華東理工大學

模式识别大作业

题　　目　多输入多输出图像分类

学　　院　信息科学与工程

专　　业　控制科学与工程

组　　员　李永生 陈旻扬

指导教师　赵海涛

完成日期： 2019 年 12 月 10 日

模式识别作业报告——多输入多输出图像分类

组员：李永生，陈旻扬

经过半个学期的对模式识别课程的学习，在赵海涛老师的辛勤指导下，我们都对模式识别与机器学习方向有了一定的了解，但是理论终究是理论，只有能够把知识熟练运用到实践当中，才能算是真正掌握了这门知识，所以赵海涛老师布置的这个大作业能够很好的锻炼我们理论运用于实际的能力。

训练一个好的卷积神经网络模型进行图像分类不仅需要计算资源还需要很长的时间。特别是模型比较复杂和数据量比较大的时候。普通的电脑动不动就需要训练几天的时间。为了能够快速地训练好自己的图片分类器，我们可以使用别人已经训练好的模型参数，在此基础之上训练我们的模型。这个便属于迁移学习。

多输入多输出的分类与普通的二分类原理相似，唯一区别较大的就是在数据预处理方面，本文所使用的深度学习框架为 tensorflow。

一、 多输入多输出图像分类及数据集简介

多输出分类旨在同时预测一个输入样例的多个离散输出值,其中多个输出可由多种离散变量(如二元变量、名义变量和有序变量)表征。本文所使用的数据集为平时自己收集的服装图片，输入可以分为两类，分别为衣服的颜色和类型，其中衣服的颜色可以分为 3 类：黑蓝红，而衣服的类型又可以分为 4 类：jeans，shoes，dress 和 shirt，总共约三千张图片，我们的数据集就是将颜色和衣服类型随机组合并将它们一一区分，由于数据集收集困难，本文只进行 7 中不同形式的组合，分别是 black_jeans, black_shoes, blue_dress, blue_jeans, blue_shirt, red_dress

和 red_shirt。

二、 数据预处理

2.1 数据读入

首先加载 pathlib 工具包用于数据的导入，这是 tensorflow 官方推荐的加载数据集的工具包，先告诉电脑应该从哪里读取数据，为了增加实验的可靠性，将 7 个文件夹下所有图片放在一起，为了区分图片原来属于哪一个文件夹即属于哪一类，导入后图片标签将加上原来图片的路径，具体如下所示：E:\VScode\multi-output-classification\dataset\blue_shirt\00000133.jpg。将他们都放入列表中。

2.2 数据编码

数据是信息的载体，计算机中的数据是以离散的“0”“1”二进制比特序列方式表示的。为了正确地传输数据，就必须对原始数据进行编码，因为输入种类较多，如果需要对数据继续编码就首先要取出所有分类。例如，对颜色分类就需要先把红黑蓝三种颜色标签取出，分别标号为：0,1,2。衣服类型同理。

2.3 数据增强

一般而言，比较成功的神经网络需要大量的参数，许许多多的神经网路的参数都是数以百万计，而使得这些参数可以正确工作则需要大量的数据进行训练，而实际情况中数据并没有我们想象中的那么多，例如本文就只要不到三千张图片，数据集比较小。数据增强是扩充数据样本规模的一种有效地方法。深度学习是基于大数据的一种方法，我们当前希望数据的规模越大、质量越高越好。模型才能够有着更好的泛化能力，然而实际采集数据的时候，往往很难覆盖掉全部的场景，比如：对于光照条件，在采集图像数据时，我们很难控制光线的比例，因此在训练模型的时候，就需要加入光照变化方面的数据增强。再有一方面就是数据的获取也需要大量的成本，如果能够自动化的生成各种训练

数据，就能做到更好的开源节流。本文主要对图像做了对一化处理，当然仍然有许多的数据增强方法，可以根据需要添加。同时为了更好的管理数据使用 Dataset 对数据进行处理，其中包括将数据集分割成 train 和 val，为了加快速度每次实验的 batch 设为 16，这主要根据电脑性能决定，到这里对图片的预处理大致就完成了。

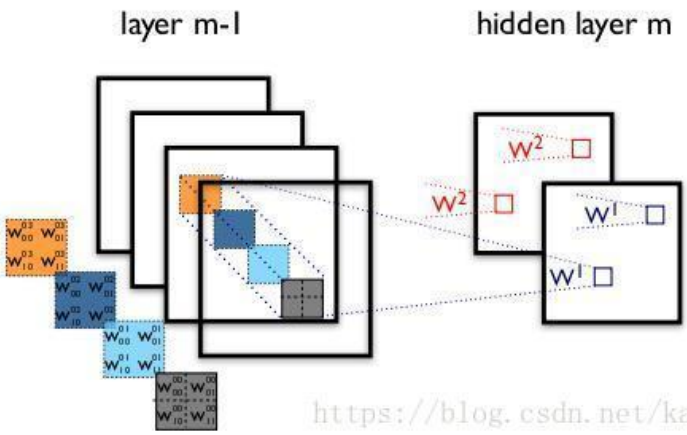
三 模型搭建

3.1 迁移学习

迁移学习是一种机器学习方法，就是把为任务 A 开发的模型作为初始点，重新使用在为任务 B 开发模型的过程中。深度学习中在计算机视觉任务和自然语言处理任务中将预训练的模型作为新模型的起点是一种常用的方法，通常这些预训练的模型在开发神经网络的时候已经消耗了巨大的时间资源和计算资源，迁移学习可以将已习得的强大技能迁移到相关的的问题上。本文选择的模型为 MobileNetV2，之所以会选择 MobileNetV2 是因为它是一个轻量化的模型且分类效果良好。

3.2 MobileNetV2

3.2.1 卷积

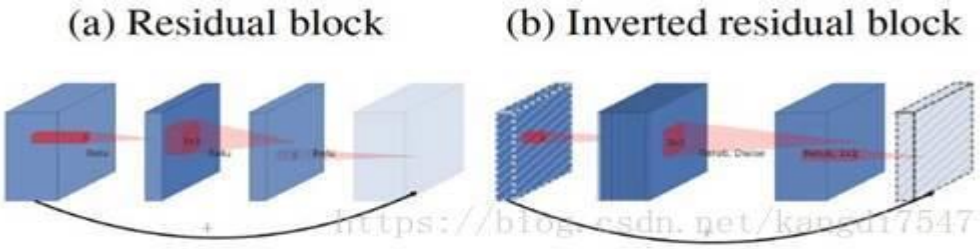


标准的卷积过程可以看上图，一个 2×2 的卷积核在卷积时，对应图像区域

中的所有通道均被同时考虑。相当于数学中的内积形式，卷积核与对应区域所有数据相乘。

3.2.2 Inverted Residual Block

MobileNetV1 没很好的利用 Residual Connection，而 Residual Connection 通常情况下总是好的，所以 MobileNet V2 加上。先看看原始的 ResNet Block 长什么样，下图左边：



先用 1x1 降通道过 ReLU，再 3x3 空间卷积过 ReLU，再用 1x1 卷积过 ReLU 恢复通道，并和输入相加。之所以要 1x1 卷积降通道，是为了减少计算量，不然中间的 3x3 空间卷积计算量太大。所以 Residual block 是沙漏形，两边宽中间窄。但是，现在我们中间的 3x3 卷积变为了 Depthwise 的了，计算量很少了，所以通道可以多一点，效果更好，所以通过 1x1 卷积先提升通道数，再 Depthwise 的 3x3 空间卷积，再用 1x1 卷积降低维度。两端的通道数都很小，所以 1x1 卷积升通道或降通道计算量都并不大，而中间通道数虽然多，但是 Depthwise 的卷积计算量也不大。作者称之为 Inverted Residual Block，两边窄中间宽，像柳叶，较小的计算量得到较好的性能。

3.3.3 网络结构

Input	Operator
$224^2 \times 3$	conv2d
$112^2 \times 32$	bottleneck
$112^2 \times 16$	bottleneck
$56^2 \times 24$	bottleneck
$28^2 \times 32$	bottleneck
$14^2 \times 64$	bottleneck
$14^2 \times 96$	bottleneck
$7^2 \times 160$	bottleneck
$7^2 \times 320$	conv2d 1x1
$7^2 \times 1280$	avgpool 7x7
$1 \times 1 \times 1280$	conv2d 1x1

MobileNetV2 网络结构如上图所示。图中的 bottleneck 就是上面介绍的残差模块。

四 模型评估

为了提高模型的性能需要给模型添加优化。

4.1 Adam

对于神经网络的反向传播依然使用梯度下降算法，这里选择的是最广泛使用的 Adam。Adam 是一种可以替代传统随机梯度下降（SGD）过程的一阶优化算法，它能基于训练数据迭代地更新神经网络权重。Adam 算法和传统的随机梯度下降不同。随机梯度下降保持单一的学习率更新所有的权重，学习率在训练过程中并不会改变。而 Adam 通过计算梯度的一阶矩估计和二阶矩估计而为不同的参数设计独立的自适应性学习率 Adam 算法的提出者描述其为两种随机梯度下降扩展式的优点集合，即：适应性梯度算法（AdaGrad）为每一个参数保留一个学习率以提升在稀疏梯度（即自然语言和计算机视觉问题）上的性能。均方根传播（RMSProp）基于权重梯度最近量级的均值为每一个参数适应性地保留学习率。这意味着算法在非稳态和在线问题上有很有优秀的性能。

Adam 算法同时获得了 AdaGrad 和 RMSProp 算法的优点。Adam 不仅如 RMSProp 算法那样基于一阶矩均值计算适应性参数学习率，它同时还充分利用了梯度的二阶矩均值（即有偏方差/uncentered variance）。具体来说，算法计算了梯度的指数移动均值（exponential moving average），超参数 beta1 和 beta2 控制了这些移动均值的衰减率。移动均值的初始值和 beta1、beta2 值接近于 1（推荐值），因此矩估计的偏差接近于 0。该偏差通过首先计算带偏差的估计而后计算偏差修正后的估计而得到提升。

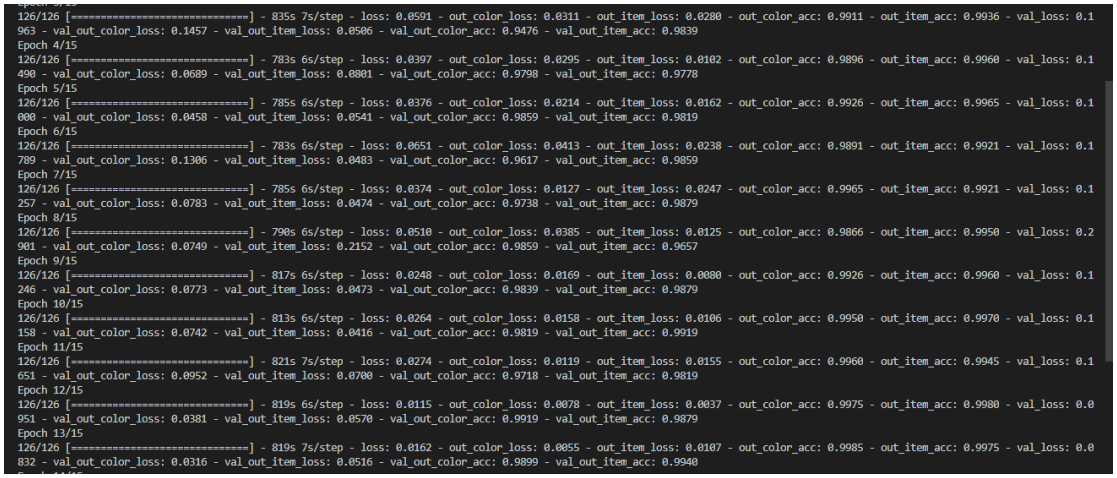
4.2 损失函数

损失函数就一个具体的样本而言，模型预测的值与真实值之间的差距。对于一个样本（xi,yi）其中 yi 为真实值，而 f（xi）为我们的预测值。使用损失函

数来表示真实值和预测值之间的差距。两者差距越小越好，最理想的情况是预测值刚好等于真实值。本实验使用 sparse_categorical_crossentropy，即交叉熵损失函数。

五 调试与结果

为了得到较好的预测结果，本次实验参数多次多种方法，不断调整学习率，同时进行多次试验，epoch 选择 15，即一次进行实验循环 15 次，尽管已经采用迁移学习的方法，但仍然需三个多小时。下图为近几次实验所得最优解：



从图中可以看出分类效果良好，正确率已经高达 95%以上，但一开始结果并不是这样，而是出现了过拟合，原因是数据集太小，而模型太复杂，尽管已经使用了数据增强但也很难去改变，为此采用两种解决方案：

1 收集更多的数据，这是解决过拟合的根本方法。

2 调试不同模型参数，因为使用了迁移学习，所以不想在模型结构上动手脚。采用以上 2 种方法后效果得到改善。

所有程序代码见附录。

六 作业总结

以前一直做的都是二分类任务，这是第一次处理多分类任务，思路简单但是真正操作起来就会有许多问题，最开始就是数据加载的时候，把所有标签路径放入列表时，忽视了直接取出不能算真正标签路径而无法加载。然后就是模型选择，一开始想使用 VGG，但参数太多模型太复杂，无法避免过拟合，最后选择了轻量

化的模型，走了不少弯路。最后就是调参了，这没什么技术含量，需要大量经验和耐心。经过一番努力模型处理多分类任务已经什么得心应手，具有代码少通俗易懂等优点，但缺点也比较明显，在处理数据量小的分类任务容易出现过拟合，所以不适合小数据集的模型。

因为能力有限，所以无法做到完美，同时在做实验的过程中也感觉到了自己的不足，所以这只是个开始，以后还要更加努力。长风破浪会有时，直挂云帆济沧海。

七 项目分工

程序调试：李永生

报告书写：陈旻扬

附录

```
import  tensorflow  as  tf
from  tensorflow  import  keras
import  matplotlib.pyplot  as  plt
import  IPython.display  as  display
import  numpy  as  np
import  random
import  pathlib
import  os

#tensorflow 会打印许多无关的信息，屏蔽信息
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'

#找到需要处理的数据集
data_dir  =  'E:/VScode/multi-output-classification/dataset'
data_root  =  pathlib.Path(data_dir)

#提取所有的文件路径，就是把所有的图片放到一起，用列表的形式,但是此时还不是真正列表的形式，而是下面这种形式
#以列表的形式将图片放到一起
#  WindowsPath('E:/VScode/multi-output-classification/dataset/black_jeans/00000001.jpeg')
all_image_paths  =  list(data_root.glob('*/*'))
image_count  =  len(all_image_paths)

#打乱所有的数据，并且转换成为真正的列表，变为下面这种形式
#E:\VScode\multi-output-classification\dataset\blue_shirt\00000133.jpg
all_image_paths  =  [str(path)  for  path  in  all_image_paths]
```

```
random.shuffle(all_image_paths)

#后面要对数据进行编码，所以先提取一共有多少类
#['black_jeans', 'black_shoes', 'blue_dress', 'blue_jeans', 'blue_shirt', 'red_dress', 'red_shirt']
label_names = sorted(item.name for item in data_root.glob('*/') if item.is_dir())

#由于多输入，将他们分别提取出来(观察 label_names 的打印形式，以——分开颜色和衣服类型)

#取出所有颜色分类
color_label_names = set(name.split('_')[0] for name in label_names)
#取出所有衣服类型分类
item_label_names = set(name.split('_')[1] for name in label_names)
#对提取的所有分类进行编码
#{'blue': 0, 'red': 1, 'black': 2}
color_label_to_index = dict((name, index) for index,name in enumerate(color_label_names))
item_label_to_index = dict((name, index) for index,name in enumerate(item_label_names))

#对所有图片的标签进行编码
#['black_jeans', 'blue_dress', 'blue_dress']
all_image_labels = [pathlib.Path(path).parent.name for path in all_image_paths]
#分别对颜色和衣服类型编码（对所有图片）
```

```
color_labels = [color_label_to_index[label.split('_')[0]] for lab
el in all_image_labels]
item_labels = [item_label_to_index[label.split('_')[1]] for label
  in all_image_labels]

#处理数据
def load_and_preprocess_image(path):
        image = tf.io.read_file(path)
        image = tf.image.decode_jpeg(image, channels=3)
        image = tf.image.resize(image, [224, 224])
        image = tf.cast(image, tf.float32)
        image = image/255.0    # normalize to [0,1] range
        image = 2*image-1
        return image

#tf.data 处理数据
path_ds = tf.data.Dataset.from_tensor_slices(all_image_paths)
AUTOTUNE = tf.data.experimental.AUTOTUNE
image_ds = path_ds.map(load_and_preprocess_image, num_parallel_cal
ls=AUTOTUNE)

label_ds = tf.data.Dataset.from_tensor_slices((color_labels, item_
labels))
#数据一一对应
image_label_ds = tf.data.Dataset.zip((image_ds, label_ds))

#分割训练集和测试集
test_count = int(image_count*0.2)
train_count = image_count - test_count
```

```
#在 dataset 划分数据
train_data = image_label_ds.skip(test_count)
test_data = image_label_ds.take(test_count)

BATCH_SIZE = 16
train_data = train_data.shuffle(buffer_size=train_count).repeat(-1)#-1 表示一直重复
train_data = train_data.batch(BATCH_SIZE)
train_data = train_data.prefetch(buffer_size=AUTOTUNE)
test_data = test_data.batch(BATCH_SIZE)

#建立模型
#使用 MobileNetV2 卷积层
mobile_net = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3),include_top=False,weights='imagenet')
mobile_net.trianable = False

inputs = tf.keras.Input(shape=(224, 224, 3))
x=mobile_net(inputs)
x0 = tf.keras.layers.GlobalAveragePooling2D()(x)    #打平数据
#颜色输入
x1 = tf.keras.layers.Dense(1024, activation='relu')(x0)
out_color = tf.keras.layers.Dense(len(color_label_names), activation='softmax',name='out_color')(x1)
#衣服类型输入
x2 = tf.keras.layers.Dense(1024, activation='relu')(x0)
out_item = tf.keras.layers.Dense(len(item_label_names), activation='softmax',name='out_item')(x2)
```

```
model  =  tf.keras.Model(inputs=inputs,outputs=[out_color,  out_item])

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001
),loss={'out_color':'sparse_categorical_crossentropy',
        'out_item':'sparse_categorical_crossentropy'},
  metrics=['acc'])
train_steps  =  train_count//BATCH_SIZE
test_steps  =  test_count//BATCH_SIZE

model.fit(train_data,
                    epochs=15,
                    steps_per_epoch=train_steps,
                    validation_data=test_data,
                    validation_steps=test_steps)
```