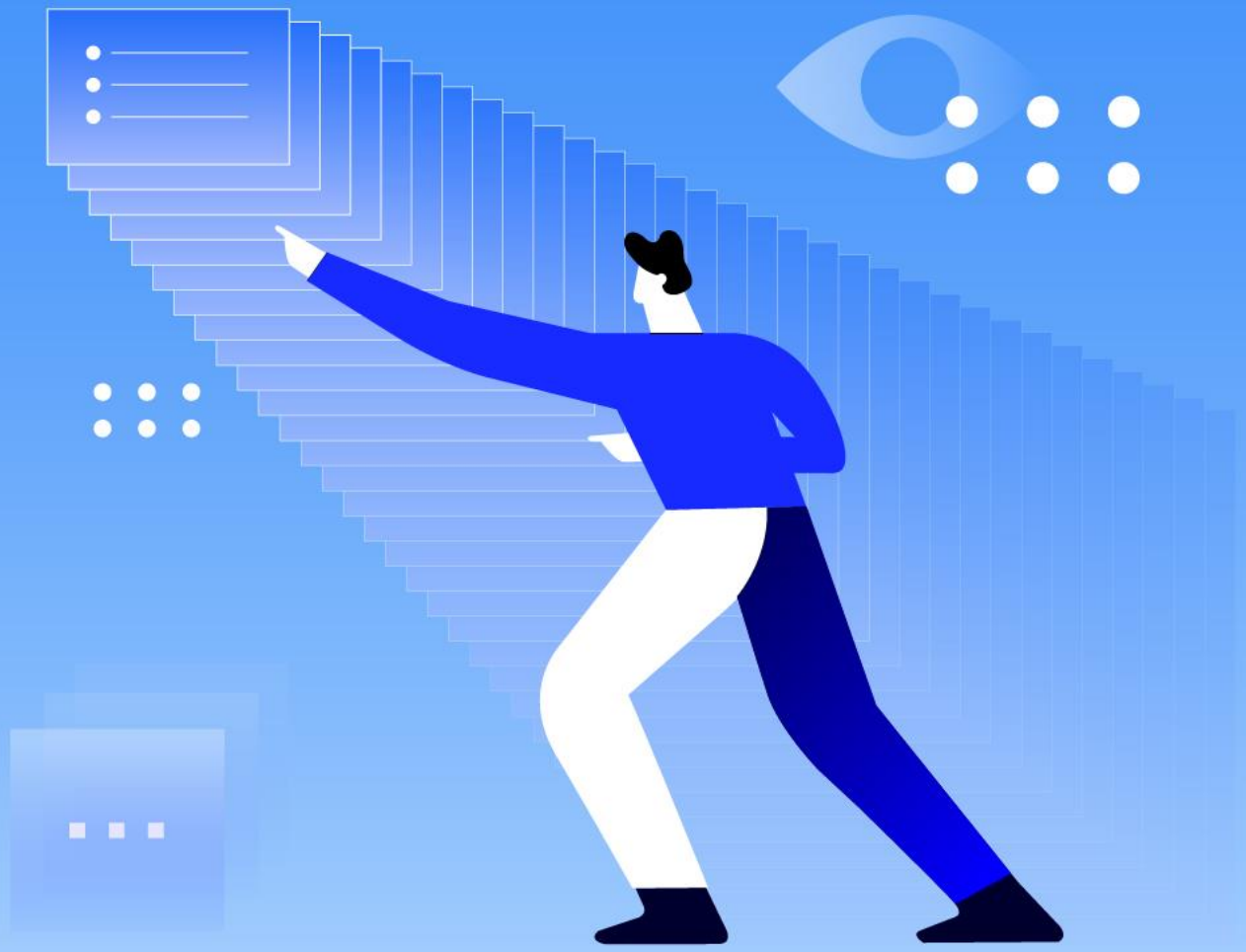
美食图片分类



任务介绍

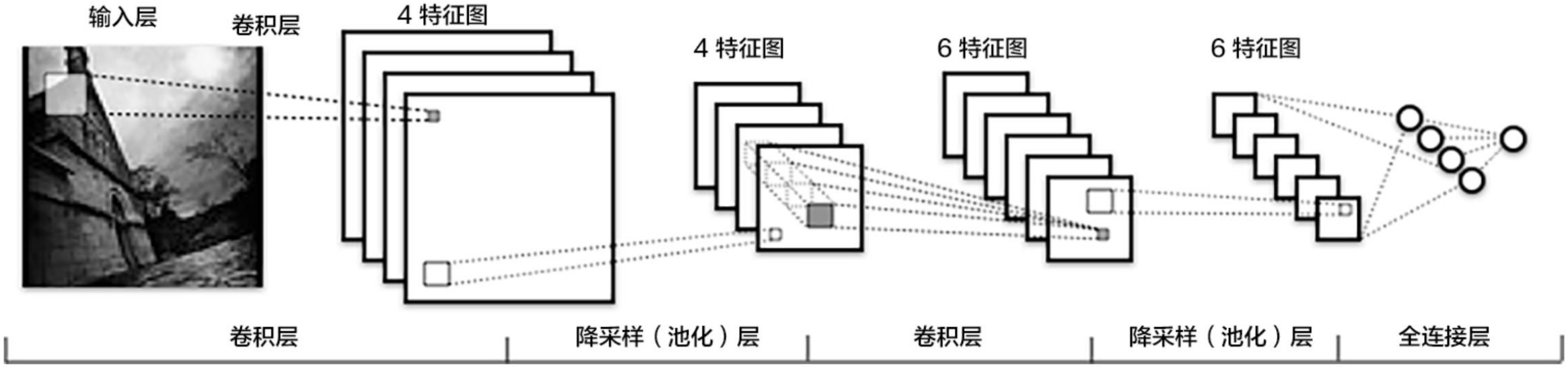
- 任务描述：美食图片分类，如何根据据美食图像的视觉内容为图像赋予一个语义类别（例如，苹果派、牛肉等）是美食图像分类的目标，本次实践将使用卷积神经网络实现美食图片分类案例。
- 实践平台：百度AI实训平台-**AI Studio**
- 实践环境：Python3.7，PaddlePaddle1.8.0

$f($  $)=$ “beef_carpaccio”

模型选择

➢ 卷积神经网络（CNN）

卷积神经网络（Convolution Neural Network，简称CNN），CNN 其实可以看作 DNN 的一种特殊形式。它跟传统 DNN 标志性的区别在于两点，Convolution Kernel 以及 Pooling。



数据集介绍

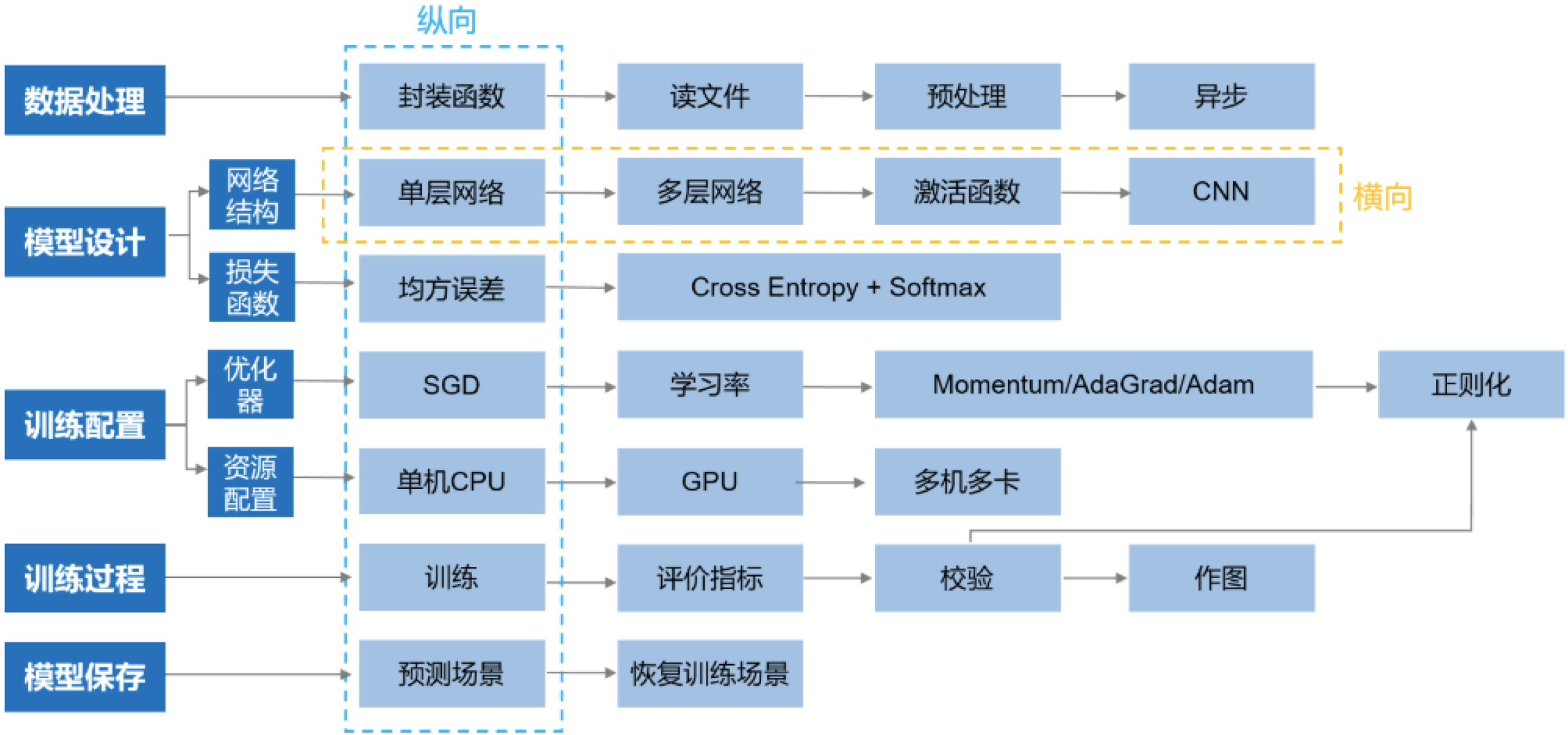
- 数据集：文件名为foods.zip，解压后有五个文件夹，apple_pie、ribs、baklava、beef_carpaccio、beef_tartare，分别存储苹果派、肋骨、蜜糖果仁千层酥、卡巴乔牛肉、鞑靼牛肉的图片。

apple_pie

baklava

特别提示：本实践所用数据集均来自互联网，请勿用于商务用途。

整体流程


纵向
横向
数据处理
封装函数
读文件
预处理
异步
模型设计
网络结构
单层网络
多层网络
激活函数
CNN
损失函数
均方误差
Cross Entropy + Softmax
训练配置
优化器
SGD
学习率
Momentum/AdaGrad/Adam
正则化
资源配置
单机CPU
GPU
多机多卡
训练过程
训练
评价指标
校验
作图
模型保存
预测场景
恢复训练场景

数据处理

◆ 加载飞桨和相关类库

```
import os
import zipfile
import random
import json
import paddle
import sys
import numpy as np
from PIL import Image
from PIL import ImageEnhance
import paddle.fluid as fluid
from multiprocessing import cpu_count
import matplotlib.pyplot as plt
```

- os----------→ python的模块，可使用该模块对操作系统进行操作
- json--------→ python标准库中的json模块，提供json数据处理功能
- numpy------→ python基本库，用于科学计算
- paddle.fluid-→ PaddlePaddle深度学习框架
- PIL----------→ python第三方图像处理库
- matplotlib--→ python的绘图库 pyplot:matplotlib的绘图框架

数据处理

- 解压数据

  将原始数据集解压到指定目录

- 划分训练集与验证集、乱序、生成数据列表

（1）遍历每个类别下的每张图片，按照一定比例划分训练集和验证集

（2）训练集和验证集乱序后，分别写入train.txt和eval.txt

```
random.shuffle(eval_list)
with open(eval_list_path, 'a') as f:
    for eval_image in eval_list:
        f.write(eval_image)

random.shuffle(trainer_list)
with open(train_list_path, 'a') as f2:
    for train_image in trainer_list:
        f2.write(train_image)
```

```
def unzip_data(src_path,target_path):
    '''
    解压原始数据集，将src_path路径下的zip包解压至target_path目录下
    '''
    if(not os.path.isdir(target_path + "foods")):
        z = zipfile.ZipFile(src_path, 'r')
        z.extractall(path=target_path)
        z.close()
```

```
#读取每个类别
for class_dir in class_dirs:
    if class_dir != ".DS_Store":
        class_dim += 1
        #每个类别的信息
        class_detail_list = {}
        eval_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_list_path  + class_dir
        # 获取所有图片
        img_paths = os.listdir(path)
        for img_path in img_paths:                                  # 遍历文件夹下的每个图片
            name_path = path + '/' + img_path                       # 每张图片的路径
            if class_sum % 8 == 0:                                  # 每8张图片取一个做验证数据
                eval_sum += 1                                       # test_sum为测试数据的数目
                eval_list.append(name_path + "\t%d" % class_label + "\n")
            else:
                trainer_sum += 1
                trainer_list.append(name_path + "\t%d" % class_label + "\n")#trainer_sum测试数据的数目
            class_sum += 1                                          #每类图片的数目
            all_class_images += 1                                   #所有类图片的数目
```

train.txt示列：第一维为图片路径，第二维为图片对应标签

```
/home/aistudio/data/foods/beef_carpaccio/754565.jpg 2
/home/aistudio/data/foods/baklava/3753974.jpg   0
/home/aistudio/data/foods/beef_tartare/2995827.jpg  3
/home/aistudio/data/foods/baby_back_ribs/1174148.jpg    4
/home/aistudio/data/foods/beef_tartare/1720102.jpg  3
```

数据处理

◆ 构造数据提供器

（1）数据处理：读入每张图片，将图片缩放到固定大小，将数据转变成np.array格式，HWC格式转换为CHW格式，并进行归一化

注意：yield为python生成器，目的是为了减少内存占用

（2）构造训练数据提供器和验证数据提供器，每batch_size个数据，组成一个批次

```
def custom_reader(file_list):
    '''
    自定义reader
    '''
    def reader():
        with open(file_list, 'r') as f:
            lines = [line.strip() for line in f]
            for line in lines:
                img_path, lab = line.strip().split('\t')
                img = Image.open(img_path)
                if img.mode != 'RGB':
                    img = img.convert('RGB')
                img = img.resize((64, 64), Image.BILINEAR)
                img = np.array(img).astype('float32')
                img = img.transpose((2, 0, 1))  # HWC to CHW
                img = img/255                   # 像素值归一化
                yield img, int(lab)
    return reader

'''
构造数据提供器
'''
train_reader = paddle.batch(custom_reader(train_list_path),
                            batch_size=batch_size,
                            drop_last=True)
eval_reader = paddle.batch(custom_reader(eval_list_path),
                           batch_size=batch_size,
                           drop_last=True)
```

模型设计

◆ 定义卷积网络

基于飞桨动态图构建卷积网络,使用继承自fluid.dygraph.Layer的类来描述操作

(1)构造函数__init__()

使用前面定义的飞桨提供的卷积池化接口，构造MyCNN网络，对每个网络层进行配置。

✓fluid.dygraph.Conv2D()中重要参数解释:

num_channels：输入图像的通道数

num_fliters：卷积核个数

filter_size：卷积核大小

stride:步长

✓ fluid.dygraph.Pool2D()中重要参数解释:

pool_size：池化核大小　pool_type：池化类型　pool_stride：池化步长

```
#定义网络
class MyCNN(fluid.dygraph.Layer):
    def __init__(self):
        super(MyCNN,self).__init__()
        self.hidden1 = fluid.dygraph.Conv2D(num_channels=3,
                                            num_filters=64,
                                            filter_size=3,
                                            stride=1)
        self.hidden2 = fluid.dygraph.Conv2D(num_channels=64,
                                            num_filters = 128,
                                            filter_size=3,
                                            stride=1)
        self.hidden3 = fluid.dygraph.Pool2D(pool_size=2,
                                            pool_type='max',
                                            pool_stride=2)
        self.hidden4 = fluid.dygraph.Linear(128*30*30,5,act='softmax')
    def forward(self,input):
        x = self.hidden1(input)
        #print(x.shape)
        x = self.hidden2(x)
        #print(x.shape)
        x = self.hidden3(x)
        #print(x.shape)
        x = fluid.layers.reshape(x, shape=[-1, 128*30*30])
        y = self.hidden4(x)
        return y
```

模型设计

◆ 定义卷积网络

基于飞桨动态图构建卷积网络,使用继承自fluid.dygraph.Layer的类来描述操作

(2)实现forward()执行函数

将前面构造函数中配置好的卷积池化层和全连接层组合起来，构造网络。

需要注意的是，从卷积池化层到全连接层之间，要将第五组卷积池化层的输出进行reshape之后，才能作为全连接的输入。

```
#定义网络
class MyCNN(fluid.dygraph.Layer):
    def __init__(self):
        super(MyCNN,self).__init__()
        self.hidden1 = fluid.dygraph.Conv2D(num_channels=3,
                                            num_filters=64,
                                            filter_size=3,
                                            stride=1)
        self.hidden2 = fluid.dygraph.Conv2D(num_channels=64,
                                            num_filters = 128,
                                            filter_size=3,
                                            stride=1)
        self.hidden3 = fluid.dygraph.Pool2D(pool_size=2,
                                            pool_type='max',
                                            pool_stride=2)
        self.hidden4 = fluid.dygraph.Linear(128*30*30,5,act='softmax')
    def forward(self,input):
        x = self.hidden1(input)
        #print(x.shape)
        x = self.hidden2(x)
        #print(x.shape)
        x = self.hidden3(x)
        #print(x.shape)
        x = fluid.layers.reshape(x, shape=[-1, 128*30*30])
        y = self.hidden4(x)
        return y
```

训练过程

◆ 模型训练

(1) fluid.dygraph.guard()

创建一个dygraph运行的上下文，执行上下文中的代码

(2) 创建一个模型cnn,配置优化方法

(3) 需要注意的是，使用fluid.dygraph.to_variable()将numpy格式的数据转换为DyGraph接收的数据

```
'''
模型训练
'''
with fluid.dygraph.guard(place = fluid.CUDAPlace(0)):

    cnn = MyCNN()
    optimizer=fluid.optimizer.AdamOptimizer(learning_rate=train_parameters['learning_strategy']['lr'],
                                            parameter_list=cnn.parameters())
    for epoch_num in range(train_parameters['num_epochs']):
        for batch_id, data in enumerate(train_reader()):
            dy_x_data = np.array([x[0] for x in data]).astype('float32')
            y_data = np.array([x[1] for x in data]).astype('int64')
            y_data = y_data[:, np.newaxis]

            #将Numpy转换为DyGraph接收的输入
            img = fluid.dygraph.to_variable(dy_x_data)
            label = fluid.dygraph.to_variable(y_data)

            out = cnn(img)
            acc=fluid.layers.accuracy(out,label)#计算精度
            loss = fluid.layers.cross_entropy(out, label)
            avg_loss = fluid.layers.mean(loss)

            #使用backward()方法可以执行反向网络
            avg_loss.backward()
            optimizer.minimize(avg_loss)

            #将参数梯度清零以保证下一轮训练的正确性
            cnn.clear_gradients()

            all_train_iter=all_train_iter+train_parameters['train_batch_size']
            all_train_iters.append(all_train_iter)
            all_train_costs.append(loss.numpy()[0])
            all_train_accs.append(acc.numpy()[0])

            if batch_id % 1 == 0:
                print("Loss at epoch {} step {}: {}, acc: {}".format(epoch_num, batch_id, avg_loss.numpy(), acc.numpy()))
    #保存模型参数
    fluid.save_dygraph(cnn.state_dict(), "cnn")
    print("Final loss: {}".format(avg_loss.numpy()))
```

训练过程

◆ 模型训练

(4) 使用backward方法执行反向传播

(5) vgg.clear_gradients()每次训练完后，将参数梯度清零，以保证下一次训练的准确性

(6)训练完成后，调用fluid.save_dygraph()保存模型参数

```
'''
模型训练
'''
with fluid.dygraph.guard(place = fluid.CUDAPlace(0)):

    cnn = MyCNN()
    optimizer=fluid.optimizer.AdamOptimizer(learning_rate=train_parameters['learning_strategy']['lr'],
                                            parameter_list=cnn.parameters())
    for epoch_num in range(train_parameters['num_epochs']):
        for batch_id, data in enumerate(train_reader()):
            dy_x_data = np.array([x[0] for x in data]).astype('float32')
            y_data = np.array([x[1] for x in data]).astype('int64')
            y_data = y_data[:, np.newaxis]

            #将Numpy转换为DyGraph接收的输入
            img = fluid.dygraph.to_variable(dy_x_data)
            label = fluid.dygraph.to_variable(y_data)

            out = cnn(img)
            acc=fluid.layers.accuracy(out,label)#计算精度
            loss = fluid.layers.cross_entropy(out, label)
            avg_loss = fluid.layers.mean(loss)

            #使用backward()方法可以执行反向网络
            avg_loss.backward()
            optimizer.minimize(avg_loss)

            #将参数梯度清零以保证下一轮训练的正确性
            cnn.clear_gradients()

            all_train_iter=all_train_iter+train_parameters['train_batch_size']
            all_train_iters.append(all_train_iter)
            all_train_costs.append(loss.numpy()[0])
            all_train_accs.append(acc.numpy()[0])

            if batch_id % 1 == 0:
                print("Loss at epoch {} step {}: {}, acc: {}".format(epoch_num, batch_id, avg_loss.numpy(), acc.numpy()))
    #保存模型参数
    fluid.save_dygraph(cnn.state_dict(), "cnn")
    print("Final loss: {}".format(avg_loss.numpy()))
```

训练过程

◆ 模型训练

(1) 调用fluid.load_dygraph()加载模型参数

(2) 调用eval()接口来切换到预测模式

●我们默认在fluid.dygraph.guard()上下文中是训练模式，训练模式下DyGraph在运行前向网络的时候会自动求导，添加反向网络

●而在预测时，DyGraph只需要执行前向的预测网络，不需要进行自动求导并执行反向网络。

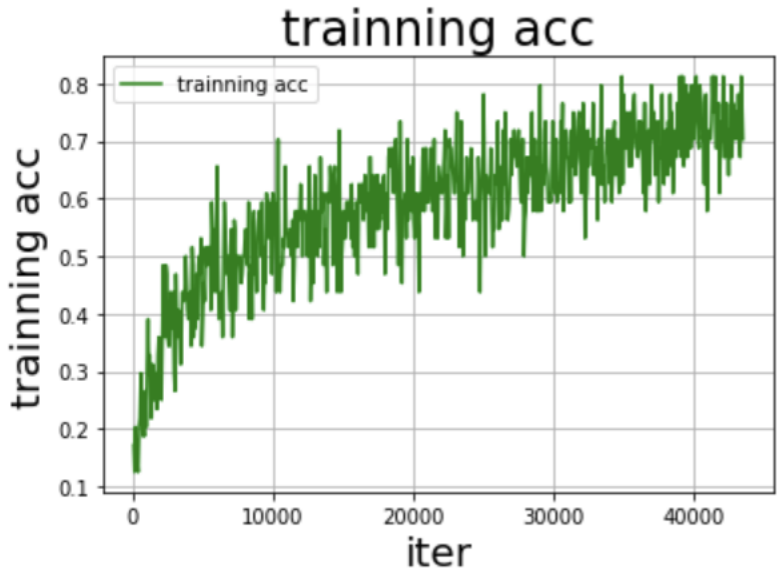
```
'''
模型校验
'''
with fluid.dygraph.guard():
    model, _ = fluid.load_dygraph("cnn")
    cnn = MyCNN()
    cnn.load_dict(model)
    cnn.eval()
    accs = []
    for batch_id, data in enumerate(eval_reader()):
        dy_x_data = np.array([x[0] for x in data]).astype('float32')
        y_data = np.array([x[1] for x in data]).astype('int')
        y_data = y_data[:, np.newaxis]

        img = fluid.dygraph.to_variable(dy_x_data)
        label = fluid.dygraph.to_variable(y_data)

        out = cnn(img)
        acc=fluid.layers.accuracy(out,label)#计算精度
        lab = np.argsort(out.numpy())
        accs.append(acc.numpy()[0])
print(np.mean(accs))
```

训练过程

◆ 观察训练过程中间结果如下：

训练集上的中间结果输出：

```
Loss at epoch 0 step 0: [3.0388918], acc: [0.171875]
Loss at epoch 0 step 1: [5.9353476], acc: [0.125]
Loss at epoch 0 step 2: [11.173545], acc: [0.203125]
Loss at epoch 0 step 3: [8.530522], acc: [0.140625]
Loss at epoch 0 step 4: [9.359183], acc: [0.140625]
Loss at epoch 0 step 5: [8.6943445], acc: [0.125]
....
Loss at epoch 9 step 62: [0.68966687], acc: [0.78125]
Loss at epoch 9 step 63: [0.7169981], acc: [0.71875]
Loss at epoch 9 step 64: [0.7927048], acc: [0.671875]
Loss at epoch 9 step 65: [0.7309673], acc: [0.8125]
Loss at epoch 9 step 66: [0.71572757], acc: [0.765625]
Loss at epoch 9 step 67: [0.77914184], acc: [0.703125]
```



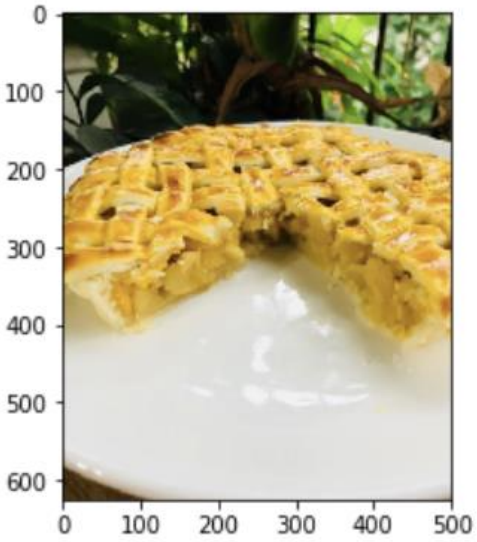
观察到训练过程中模型的误差相对较低，而准确率较高。

模型预测

◆ 模型预测

```
'''
模型预测
'''
with fluid.dygraph.guard():
    model, _ = fluid.dygraph.load_dygraph("cnn")
    cnn = MyCNN()
    cnn.load_dict(model)
    cnn.eval()

    #展示预测图片
    infer_path='work/infer_apple_pie.jpg'
    img = Image.open(infer_path)
    plt.imshow(img)          #根据数组绘制图像
    plt.show()               #显示图像

    #对预测图片进行预处理
    infer_imgs = []
    infer_imgs.append(load_image(infer_path))
    infer_imgs = np.array(infer_imgs)

    for  i in range(len(infer_imgs)):
        data = infer_imgs[i]
        dy_x_data = np.array(data).astype('float32')
        dy_x_data=dy_x_data[np.newaxis,:, : ,:]
        img = fluid.dygraph.to_variable(dy_x_data)
        out = cnn(img)
        lab = np.argmax(out.numpy())  #argmax():返回最大数的索引
        print("第{}个样本,被预测为：{}".format(i+1,label_dic[str(lab)]))

print("结束")
```

```
def load_image(img_path):
    '''
    预测图片预处理
    '''
    img = Image.open(img_path)
    if img.mode != 'RGB':
        img = img.convert('RGB')
    img = img.resize((64, 64), Image.BILINEAR)
    img = np.array(img).astype('float32')
    img = img.transpose((2, 0, 1))  # HWC to CHW
    img = img/255                   # 像素值归一化
    return img
```




```
第1个样本,被预测为：apple_pie
结束
```

感谢聆听

有什么问题吗？