
import  os
from  PIL  import  Image

data_path  =  'work/food-11/'    #  设置初始文件地址
character_folders  =  os.listdir(data_path)    #  查看地址下文件夹
print('character_folders:{}'.format(character_folders))

#  每次运行前删除 txt，重新新建标签列表
if(os.path.exists('work/food-11/training.txt')):    #  判断有无文件
        os.remove('work/food-11/training.txt')    #  删除文件
if(os.path.exists('work/food-11/validation.txt')):
        os.remove('work/food-11/validation.txt')
if(os.path.exists('work/food-11/testing.txt')):
        os.remove('work/food-11/testing.txt')

#  三、数据预处理
```

```
#  数据集共有三个资料夹，分别为 training、validation  以及  testing。这三个文件夹里直接存放着照片，照片名称格式为  [类别]_[编号].jpg，例如  3_100.jpg  即为类别  3  的照片（编号不重要），每个文件夹里都有 11 类。
#  对这些样本进行一个标注处理,最终生成 train.txt/valid.txt/test.txt 三个数据标注文件。

#  数据集根目录
DATA_ROOT  =  'work/food-11'

#  训练集与验证集标注生成函数
def  generate_annotation(mode):
        #  建立标注文件
        with  open('{}/{}.txt'.format(DATA_ROOT,  mode),  'w')  as  f:
                #  对应每个用途的数据文件夹，train/valid/test
                train_dir  =  '{}/{}'.format(DATA_ROOT,  mode)

                #  图像样本所在的路径
                image_path  =  '{}'.format(train_dir)

                #  遍历所有图像
                for  image  in  os.listdir(image_path):
                        #  图像完整路径和名称
                        image_file  =  '{}/{}'.format(image_path,  image)
                        for  k  in  image:
                                if  k=='_':      #  如果图片名称有下划线‘—’
                                        stop  =  image.index(k)      #  下划线所在索引
                                        label_index  =  image[0:stop]  #  image的索引从 0——下划线前的数字为为图片的标签
                                        label_index  =int(label_index)
```

```
                        try:
                                #  验证图片格式是否 ok
                                with  open(image_file,  'rb')  as  f_img:
                                        image  =  Image.open(io.BytesIO(f_img.r
ead()))
                                        image.load()
                                        if  image.mode  ==  'RGB':
                                                f.write('{}\t{}\n'.format(image
_file,  label_index))
                        except:
                                continue


generate_annotation('training')    #  生成训练集标注文件
generate_annotation('validation')    #  生成验证集标注文件

#  测试数据集的特别处理
#  这里需要说明的是由于测试集没有标签，就不能直接进行实例化，即不能直接使用
FoodDataset 函数。
#  因此在标注的时候将其补图片路径补为 0，才能输入网络进行预测。这对结果是没有影
响的，实例化只是对测试集的图片进行处理。(由于测试数据集没有标签，所以只生成其数
据集的路径文件)

#  数据集根目录
DATA_ROOT  =  'work/food-11'
def  generate_annotation(mode):

        with  open('{}/{}.txt'.format(DATA_ROOT,  mode),  'w')  as  f:
                #  对应每个用途的数据文件夹，train/valid/test
                train_dir  =  '{}/{}'.format(DATA_ROOT,  mode)

                #  图像样本所在的路径
```

```
                image_path  =  '{}'.format(train_dir)

                #  遍历所有图像
                for  image  in  os.listdir(image_path):
                        #  图像完整路径和名称
                        image_file  =  '{}/{}'.format(image_path,  image)
                        label_index  =  0      #测试集没有标签，为了后续的预测，讲其标签设置为0

                        try:
                                #  验证图片格式是否ok
                                with  open(image_file,  'rb')  as  f_img:
                                        image  =  Image.open(io.BytesIO(f_img.read()))
                                        image.load()

                                        if  image.mode  ==  'RGB':
                                                f.write('{}\t{}\n'.format(image_file,label_index))
                        except:
                                continue
#  生成测试集
generate_annotation('testing')

#四、  数据集封装处理
#  继承自PaddlePaddle的Dataset类。

import  paddle
import  paddle.vision.transforms  as  T
import  numpy  as  np
from  PIL  import  Image
```

```
__all__ = ['FoodDataset']

# 定义图像的大小
image_shape = [3,100,100]
IMAGE_SIZE = (image_shape[1], image_shape[2]) # [100,100]

class FoodDataset(paddle.io.Dataset):
        # 数据集类的定义
        def __init__(self, mode='training'):
                # 初始化函数
                assert mode in ['training','validation', 'testing'], 'mode
  is one of train, valid, test.'
                self.data = []
                with open('work/food-11/{}.txt'.format(mode)) as f:
                        for line in f. readlines():
                                info = line.strip().split('\t')
                                if len(info) > 1:
                                        self.data.append([info[0].strip(), inf
o[1].strip()])
                if mode == 'training':
                        self.transforms = T.Compose([
                                T.Resize((256,256)),
                                T.RandomCrop(IMAGE_SIZE),
                                #随机裁剪大小[100,100]
                                T.RandomRotation(15),
                                T.RandomHorizontalFlip(0.5),#随机水平翻转
                                T.RandomVerticalFlip(0.5),
                                #随机垂直翻转
                                T.ToTensor(),
```

```
                                T.Normalize(mean=[0.485,  0.456,  0.406],  std=
[0.229,  0.2224,0.225])#图像归一化
                        ])
                else:
                        self.transforms  =  T.Compose([
                                T.Resize((256,256)),  #  图像大小修改
                                T.RandomCrop(IMAGE_SIZE),  #  随机裁剪
                                T.ToTensor(),  #  数据的格式转换和标准化
HWC=>CHW
                                T.Normalize(mean=[0.485,  0.456,  0.406],  std=
[0.229,  0.2224,  0.225])#图像归一化
                        ])

        def  __getitem__(self,  index):
                #  根据索引获取单个样本

                image_file,  label  =  self.data[index]
                image  =  Image.open(image_file)
                if  image.mode  !=  'RGB':
                        image  =  image.convert('RGB')
                image  =  self.transforms(image)
                return  image,  np.array(label,  dtype='int64')

        def  __len__(self):
        #  获取样本总数
                return  len(self.data)



#  实例化数据集类
#  根据所使用的数据集需求实例化数据集类，并查看总样本量。
training_dataset  =  FoodDataset(mode='training')
```

```
validation_dataset  =  FoodDataset(mode='validation')


print('训练数据集：{}张;  验证数据集：{}张
'.format(len(training_dataset),len(validation_dataset)))


#  实例化测试集类
#  根据所使用的数据集需求实例化测试集类，并查看测试总样本量。
testing_dataset  =  FoodDataset(mode='testing')
print('测试数据集样本量：{}张'.format(len(testing_dataset)))


#五、搭建模型
#  继承 paddle.nn.Layer 类，用于搭建模型


#构建模型
class  Residual(paddle.nn.Layer):
        def  __init__(self,  in_channel,  out_channel,  use_conv1x1=False,  str
ide=1):
                super(Residual,  self).__init__()
                self.conv1  =  paddle.nn.Conv2D(in_channel,  out_channel,  kern
el_size=3,  padding=1,  stride=stride)
                self.conv2  =  paddle.nn.Conv2D(out_channel,  out_channel,  ker
nel_size=3,  padding=1)
                if  use_conv1x1:  #使用 1x1 卷积核
                        self.conv3  =  paddle.nn.Conv2D(in_channel,  out_channe
l,  kernel_size=1,  stride=stride)
                else:
                        self.conv3  =  None
                self.batchNorm1  =  paddle.nn.BatchNorm2D(out_channel)
                self.batchNorm2  =  paddle.nn.BatchNorm2D(out_channel)


        def  forward(self,  x):
```

```
                y  =  F.relu(self.batchNorm1(self.conv1(x)))
                y  =  self.batchNorm2(self.conv2(y))
                if  self.conv3:
                        x  =  self.conv3(x)
                out  =  F.relu(y+x)  #核心代码
                return  out


def  ResNetBlock(in_channel,  out_channel,  nuim_layers,  is_first=False):
        if  is_first:
                assert  in_channel  ==  out_channel
        block_list  =  []
        for  i  in  range(nuim_layers):
                if  i  ==  0  and  not  is_first:
                        block_list.append(Residual(in_channel,  out_channel,use
_conv1x1=True,  stride=2))
                else:
                        block_list.append(Residual(out_channel,  out_channel))
        resNetBlock  =  paddle.nn.Sequential(*block_list)  #用*可以把 list 列表
展开为元素
        return  resNetBlock



class  ResNetModel(paddle.nn.Layer):
        def  __init__(self):
                super(ResNetModel,  self).__init__()
                self.b1  =  paddle.nn.Sequential(
                                        paddle.nn.Conv2D(3,64,kernel_size=7,  s
tride=2,  padding=3),
                                        paddle.nn.BatchNorm2D(64),
                                        paddle.nn.ReLU(),
```

```
                                        paddle.nn.MaxPool2D(kernel_size=3,  str
ide=2,  padding=1))
                self.b2  =  ResNetBlock(64,  64,  2,  is_first=True)
                self.b3  =  ResNetBlock(64,  128,  2)
                self.b4  =  ResNetBlock(128,  256,  2)
                self.b5  =  ResNetBlock(256,  512,  2)
                self.AvgPool  =  paddle.nn.AvgPool2D(2)
                self.flatten  =  paddle.nn.Flatten()
                self.Linear  =  paddle.nn.Linear(2048,  11)
        def  forward(self,  x):
                x  =  self.b1(x)
                x  =  self.b2(x)
                x  =  self.b3(x)
                x  =  self.b4(x)
                x  =  self.b5(x)
                x  =  self.AvgPool(x)
                x  =  self.flatten(x)
                x  =  self.Linear(x)
                return  x



#  声明模型
network  =  ResNetModel()
#  可视化模型
paddle.summary(network,(-1,3,100,100))



#六、训练模型
#  实例化模型
inputs  =  paddle.static.InputSpec(shape=[None,  3,  100,  100],  name='inputs'
)
```

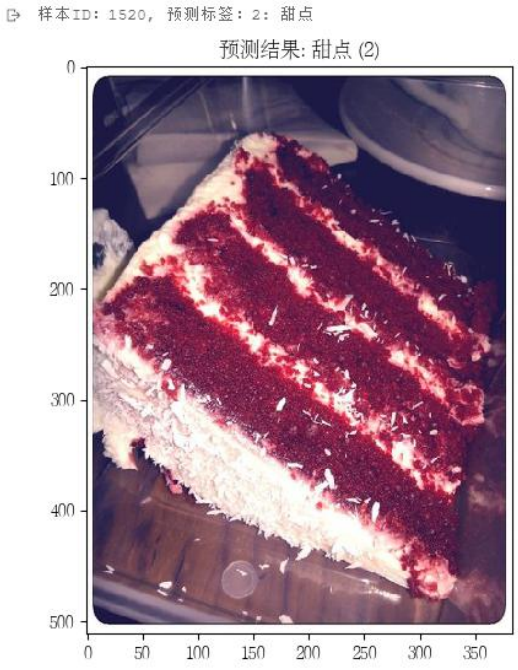
```
labels = paddle.static.InputSpec(shape=[None, 11], name='labels')
model = paddle.Model(network,inputs,labels)

# 余弦退火
scheduler = paddle.optimizer.lr.CosineAnnealingDecay(
        learning_rate=0.001,    # 初始学习率
        T_max=10,    # 学习率周期（即每隔 10 轮重置一次）
        eta_min=0.0001    # 最小学习率
)

# AdamW 优化器
optim = paddle.optimizer.AdamW(learning_rate=scheduler, parameters=model.par
ameters(), weight_decay=0.01)

# 或 Momentum-SGD 优化器
# optim = paddle.optimizer.Momentum(learning_rate=scheduler, parameters=mod
el.parameters(), momentum=0.9)

# 配置模型
model.prepare(
        optim,
        paddle.nn.CrossEntropyLoss(),
        paddle.metric.Accuracy()
        )

visualdl = paddle.callbacks.VisualDL(log_dir='visualdl_log')

# 模型训练与评估
model.fit(
                training_dataset,    # 训练数据集
                validation_dataset,      # 评估数据集
```

```
                #epochs=10,                 #  训练的总轮次
                epochs=10,                  #  训练的总轮次
                batch_size=128,     #  训练使用的批大小
                verbose=1,                  #  日志展示形式
                callbacks=[visualdl])     #  设置可视化


#  模型评估
model.evaluate(validation_dataset,  batch_size=128,  verbose=1)


#  保存模型
model.save('./finetuning/food-11',
                        training=True)     #  保存模型，文件夹路径
为  './finetuning/food-11'

print("模型已保存至  './finetuning/food-11'")



#八、测试
from  paddle.static  import  InputSpec

#  网络结构示例化
network  =    ResNetModel()
#  模型封装
model_2  =  paddle.Model(network,  inputs=[InputSpec(shape=[-1,3,100,100],  dty
pe='float64',  name='image')])

#  训练好的模型加载
model_2.load('finetuning/food-11.pdparams')
```

```
#  模型配置
model_2.prepare()

#  执行预测
result  =  model_2.predict(testing_dataset)



#  测试样例预测结果与可视化输出
import  matplotlib.pyplot  as  plt
from  matplotlib  import  rcParams
%matplotlib  inline

#  设置中文字体（这里以  SimHei  为例，需确保系统安装了该字体）
rcParams['font.sans-serif']  =  ['FZSongYi-Z13S']    #  设置中文字体
rcParams['axes.unicode_minus']  =  False      #  解决负号显示问题

#  随机取样本展示,测试数据集样本量：3347
#  样本映射
LABEL_MAP  =  [
                "面包",
                "乳制品",
                "甜点",
                "鸡蛋",
                "油炸食品",
                "肉类",
                "面条 or 意大利面",
                "米饭",
                "海鲜",
                "汤",
                "蔬菜 or 水果",
```

```
        ]


idx  =  1520
idx_str  =  'work/food-11/testing/'+str(idx)+'.jpg'


predict_label  =  np.argmax(result[0][idx])
print('样本 ID：{},  预测标签：
{}:  {}'.format(idx,predict_label,LABEL_MAP[predict_label]))


image  =  Image.open(idx_str)
plt.figure(figsize=(10,6))
plt.imshow(image)
#  plt.title('predict:  {}  {}'.format(predict_label,  LABEL_MAP[predict_label])
)
plt.title(f'预测结果:  {LABEL_MAP[predict_label]}  ({predict_label})')    #  中
文标题
plt.show()
```

4. 深度残差网络实验结果（截图）



5．基于 RNN 的文本分类实验代码

```
#  导入必要的包
import  paddle
import  paddle.dataset.imdb  as  imdb
import  paddle.fluid  as  fluid
import  numpy  as  np
import  os

#  获取数据字典
print("加载数据字典中...")
word_dict  =  imdb.word_dict()
#  获取数据字典长度
dict_dim  =  len(word_dict)
print('完成')

#  获取训练和预测数据
print("加载训练数据中...")
train_reader  =  paddle.batch(paddle.reader.shuffle(imdb.train(word_dict),


                      512),
                                                    batch_size=128)
print("加载测试数据中...")
test_reader  =  paddle.batch(imdb.test(word_dict),
                                                  batch_size=128)
print('完成')

#  定义长短期记忆网络
def  lstm_net(ipt,  input_dim):
        #  以数据的 IDs 作为输入
        emb  =  fluid.layers.embedding(input=ipt,  size=[input_dim,  128],  is_
sparse=True)
```

```
        #  第一个全连接层
        fc1  =  fluid.layers.fc(input=emb,  size=128)
        #  进行一个长短期记忆操作
        lstm1,  _  =  fluid.layers.dynamic_lstm(input=fc1,  #返回：隐藏状态
（hidden  state），LSTM 的神经元状


    size=128)  #size=4*hidden_size
        #  第一个最大序列池操作
        fc2  =  fluid.layers.sequence_pool(input=fc1,  pool_type='max')
        #  第二个最大序列池操作
        lstm2  =  fluid.layers.sequence_pool(input=lstm1,  pool_type='max')
        #  以 softmax 作为全连接的输出层，大小为 2,也就是正负面
        out  =  fluid.layers.fc(input=[fc2,  lstm2],  size=2,  act='softmax')
    return  out


import  paddle
paddle.enable_static()
#  定义输入数据，  lod_level 不为 0 指定输入数据为序列数据
words  =  fluid.layers.data(name='words',  shape=[1],  dtype='int64',  lod_leve
l=1)
label  =  fluid.layers.data(name='label',  shape=[1],  dtype='int64')
#  获取长短期记忆网络
model  =  lstm_net(words,  dict_dim)


#  获取损失函数和准确率
cost  =  fluid.layers.cross_entropy(input=model,  label=label)
avg_cost  =  fluid.layers.mean(cost)
acc  =  fluid.layers.accuracy(input=model,  label=label)


#  获取预测程序
test_program  =  fluid.default_main_program().clone(for_test=True)
```

```
#  定义优化方法
optimizer  =  fluid.optimizer.AdagradOptimizer(learning_rate=0.002)
opt  =  optimizer.minimize(avg_cost)

#  定义使用CPU还是GPU,使用CPU时use_cuda  =  False,使用GPU时use_cuda  =  True
use_cuda  =  False
place  =  fluid.CUDAPlace(0)  if  use_cuda  else  fluid.CPUPlace()
exe  =  fluid.Executor(place)
#  进行参数初始化
exe.run(fluid.default_startup_program())

#  定义输入数据的维度
#  定义数据数据的维度，数据的顺序是一条句子数据对应一个标签
feeder  =  fluid.DataFeeder(place=place,  feed_list=[words,  label])

#  开始训练
for  pass_id  in  range(1):
        #  进行训练
        train_cost  =  10
        for  batch_id,  data  in  enumerate(train_reader()):
          #遍历train_reader迭代器
                train_cost  =  exe.run(program=fluid.default_main_program(),#运
行主程序
                                                        feed=feeder.feed(data)
,                               #喂入一个batch的数据
                                                        fetch_list=[avg_cost])
                                 #fetch均方误差

                if  batch_id  %  40  ==  0:                                    #
每40次batch打印一次训练、进行一次测试
```

```
                    print('Pass:%d, Batch:%d, Cost:%0.5f' % (pass_id,
batch_id, train_cost[0]))
        # 进行测试
        test_costs = []      #测试的损失值
        test_accs = []       #测试的准确率
        for batch_id, data in enumerate(test_reader()):
                test_cost, test_acc = exe.run(program=test_program,


          feed=feeder.feed(data),


            fetch_list=[avg_cost, acc])
                test_costs.append(test_cost[0])
                test_accs.append(test_acc[0])
        # 计算平均预测损失在和准确率
        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, tes
t_acc))
#保存模型
model_save_dir = "/home/aistudio/work/emotionclassify.inference.model"
# 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
                                                                ['words'],
            #推理（inference）需要 feed 的数据
                                                                [model],
                #保存推理（inference）结果的 Variables
                                                                exe)
                  #exe 保存 inference mo
```

```
#  定义预测数据
reviews_str  =  ['read  the  book  forget  the  movie',  'this  is  a  great  m
ovie',  'this  is  very  bad']
#  把每个句子拆成一个个单词
reviews  =  [c.split()  for  c  in  reviews_str]

#  获取结束符号的标签
UNK  =  word_dict['<unk>']
#  获取每句话对应的标签
lod  =  []
for  c  in  reviews:
        #  需要把单词进行字符串编码转换
    lod.append([word_dict.get(words.encode('utf-8'),  UNK)  for  words  in  c])

#  获取每句话的单词数量
base_shape  =  [[len(c)  for  c  in  lod]]

#  生成预测数据
tensor_words  =  fluid.create_lod_tensor(lod,  base_shape,  place)

infer_exe  =  fluid.Executor(place)          #创建推测用的 executor
inference_scope  =  fluid.core.Scope()  #Scope 指定作用域

with  fluid.scope_guard(inference_scope):#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 scope。
        #从指定目录中加载  推理 model(inference  model)
        [inference_program,
                                    #推理的 program
```

```
        feed_target_names,
                                        #str 列表，包含需要在推理 program 中提供数
据的变量名称
        fetch_targets] = fluid.io.load_inference_model(model_save_dir,#fetc
h_targets: 推断结果，model_save_dir:模型训练路径


                                    infer_exe) #infer_exe: 运
行 inference model 的 executor
    results = infer_exe.run(inference_program,
                                        #运行预测程序
                                                    feed={feed_target_names
[0]: tensor_words},#喂入要预测的 x 值
                                                    fetch_list=fetch_target
s)                                                  #得到推测结果
    # 打印每句话的正负面概率
    for i, r in enumerate(results[0]):
        print("\'%s\'的预测结果为：正面概率为：%0.5f，负面概率
为：%0.5f" % (reviews_str[i], r[0], r[1]))
```

6．基于 RNN 的文本分类实验结果（截图）

```
'read the book forget the movie'的预测结果为: 正面概率为: 0.38083，负面概率为: 0.61917
'this is a great movie'的预测结果为: 正面概率为: 0.38522，负面概率为: 0.61478
'this is very bad'的预测结果为: 正面概率为: 0.37640，负面概率为: 0.62360
```

四、延伸实验

在本次实验中的三个项目中任选一个，尝试在现有模型基础上，调整模型参数，改进基于模型的性能。（给出源代码并将结果截图）

选择第三个项目：

调整：

1. 使用学习率调度器：

```
    learning_rate = fluid.layers.exponential_decay(
```

```
            learning_rate=0.002,  decay_steps=1000,  decay_rate=0.96,  staircase=True
    )
    optimizer  =  fluid.optimizer.AdamOptimizer(learning_rate=learning_rate)
```

2. 增加训练轮次到 10 次

3. 增加注意力机制：

在池化层之前加入注意力机制，增强模型对重要单词的关注。

```
#  Attention
        attention  =  fluid.layers.fc(input=lstm1,  size=128,  act="tanh")
        attention_weight  =  fluid.layers.fc(input=attention,  size=1,  act="softmax")
        scaled_attention  =  fluid.layers.elementwise_mul(lstm1,  attention_weight,  axis=0)
        lstm_out  =  fluid.layers.sequence_pool(input=scaled_attention,  pool_type="sum")
```

4. Dropout 正则化：

在 LSTM 和全连接层之间加入 Dropout（如 0.5），减少过拟合风险：

```
    lstm1 = fluid.layers.dropout(lstm1, dropout_prob=0.5)
```

完整代码：

```
#  导入必要的包
import  paddle
import  paddle.dataset.imdb  as  imdb
import  paddle.fluid  as  fluid
import  numpy  as  np
import  os

#  获取数据字典
print("加载数据字典中...")
word_dict  =  imdb.word_dict()
#  获取数据字典长度
```

```
dict_dim = len(word_dict)
print('完成')

# 获取训练和预测数据
print("加载训练数据中...")
train_reader = paddle.batch(paddle.reader.shuffle(imdb.train(word_dict),

                    512),
                                                    batch_size=128)
print("加载测试数据中...")
test_reader = paddle.batch(imdb.test(word_dict),
                                                    batch_size=128)
print('完成')

# 定义长短期记忆网络
def lstm_net(ipt, input_dim):
        emb = fluid.layers.embedding(input=ipt, size=[input_dim, 300], is_
sparse=True)
        fc1 = fluid.layers.fc(input=emb, size=128, act="relu")

        # 进行一个长短期记忆操作
        lstm1, _ = fluid.layers.dynamic_lstm(input=fc1, #返回：隐藏状态
（hidden state），LSTM的神经元状

    size=128) #size=4*hidden_size

        # Attention
        attention = fluid.layers.fc(input=lstm1, size=128, act="tanh")
        attention_weight = fluid.layers.fc(input=attention, size=1, act="so
ftmax")
```

```
        scaled_attention = fluid.layers.elementwise_mul(lstm1, attention_wei
ght, axis=0)
        lstm_out = fluid.layers.sequence_pool(input=scaled_attention, pool_t
ype="sum")

        # Dropout
        lstm_out = fluid.layers.dropout(lstm_out, dropout_prob=0.5)

        # 输出层
        out = fluid.layers.fc(input=lstm_out, size=2, act="softmax")
        return out

import paddle
paddle.enable_static()
# 定义输入数据， lod_level 不为 0 指定输入数据为序列数据
words = fluid.layers.data(name='words', shape=[1], dtype='int64', lod_leve
l=1)
label = fluid.layers.data(name='label', shape=[1], dtype='int64')
# 获取长短期记忆网络
model = lstm_net(words, dict_dim)

# 获取损失函数和准确率
cost = fluid.layers.cross_entropy(input=model, label=label)
avg_cost = fluid.layers.mean(cost)
acc = fluid.layers.accuracy(input=model, label=label)

# 获取预测程序
test_program = fluid.default_main_program().clone(for_test=True)

# 定义优化方法
learning_rate = fluid.layers.exponential_decay(
```

```
        learning_rate=0.002,  decay_steps=1000,  decay_rate=0.96,  staircase=Tr
ue
)
optimizer  =  fluid.optimizer.AdamOptimizer(learning_rate=learning_rate)
opt  =  optimizer.minimize(avg_cost)

#  定义使用CPU还是GPU,使用CPU时use_cuda  =  False,使用GPU时use_cuda  =  True
use_cuda  =  True
place  =  fluid.CUDAPlace(0)  if  use_cuda  else  fluid.CPUPlace()
exe  =  fluid.Executor(place)
#  进行参数初始化
exe.run(fluid.default_startup_program())

#  定义输入数据的维度
#  定义数据数据的维度，数据的顺序是一条句子数据对应一个标签
feeder  =  fluid.DataFeeder(place=place,  feed_list=[words,  label])

#  开始训练
for  pass_id  in  range(10):
        #  进行训练
        train_cost  =  10
        for  batch_id,  data  in  enumerate(train_reader()):
          #遍历train_reader迭代器
                train_cost  =  exe.run(program=fluid.default_main_program(),#运
行主程序
                                                        feed=feeder.feed(data)
,                               #喂入一个batch的数据
                                                        fetch_list=[avg_cost])
                                 #fetch均方误差

                if  batch_id  %  40  ==  0:                                 #
```

```
每 40 次 batch 打印一次训练、进行一次测试
                        print('Pass:%d,  Batch:%d,  Cost:%0.5f'  %  (pass_id,
batch_id,  train_cost[0]))
        #  进行测试
        test_costs  =  []       #测试的损失值
        test_accs  =  []         #测试的准确率
        for  batch_id,  data  in  enumerate(test_reader()):
                test_cost,  test_acc  =  exe.run(program=test_program,


          feed=feeder.feed(data),


             fetch_list=[avg_cost,  acc])
                test_costs.append(test_cost[0])
                test_accs.append(test_acc[0])
        #  计算平均预测损失在和准确率
        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,  tes
t_acc))
#保存模型
model_save_dir  =  "/home/aistudio/work/emotionclassify.inference.model"
#  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
                                                                ['words'],
            #推理（inference）需要  feed  的数据
                                                                [model],
                #保存推理（inference）结果的  Variables
                                                                exe)
```

```
                #exe  保存  inference  mo


#  定义预测数据
reviews_str  =  ['read  the  book  forget  the  movie',  'this  is  a  great  m
ovie',  'this  is  very  bad']
#  把每个句子拆成一个个单词
reviews  =  [c.split()  for  c  in  reviews_str]


#  获取结束符号的标签
UNK  =  word_dict['<unk>']
#  获取每句话对应的标签
lod  =  []
for  c  in  reviews:
        #  需要把单词进行字符串编码转换
lod.append([word_dict.get(words.encode('utf-8'),  UNK)  for  words  in  c])


#  获取每句话的单词数量
base_shape  =  [[len(c)  for  c  in  lod]]


#  生成预测数据
tensor_words  =  fluid.create_lod_tensor(lod,  base_shape,  place)


infer_exe  =  fluid.Executor(place)          #创建推测用的 executor
inference_scope  =  fluid.core.Scope()  #Scope 指定作用域


with  fluid.scope_guard(inference_scope):#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 scope。
        #从指定目录中加载  推理 model(inference  model)
        [inference_program,
                                    #推理的 program
          feed_target_names,
```

```
                                        #str 列表，包含需要在推理 program 中提供数
据的变量名称
          fetch_targets] = fluid.io.load_inference_model(model_save_dir,#fetc
h_targets: 推断结果，model_save_dir:模型训练路径


                                                    infer_exe) #infer_exe: 运
行 inference model 的 executor
        results = infer_exe.run(inference_program,
                                        #运行预测程序
                                                    feed={feed_target_names
[0]: tensor_words},#喂入要预测的 x 值
                                                    fetch_list=fetch_target
s)                                                  #得到推测结果
        # 打印每句话的正负面概率
        for i, r in enumerate(results[0]):
                print("\'%s\'的预测结果为：正面概率为：%0.5f，负面概率
为：%0.5f" % (reviews_str[i], r[0], r[1]))
```

运行结果：

```
'read the book forget the movie'的预测结果为：正面概率为：0.47315，负面概率为：0.52685
'this is a great movie'的预测结果为：正面概率为：0.47846，负面概率为：0.52154
'this is very bad'的预测结果为：正面概率为：0.47827，负面概率为：0.52173
```