图像分类

背景介绍

图像相比文字能够提供更加生动、容易理解及更具艺术感的信息，是人们转递与交换信息的重要来源。在本教程中，我们专注于图像识别领域的一个重要问题，即图像分类。

图像分类是根据图像的语义信息将不同类别图像区分开来，是计算机视觉中重要的基本问题，也是图像检测、图像分割、物体跟踪、行为分析等其他高层视觉任务的基础。图像分类在很多领域有广泛应用，包括安防领域的人脸识别和智能视频分析等，交通领域的交通场景识别，互联网领域基于内容的图像检索和相册自动归类，医学领域的图像识别等。

一般来说，图像分类通过手工特征或特征学习方法对整个图像进行全部描述，然后使用分类器判别物体类别，因此如何提取图像的特征至关重要。在深度学习算法之前使用较多的是基于词袋(Bag of Words)模型的物体分类方法。词袋方法从自然语言处理中引入，即一句话可以用一个装了词的袋子表示其特征，袋子中的词为句子中的单词、短语或字。对于图像而言，词袋方法需要构建字典。最简单的词袋模型框架可以设计为**底层特征抽取**、**特征编码**、**分类器设计**三个过程。

而基于深度学习的图像分类方法，可以通过有监督或无监督的方式**学习**层次化的特征描述，从而取代了手工设计或选择图像特征的工作。深度学习模型中的卷积神经网络(Convolution Neural Network, CNN)近年来在图像领域取得了惊人的成绩，CNN直接利用图像像素信息作为输入，最大程度上保留了输入图像的所有信息，通过卷积操作进行特征的提取和高层抽象，模型输出直接是图像识别的结果。这种基于"输入-输出"直接端到端的学习方法取得了非常好的效果，得到了广泛的应用。

本教程主要介绍图像分类的深度学习模型，以及如何使用PaddlePaddle训练CNN模型。

效果展示

图像分类包括通用图像分类、细粒度图像分类等。图1展示了通用图像分类效果，即模型可以正确识别图像上的主要物体。

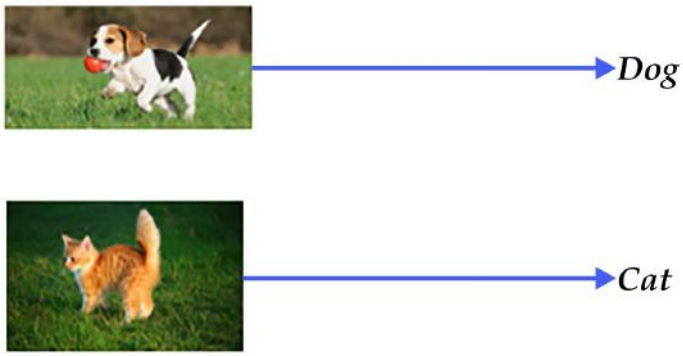

图1. 通用图像分类展示

图2展示了细粒度图像分类-花卉识别的效果，要求模型可以正确识别花的类别。

图2. 细粒度图像分类展示

一个好的模型既要对不同类别识别正确，同时也应该能够对不同视角、光照、背景、变形或部分遮挡的图像正确识别(这里我们统一称作图像扰动)。图3展示了一些图像的扰动，较好的模型会像聪明的人类一样能够正确识别。

图3. 扰动图片展示[22]

模型概览

图像识别领域大量的研究成果都是建立在PASCAL VOC、ImageNet等公开的数据集上，很多图像识别算法通常在这些数据集上进行测试和比较。PASCAL VOC是2005年发起的一个视觉挑战赛，ImageNet是2010年发起的大规模视觉识别竞赛(ILSVRC)的数据集，在本章中我们基于这些竞赛的一些论文介绍图像分类模型。

在2012年之前的传统图像分类方法可以用背景描述中提到的三步完成，但通常完整建立图像识别模型一般包括底层特征学习、特征编码、空间约束、分类器设计、模型融合等几个阶段。
1). **底层特征提取**: 通常从图像中按照固定步长、尺度提取大量局部特征描述。常用的局部特征包括SIFT(Scale-Invariant Feature Transform, 尺度不变特征转换) [1]、HOG(Histogram of Oriented Gradient, 方向梯度直方图) [2]、LBP(Local Bianray Pattern, 局部二值模式) [3] 等，一般也采用多种特征描述子，防止丢失过多的有用信息。
2). **特征编码**: 底层特征中包含了大量冗余与噪声，为了提高特征表达的鲁棒性，需要使用一种特征变换算法对底层特征进行编码，称作特征编码。常用的特征编码包括向量量化编码 [4]、稀疏编码 [5]、局部线性约束编码 [6]、Fisher向量编码 [7] 等。
3). **空间特征约束**: 特征编码之后一般会经过空间特征约束，也称作**特征汇聚**。特征汇聚是指在一个空间范围内，对每一维特征取最大值或者平均值，可以获得一定特征不变形的特征表达。金字塔特征匹配是一种常用的特征聚会方法，这种方法提出将图像均匀分块，在分块内做特征汇聚。
4). **通过分类器分类**: 经过前面步骤之后一张图像可以用一个固定维度的向量进行描述，接下来就是经过分类器对图像进行分类。通常使用的分类器包括SVM(Support Vector Machine, 支持向量机)、随机森林等。而使用核方法的SVM是最为广泛的分类器，在传统图像分类任务上性能很好。

这种方法在PASCAL VOC竞赛中的图像分类算法中被广泛使用 [18]。NEC实验室在ILSVRC2010中采用SIFT和LBP特征，两个非线性编码器以及SVM分类器获得图像分类的冠军 [8]。

Alex Krizhevsky在2012年ILSVRC提出的CNN模型 [9] 取得了历史性的突破，效果大幅度超越传统方法，获得了ILSVRC2012冠军，该模型被称作AlexNet。这也是首次将深度学习用于大规模图像分类中。从AlexNet之后，涌现了一系列CNN模型，不断地在ImageNet上刷新成绩，如图4展示。随着模型变得越来越深以及精妙的结构设计，Top-5的错误率也越来越低，降到了3.5%附近。而在同样的ImageNet数据集上，人眼的辨识错误率大概在5.1%，也就是

目前的深度学习模型的识别能力已经超过了人眼。

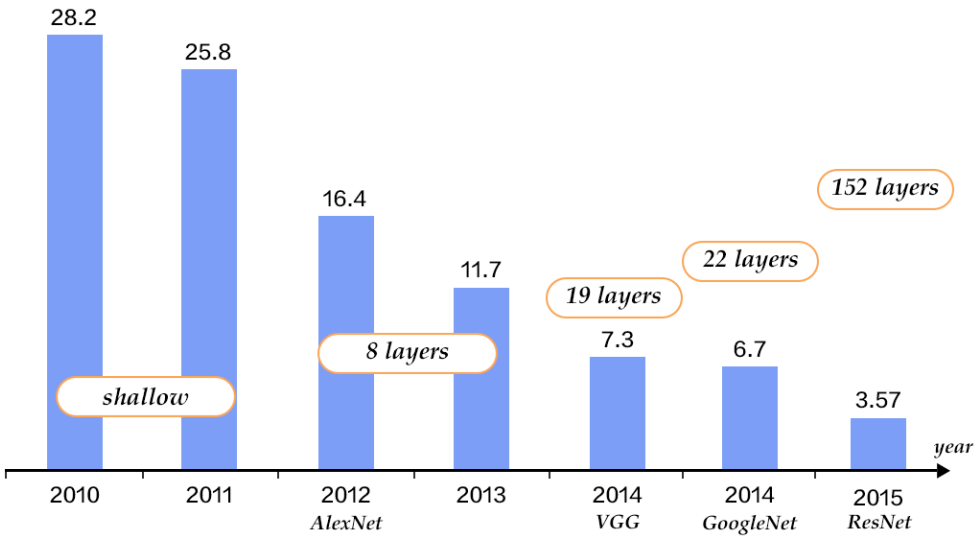

图4. ILSVRC图像分类Top-5错误率

CNN

传统CNN包含卷积层、全连接层等组件，并采用softmax多类别分类器和多类交叉熵损失函数，一个典型的卷积神经网络如图5所示，我们先介绍用来构造CNN的常见组件。

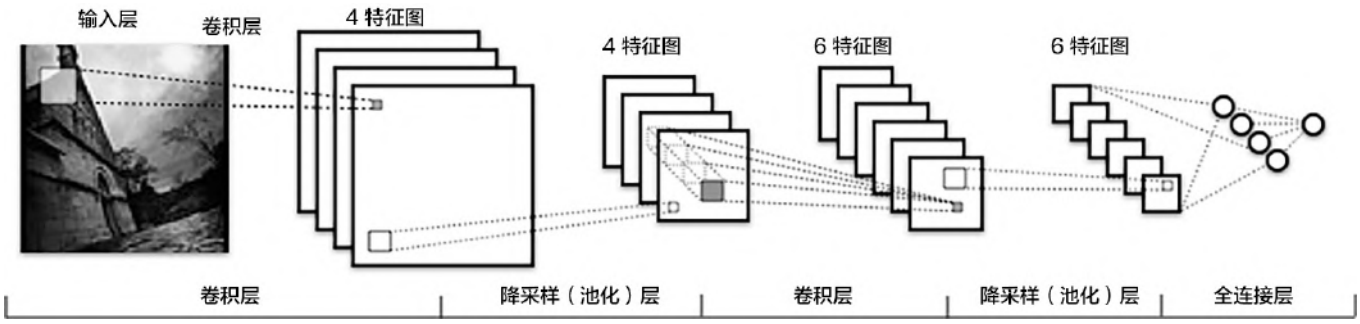

图5. CNN网络示例[20]

- 卷积层(convolution layer): 执行卷积操作提取底层到高层的特征，发掘出图片局部关联性质和空间不变性质。
- 池化层(pooling layer): 执行降采样操作。通过取卷积输出特征图中局部区块的最大值(max-pooling)或者均值(avg-pooling)。降采样也是图像处理中常见的一种操作，可以过滤掉一些不重要的高频信息。
- 全连接层(fully-connected layer，或者fc layer): 输入层到隐藏层的神经元是全部连接的。

- 非线性变化: 卷积层、全连接层后面一般都会接非线性变化层，例如Sigmoid、Tanh、ReLu等来增强网络的表达能力，在CNN里最常使用的为ReLu激活函数。
- Dropout [10] : 在模型训练阶段随机让一些隐层节点权重不工作，提高网络的泛化能力，一定程度上防止过拟合。

另外，在训练过程中由于每层参数不断更新，会导致下一次输入分布发生变化，这样导致训练过程需要精心设计超参数。如2015年Sergey Ioffe和Christian Szegedy提出了Batch Normalization (BN)算法 [14] 中，每个batch对网络中的每一层特征都做归一化，使得每层分布相对稳定。BN算法不仅起到一定的正则作用，而且弱化了一些超参数的设计。经过实验证明，BN算法加速了模型收敛过程，在后来较深的模型中被广泛使用。

接下来我们主要介绍VGG，GoogleNet和ResNet网络结构。

VGG

牛津大学VGG(Visual Geometry Group)组在2014年ILSVRC提出的模型被称作VGG模型 [11] 。该模型相比以往模型进一步加宽和加深了网络结构，它的核心是五组卷积操作，每两组之间做Max-Pooling空间降维。同一组内采用多次连续的3X3卷积，卷积核的数目由较浅组的64增多到最深组的512，同一组内的卷积核数目是一样的。卷积之后接两层全连接层，之后是分类层。由于每组内卷积层的不同，有11、13、16、19层这几种模型，下图展示一个16层的网络结构。VGG模型结构相对简洁，提出之后也有很多文章基于此模型进行研究，如在ImageNet上首次公开超过人眼识别的模型[19]就是借鉴VGG模型的结构。

图6. 基于ImageNet的VGG16模型

GoogleNet

GoogleNet [12] 在2014年ILSVRC的获得了冠军，在介绍该模型之前我们先来了解

NIN(Network in Network)模型 [13] 和Inception模块，因为GoogleNet模型由多组Inception模块组成，模型设计借鉴了NIN的一些思想。

NIN模型主要有两个特点：1) 引入了多层感知卷积网络(Multi-Layer Perceptron Convolution, MLPconv)代替一层线性卷积网络。MLPconv是一个微小的多层卷积网络，即在线性卷积后面增加若干层1x1的卷积，这样可以提取出高度非线性特征。2) 传统的CNN最后几层一般都是全连接层，参数较多。而NIN模型设计最后一层卷积层包含类别维度大小的特征图，然后采用全局均值池化(Avg-Pooling)替代全连接层，得到类别维度大小的向量，再进行分类。这种替代全连接层的方式有利于减少参数。

Inception模块如下图7所示，图(a)是最简单的设计，输出是3个卷积层和一个池化层的特征拼接。这种设计的缺点是池化层不会改变特征通道数，拼接后会导致特征的通道数较大，经过几层这样的模块堆积后，通道数会越来越大，导致参数和计算量也随之增大。为了改善这个缺点，图(b)引入3个1x1卷积层进行降维，所谓的降维就是减少通道数，同时如NIN模型中提到的1x1卷积也可以修正线性特征。

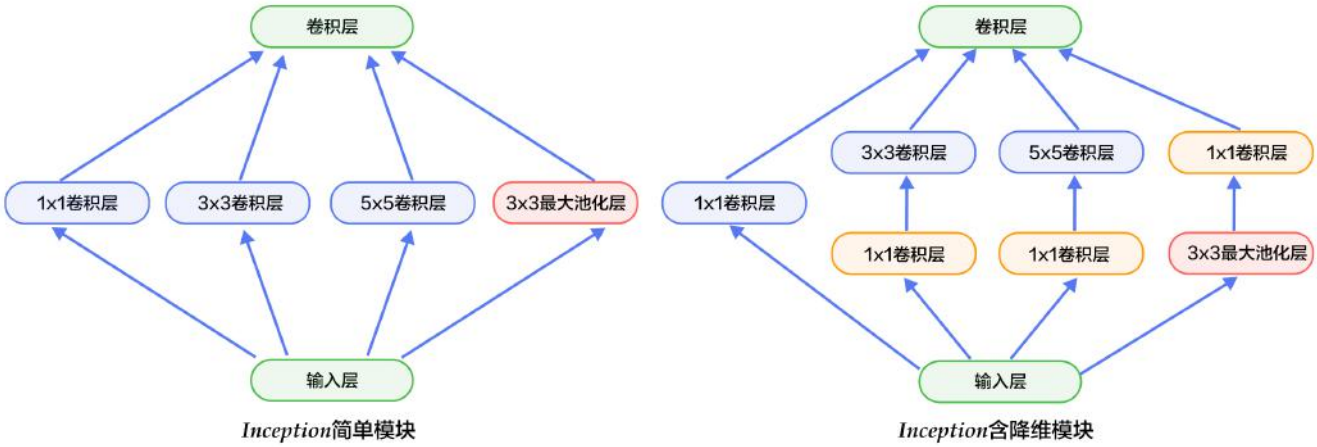

图7. Inception模块

GoogleNet由多组Inception模块堆积而成。另外，在网络最后也没有采用传统的多层全连接层，而是像NIN网络一样采用了均值池化层；但与NIN不同的是，池化层后面接了一层到类别数映射的全连接层。除了这两个特点之外，由于网络中间层特征也很有判别性，GoogleNet在中间层添加了两个辅助分类器，在后向传播中增强梯度并且增强正则化，而整个网络的损失函数是这个三个分类器的损失加权求和。

GoogleNet整体网络结构如图8所示，总共22层网络：开始由3层普通的卷积组成；接下来由三组子网络组成，第一组子网络包含2个Inception模块，第二组包含5个Inception模块，第三

组包含2个Inception模块；然后接均值池化层、全连接层。

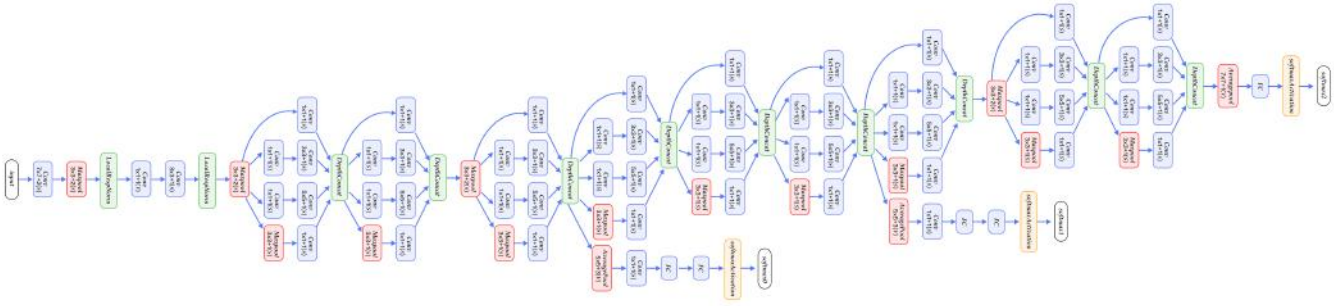

图8. GoogleNet[12]

上面介绍的是GoogleNet第一版模型(称作GoogleNet-v1)。GoogleNet-v2 [14] 引入BN层；GoogleNet-v3 [16] 对一些卷积层做了分解，进一步提高网络非线性能力和加深网络；GoogleNet-v4 [17] 引入下面要讲的ResNet设计思路。从v1到v4每一版的改进都会带来准确度的提升，介于篇幅，这里不再详细介绍v2到v4的结构。

ResNet

ResNet(Residual Network) [15] 是2015年ImageNet图像分类、图像物体定位和图像物体检测比赛的冠军。针对训练卷积神经网络时加深网络导致准确度下降的问题，ResNet提出了采用残差学习。在已有设计思路(BN, 小卷积核，全卷积网络)的基础上，引入了残差模块。每个残差模块包含两条路径，其中一条路径是输入特征的直连通路，另一条路径对该特征做两到三次卷积操作得到该特征的残差，最后再将两条路径上的特征相加。

残差模块如图9所示，左边是基本模块连接方式，由两个输出通道数相同的3x3卷积组成。右边是瓶颈模块(Bottleneck)连接方式，之所以称为瓶颈，是因为上面的1x1卷积用来降维(图示例即256->64)，下面的1x1卷积用来升维(图示例即64->256)，这样中间3x3卷积的输入和输出通道数都较小(图示例即64->64)。

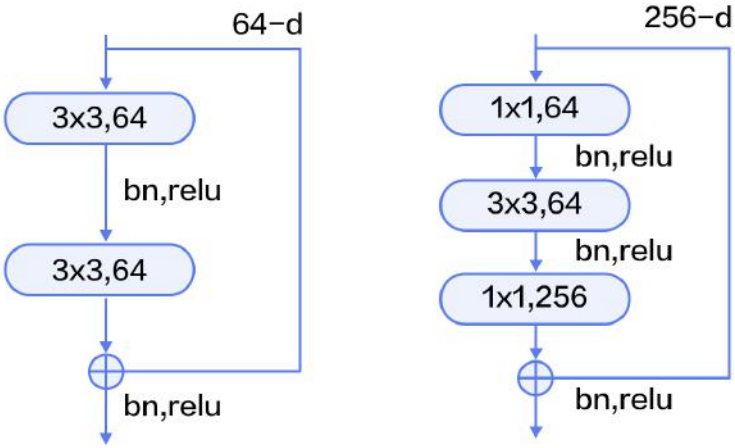

图9. 残差模块

图10展示了50、101、152层网络连接示意图，使用的是瓶颈模块。这三个模型的区别在于每组中残差模块的重复次数不同(见图右上角)。ResNet训练收敛较快，成功的训练了上百乃至近千层的卷积神经网络。

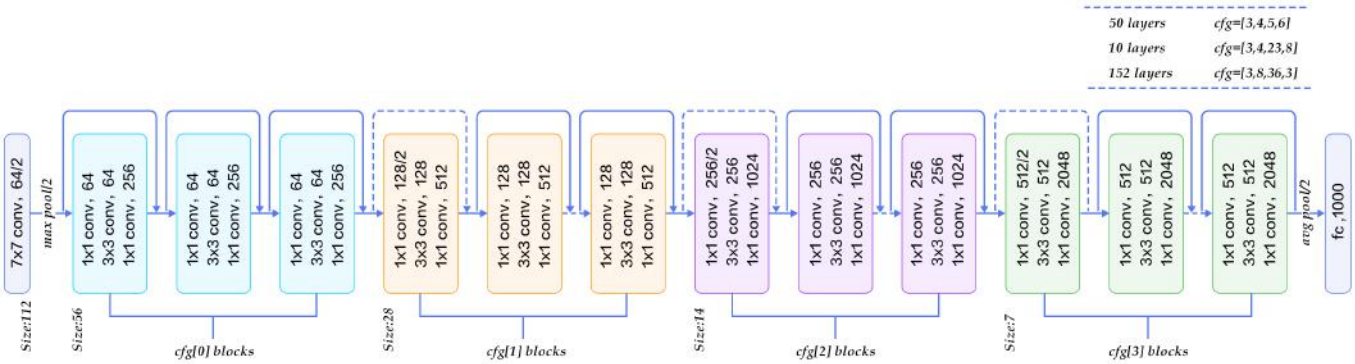

图10. 基于ImageNet的ResNet模型

数据准备

数据介绍与下载

通用图像分类公开的标准数据集常用的有CIFAR、ImageNet、COCO等，常用的细粒度图像分类数据集包括CUB-200-2011、Stanford Dog、Oxford-flowers等。其中ImageNet数据集规模相对较大，如模型概览一章所讲，大量研究成果基于ImageNet。ImageNet数据从2010年来稍有变化，常用的是ImageNet-2012数据集，该数据集包含1000个类别：训练集包含1,281,167张图片，每个类别数据732至1300张不等，验证集包含50,000张图片，平均每

个类别50张图片。

由于ImageNet数据集较大，下载和训练较慢，为了方便大家学习，我们使用CIFAR10数据集。CIFAR10数据集包含60,000张32x32的彩色图片，10个类别，每个类包含6,000张。其中50,000张图片作为训练集，10000张作为测试集。图11从每个类别中随机抽取了10张图片，展示了所有的类别。

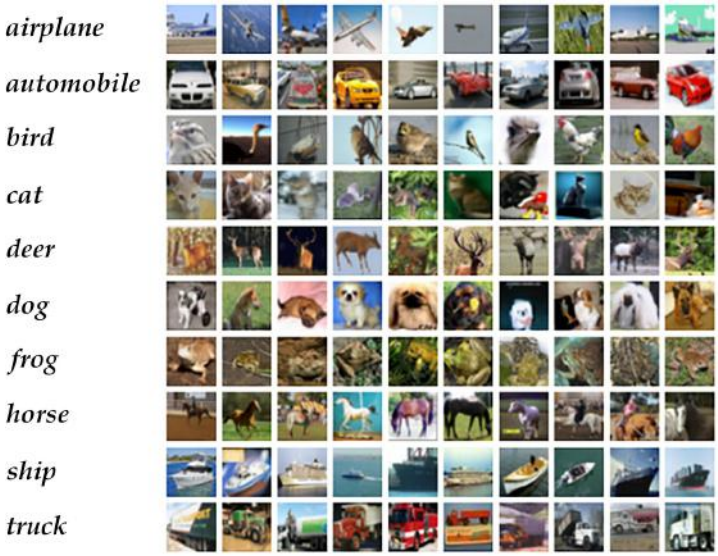

图11. CIFAR10数据集[21]

下面命令用于下载数据和基于训练集计算图像均值，在网络输入前，基于该均值对输入数据做预处理。

```
./data/get_data.sh
```

数据提供给PaddlePaddle

我们使用Python接口传递数据给系统，下面 `dataprovider.py` 针对CIFAR10数据给出了完整示例。

- `initializer` 函数进行dataprovider的初始化，这里加载图像的均值，定义了输入image和label两个字段的类型。
- `process` 函数将数据逐条传输给系统，在图像分类任务里，可以在该函数中完成数据扰动操作，再传输给PaddlePaddle。这里对训练集做随机左右翻转，并将原始图片减去均值后传输给系统。

```python
import numpy as np
import cPickle
from paddle.trainer.PyDataProvider2 import *

def initializer(settings, mean_path, is_train, **kwargs):
  settings.is_train = is_train
  settings.input_size = 3 * 32 * 32
  settings.mean = np.load(mean_path)['mean']
  settings.input_types = {
      'image': dense_vector(settings.input_size),
      'label': integer_value(10)
  }


@provider(init_hook=initializer, cache=CacheType.CACHE_PASS_IN_MEM)
def process(settings, file_list):
  with open(file_list, 'r') as fdata:
      for fname in fdata:
          fo = open(fname.strip(), 'rb')
          batch = cPickle.load(fo)
          fo.close()
          images = batch['data']
          labels = batch['labels']
          for im, lab in zip(images, labels):
              if settings.is_train and np.random.randint(2):
                  im = im[:,:,::-1]
              im = im - settings.mean
              yield {
                  'image': im.astype('float32'),
                  'label': int(lab)
              }
```

模型配置说明

数据定义

在模型配置中，定义通过 `define_py_data_sources2` 函数从 dataprovider 中读入数据， 其中 args 指定均值文件的路径。如果该配置文件用于预测，则不需要数据定义部分。

```
from paddle.trainer_config_helpers import *
```

```
is_predict = get_config_arg("is_predict", bool, False)
if not is_predict:
    define_py_data_sources2(
        train_list='data/train.list',
        test_list='data/test.list',
        module='dataprovider',
        obj='process',
        args={'mean_path': 'data/mean.meta'})
```

算法配置

在模型配置中，通过 `settings` 设置训练使用的优化算法，并指定batch size 、初始学习率、momentum以及L2正则。

```
settings(
    batch_size=128,
    learning_rate=0.1 / 128.0,
    learning_rate_decay_a=0.1,
    learning_rate_decay_b=50000 * 100,
    learning_rate_schedule='discexp',
    learning_method=MomentumOptimizer(0.9),
    regularization=L2Regularization(0.0005 * 128),)
```

通过 `learning_rate_decay_a` (简写a) 、 `learning_rate_decay_b` (简写b) 和 `learning_rate_schedule` 指定学习率调整策略，这里采用离散指数的方式调节学习率，计算公式如下， n 代表已经处理过的累计总样本数，lr_0 即为 `settings` 里设置的 `learning_rate`。

$$lr = lr_0 * a^{\lfloor \frac{n}{b} \rfloor}$$

模型结构

本教程中我们提供了VGG和ResNet两个模型的配置。

VGG

首先介绍VGG模型结构，由于CIFAR10图片大小和数量相比ImageNet数据小很多，因此这里

的模型针对CIFAR10数据做了一定的适配。卷积部分引入了BN和Dropout操作。

1. 定义数据输入及其维度

   网络输入定义为 `data_layer` (数据层)，在图像分类中即为图像像素信息。CIFRAR10是RGB 3通道32x32大小的彩色图，因此输入数据大小为3072(3x32x32)，类别大小为10，即10分类。

```
1.
2.  datadim = 3 * 32 * 32
3.  classdim = 10
4.  data = data_layer(name='image', size=datadim)
```

2. 定义VGG网络核心模块

```
1.  net = vgg_bn_drop(data)
```

   VGG核心模块的输入是数据层，`vgg_bn_drop` 定义了16层VGG结构，每层卷积后面引入BN层和Dropout层，详细的定义如下：

```
1.  def vgg_bn_drop(input, num_channels):
2.      def conv_block(ipt, num_filter, groups, dropouts, num_channels_=
    None):
3.          return img_conv_group(
4.              input=ipt,
5.              num_channels=num_channels_,
6.              pool_size=2,
7.              pool_stride=2,
8.              conv_num_filter=[num_filter] * groups,
9.              conv_filter_size=3,
10.             conv_act=ReluActivation(),
11.             conv_with_batchnorm=True,
12.             conv_batchnorm_drop_rate=dropouts,
13.             pool_type=MaxPooling())
14.
15.     conv1 = conv_block(input, 64, 2, [0.3, 0], 3)
16.     conv2 = conv_block(conv1, 128, 2, [0.4, 0])
17.     conv3 = conv_block(conv2, 256, 3, [0.4, 0.4, 0])
18.     conv4 = conv_block(conv3, 512, 3, [0.4, 0.4, 0])
19.     conv5 = conv_block(conv4, 512, 3, [0.4, 0.4, 0])
20.
21.     drop = dropout_layer(input=conv5, dropout_rate=0.5)
```

```
    fc1 = fc_layer(input=drop, size=512, act=LinearActivation())
    bn = batch_norm_layer(
        input=fc1, act=ReluActivation(), layer_attr=ExtraAttr(drop_r
ate=0.5))
    fc2 = fc_layer(input=bn, size=512, act=LinearActivation())
    return fc2
```

2.1. 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.trainer_config_helpers`中预定义的模块，由若干组`Conv->BN->ReLu->Dropout`和一组`Pooling`组成，

2.2. 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。

2.3. 最后接两层512维的全连接。

3. 定义分类器

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。

```
out = fc_layer(input=net, size=class_num, act=SoftmaxActivation())
```

4. 定义损失函数和网络输出

在有监督训练中需要输入图像对应的类别信息，同样通过`data_layer`来定义。训练中采用多类交叉熵作为损失函数，并作为网络的输出，预测阶段定义网络的输出为分类器得到的概率信息。

```
if not is_predict:
    lbl = data_layer(name="label", size=class_num)
    cost = classification_cost(input=out, label=lbl)
    outputs(cost)
else:
    outputs(out)
```

ResNet

ResNet模型的第1、3、4步和VGG模型相同，这里不再介绍。主要介绍第2步即CIFAR10数据集上ResNet核心模块。

```
net = resnet_cifar10(data, depth=56)
```

先介绍`resnet_cifar10`中的一些基本函数，再介绍网络连接过程。

- `conv_bn_layer`：带BN的卷积层。
- `shortcut`：残差模块的"直连"路径，"直连"实际分两种形式：残差模块输入和输出特征通道数不等时，采用1x1卷积的升维操作；残差模块输入和输出通道相等时，采用直连操作。
- `basicblock`：一个基础残差模块，即图9左边所示，由两组3x3卷积组成的路径和一条"直连"路径组成。
- `bottleneck`：一个瓶颈残差模块，即图9右边所示，由上下1x1卷积和中间3x3卷积组成的路径和一条"直连"路径组成。
- `layer_warp`：一组残差模块，由若干个残差模块堆积而成。每组中第一个残差模块滑动窗口大小与其他可以不同，以用来减少特征图在垂直和水平方向的大小。

```
def conv_bn_layer(input,
                  ch_out,
                  filter_size,
                  stride,
                  padding,
                  active_type=ReluActivation(),
                  ch_in=None):
    tmp = img_conv_layer(
        input=input,
        filter_size=filter_size,
        num_channels=ch_in,
        num_filters=ch_out,
        stride=stride,
        padding=padding,
        act=LinearActivation(),
        bias_attr=False)
    return batch_norm_layer(input=tmp, act=active_type)

def shortcut(ipt, n_in, n_out, stride):
    if n_in != n_out:
        return conv_bn_layer(ipt, n_out, 1, stride, 0,
```

```
    LinearActivation())
    else:
        return ipt

def basicblock(ipt, ch_out, stride):
    ch_in = ipt.num_filters
    tmp = conv_bn_layer(ipt, ch_out, 3, stride, 1)
    tmp = conv_bn_layer(tmp, ch_out, 3, 1, 1, LinearActivation())
    short = shortcut(ipt, ch_in, ch_out, stride)
    return addto_layer(input=[ipt, short], act=ReluActivation())

def bottleneck(ipt, ch_out, stride):
    ch_in = ipt.num_filter
    tmp = conv_bn_layer(ipt, ch_out, 1, stride, 0)
    tmp = conv_bn_layer(tmp, ch_out, 3, 1, 1)
    tmp = conv_bn_layer(tmp, ch_out * 4, 1, 1, 0, LinearActivation())
    short = shortcut(ipt, ch_in, ch_out, stride)
    return addto_layer(input=[ipt, short], act=ReluActivation())

def layer_warp(block_func, ipt, features, count, stride):
    tmp = block_func(ipt, features, stride)
    for i in range(1, count):
        tmp = block_func(tmp, features, 1)
    return tmp
```

`resnet_cifar10` 的连接结构主要有以下几个过程。

1. 底层输入连接一层 `conv_bn_layer`，即带BN的卷积层。
2. 然后连接3组残差模块即下面配置3组 `layer_warp` ，每组采用图 10 左边残差模块组成。
3. 最后对网络做均值池化并返回该层。

```
def resnet_cifar10(ipt, depth=56):
  # depth should be one of 20, 32, 44, 56, 110, 1202
  assert (depth - 2) % 6 == 0
  n = (depth - 2) / 6
  nStages = {16, 64, 128}
  conv1 = conv_bn_layer(ipt,
      ch_in=3,
      ch_out=16,
      filter_size=3,
      stride=1,
      padding=1)
  res1 = layer_warp(basicblock, conv1, 16, n, 1)
```

```
    res2 = layer_warp(basicblock, res1, 32, n, 2)
    res3 = layer_warp(basicblock, res2, 64, n, 2)
    pool = img_pool_layer(input=res3,
                          pool_size=8,
                          stride=1,
                          pool_type=AvgPooling())
    return pool
```

注意：除过第一层卷积层和最后一层全连接层之外，要求三组 `layer_warp` 总的含参层数能够被6整除，即 `resnet_cifar10` 的 depth 要满足 $(depth - 2)$ 。

模型训练

执行脚本 train.sh 进行模型训练， 其中指定配置文件、设备类型、线程个数、总共训练的轮数、模型存储路径等。

```
sh train.sh
```

脚本 `train.sh` 如下：

```
#cfg=models/resnet.py
cfg=models/vgg.py
output=output
log=train.log

paddle train \
    --config=$cfg \
        --use_gpu=true \
        --trainer_count=1 \
        --log_period=100 \
        --num_passes=300 \
        --save_dir=$output \
    2>&1 | tee $log
```

- `--config=$cfg` : 指定配置文件，默认是 `models/vgg.py`。
- `--use_gpu=true` : 指定使用GPU训练，若使用CPU，设置为false。
- `--trainer_count=1` : 指定线程个数或GPU个数。

- `--log_period=100`：指定日志打印的batch间隔。
- `--save_dir=$output`：指定模型存储路径。

一轮训练log示例如下所示，经过1个pass， 训练集上平均error为0.79958 ，测试集上平均error为0.7858 。

```
TrainerInternal.cpp:165]  Batch=300 samples=38400 AvgCost=2.07708 CurrentCost=1.96158 Eval: classification_error_evaluator=0.81151 CurrentEval: classification_error_evaluator=0.789297
TrainerInternal.cpp:181]  Pass=0 Batch=391 samples=50000 AvgCost=2.03348 Eval: classification_error_evaluator=0.79958
Tester.cpp:115]  Test samples=10000 cost=1.99246 Eval: classification_error_evaluator=0.7858
```

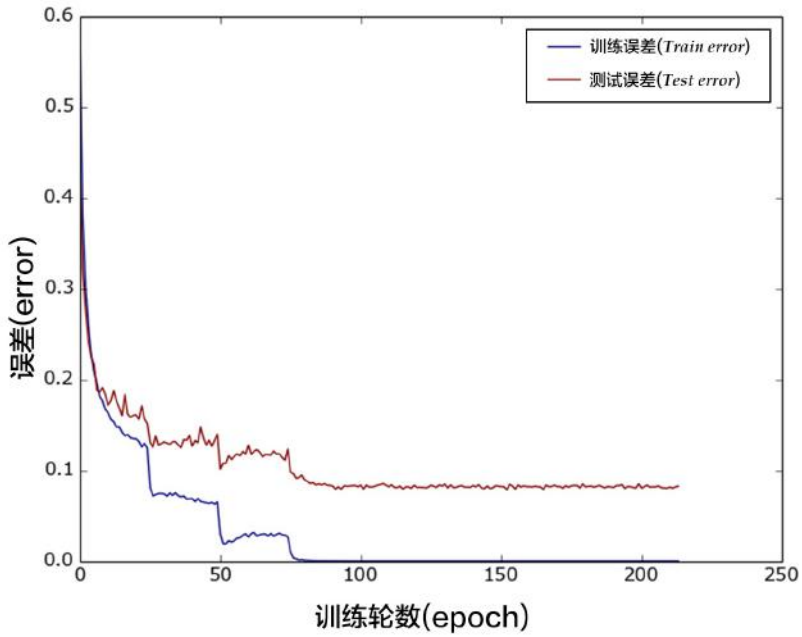

图12. CIFAR10数据集上VGG模型的分类错误率

模型应用

在训练完成后，模型会保存在路径 `output/pass-%05d` 下，例如第300个pass的模型会保存在路径 `output/pass-00299`。 可以使用脚本 `classify.py` 对图片进行预测或提取特征，注意该脚本默认使用模型配置为 `models/vgg.py` ，

预测

可以按照下面方式预测图片的类别，默认使用GPU预测，如果使用CPU预测，在后面加参数`-c`即可。

```
python classify.py --job=predict --model=output/pass-00299 --data=image/dog
.png # -c
```

预测结果为：

```
Label of image/dog.png is: 5
```

特征提取

可以按照下面方式对图片提取特征，和预测使用方式不同的是指定job类型为extract，并需要指定提取的层。`classify.py`默认以第一层卷积特征为例提取特征，并画出了类似图13的可视化图。VGG模型的第一层卷积有64个通道，图13展示了每个通道的灰度图。

```
python classify.py --job=extract --model=output/pass-00299 --data=image/dog
.png # -c
```

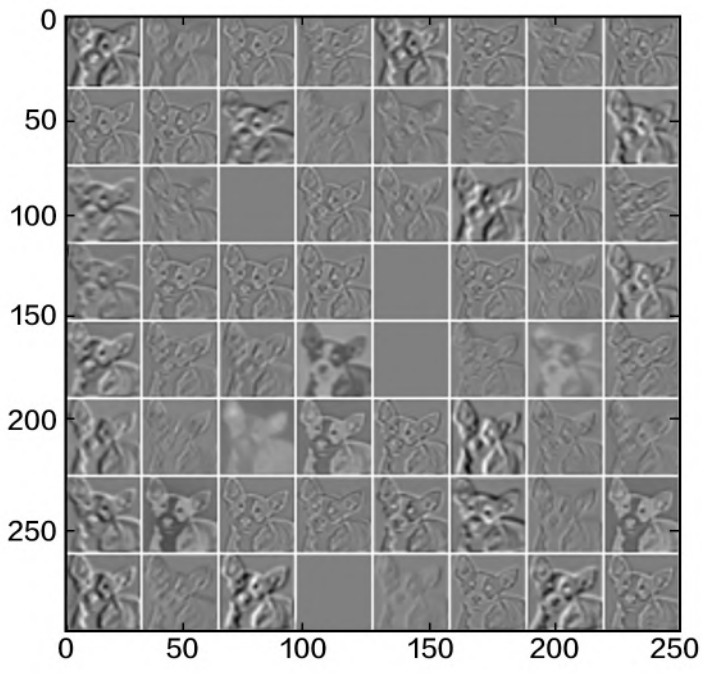

图13. 卷积特征可视化图

总结

传统图像分类方法由多个阶段构成，框架较为复杂，而端到端的CNN模型结构可一步到位，而且大幅度提升了分类准确率。本文我们首先介绍VGG、GoogleNet、ResNet三个经典的模型；然后基于CIFAR10数据集，介绍如何使用PaddlePaddle配置和训练CNN模型，尤其是VGG和ResNet模型；最后介绍如何使用PaddlePaddle的API接口对图片进行预测和特征提取。对于其他数据集比如ImageNet，配置和训练流程是同样的，大家可以自行进行实验。

参考文献

[1] D. G. Lowe, Distinctive image features from scale-invariant keypoints. IJCV, 60(2):91-110, 2004.

[2] N. Dalal, B. Triggs, Histograms of Oriented Gradients for Human Detection, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 2005.

[3] Ahonen, T., Hadid, A., and Pietikinen, M. (2006). Face description with local binary patterns: Application to face recognition. PAMI, 28.

[4] J. Sivic, A. Zisserman, Video Google: A Text Retrieval Approach to Object Matching in Videos, Proc. Ninth Int'l Conf. Computer Vision, pp. 1470-1478, 2003.

[5] B. Olshausen, D. Field, Sparse Coding with an Overcomplete Basis Set: A Strategy Employed by V1?, Vision Research, vol. 37, pp. 3311-3325, 1997.

[6] Wang, J., Yang, J., Yu, K., Lv, F., Huang, T., and Gong, Y. (2010). Locality-constrained Linear Coding for image classification. In CVPR.

[7] Perronnin, F., Sánchez, J., & Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In ECCV (4).

[8] Lin, Y., Lv, F., Cao, L., Zhu, S., Yang, M., Cour, T., Yu, K., and Huang, T. (2011). Large-scale image clas- sification: Fast feature extraction and SVM training. In CVPR.

[9] Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with

deep convolutional neu- ral networks. In NIPS.

[10] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R.R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.

[11] K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman. Return of the Devil in the Details: Delving Deep into Convolutional Nets. BMVC, 2014。

[12] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A., Going deeper with convolutions. In: CVPR. (2015)

[13] Lin, M., Chen, Q., and Yan, S. Network in network. In Proc. ICLR, 2014.

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.

[15] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. CVPR 2016.

[16] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. Rethinking the inception architecture for computer vision. In: CVPR. (2016).

[17] Szegedy, C., Ioffe, S., Vanhoucke, V. Inception-v4, inception-resnet and the impact of residual connections on learning. arXiv:1602.07261 (2016).

[18] Everingham, M., Eslami, S. M. A., Van Gool, L., Williams, C. K. I., Winn, J. and Zisserman, A. The Pascal Visual Object Classes Challenge: A Retrospective. International Journal of Computer Vision, 111(1), 98-136, 2015.

[19] He, K., Zhang, X., Ren, S., and Sun, J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. ArXiv e-prints, February 2015.

[20] http://deeplearning.net/tutorial/lenet.html

[21] https://www.cs.toronto.edu/~kriz/cifar.html

[22] http://cs231n.github.io/classification/