

教程 | TensorFlow从基础到实战：一步步教你创建交通标志分类神经网络

2017-07-30 机器之心

选自DataCamp

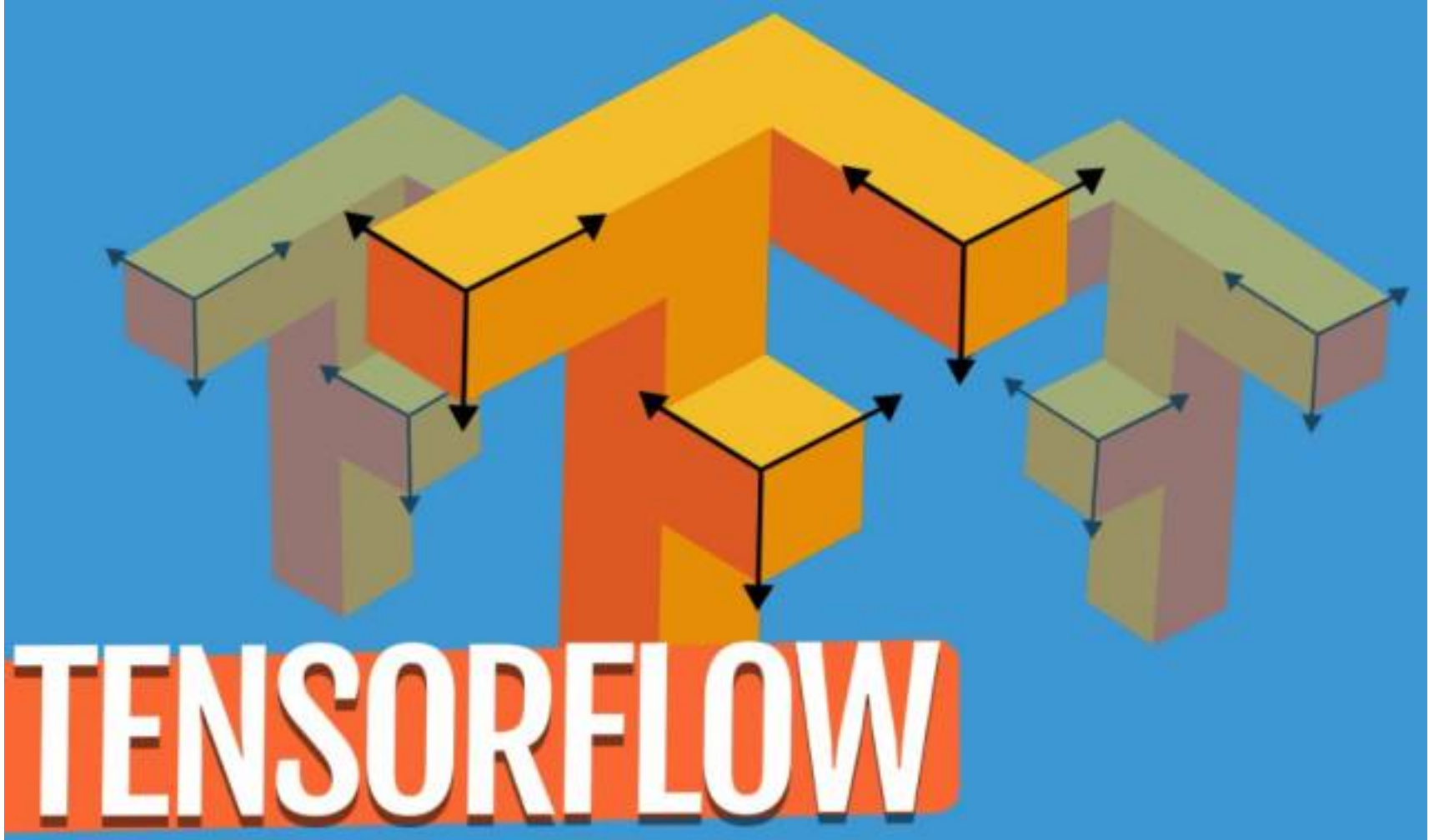
作者：Karlijn Willems

机器之心编译

参与：Panda

TensorFlow 已经成为了现在最流行的深度学习框架，相信很多对人工智能和深度学习有兴趣的人都跃跃欲试。对于初学者来说，TensorFlow 也是一个非常好的选择，它有非常丰富的入门学习资料和庞大的开发者社区。近日，数据科学学习平台 DataCamp 发表了一篇针对 TensorFlow 初学者的教程，从向量和张量的基本概念说起，一步步实现了一个分类交通标志图像的神经网络。机器之心对本教程进行了编译介绍。

深度学习是机器学习的一个子领域，包含了一系列受大脑的结构和功能启发得到的算法。



TensorFlow 是谷歌开发的第二个机器学习框架，可用于设计、构建和训练深度学习模型。你可以使用 TensorFlow 库进行数值计算，这本身似乎并没有什么特别的，但这些计算是使用数据流图完成的。在这些图中，节点表示数学运算，而边则表示数据——通常是多维的数组或张量，在这些边之间传递。

看到了吧？TensorFlow 的名字就源自神经网络在多维数组或张量上执行的这种运算！它本质上就是张量的流。这就是目前你需要了解的关于张量的所有内容，但在接下来的章节中，我们还会更加深入！

这份 TensorFlow 入门教程将会以一种交互式的方式为你呈现如何进行深度学习：

- 首先你将了解张量。
- 然后，本教程将简单介绍几种在你的系统上安装 TensorFlow 的方式，以便你能上手练习以及在你的工作空间中加载数据。
- 之后，你将了解一些 TensorFlow 基础知识：你会看到你可以轻松开始执行一些简单计算。
- 之后，你将开始进入实际工作：你将加载比利时交通标志数据，然后使用简单的统计和绘图工具对其进行探索。
- 在你的探索中，你将看到你需要操作数据，以便你能将其馈送给你的模型。所以你需要花时间调整你的图片

的大小，并将其转换成灰度图像。

- 接下来，你终于可以开始做你自己的神经网络模型了！你将一层层地构建起你的模型。
- 一旦设置好了架构，你就可以使用它来迭代式地训练你的模型，并且最终通过给它馈送一些测试数据来评估它。
- 最后，你将得到一些用于未来进步的建议指导，以便你了解你能用你刚构建好的模型做什么以及你该如何继续使用 TensorFlow 进行学习。

你也可以在这里下载本教程：<https://github.com/Kacawi/datacamp-community/blob/master/TensorFlow%20Tutorial%20For%20Beginners/TensorFlow%20Tutorial%20For%20Beginners.ipynb>

另外，你可能也会对下面三个课程感兴趣：

- Python 深度学习：<https://www.datacamp.com/courses/deep-learning-in-python>
- DataCamp 的 Keras 教程：<https://www.datacamp.com/community/tutorials/deep-learning-python>
- 使用 R 语言的 Keras 教程：<https://www.datacamp.com/community/tutorials/keras-r-deep-learning>

介绍张量

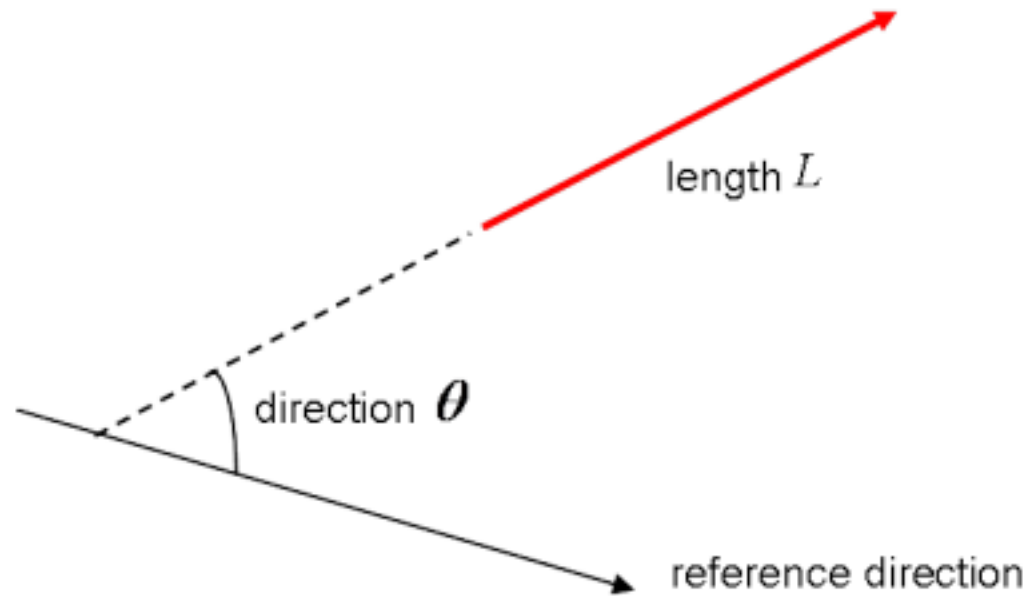
为了很好地了解张量，最好先了解一点线性代数和向量计算的知识。你在引言中已经读到了，张量在 TensorFlow 中是作为多维数据数组实现的，但为了完全理解张量及其在机器学习领域的应用，也许还是需要更多一些介绍。

平面向量 (plane vector)

在你了解平面向量之前，我们先简单澄清一下「向量」的概念。向量是特殊类型的矩阵（即数字构成的矩形阵列）。因为向量是有序的数字集合，所以它们往往被看作是列矩阵：它们只有一列和一定数量的行。换句话说，你也可以将向量看作是有一个方向的标量。

记住：标量是「5 米」或「60 米/秒」这样的量，而向量则是「向北 5 米」或「向东 60 米/秒」这样的量。这两

者之间的不同很显然：向量有一个方向。但是，到目前为止你看到的这些例子与你在机器学习问题中实际操作的向量可能相差很大。这很正常；数学向量的长度是纯数字：是绝对的。而方向则是相对的：它的度量是相对于某个参考方向，并且有弧度或度作单位。你通常假设方向是正的，并且从参考方向按逆时针方向旋转。



当然在视觉上你可以将向量表示成箭头，如上图所示。这意味着你可以将向量看作是有方向和长度的箭头。方向又箭头的头表示，而长度则由箭头的长度表示。

那么什么又是平面向量呢？

平面向量是最简单的张量配置。它们就像是你上面看到的常规向量，唯一的不同是它们处在一个向量空间中。为了更好地理解这一点，让我们从一个例子开始：你有一个 2×1 的向量。也就是说该向量属于一次配对两个数的实数集。或者换句话说，它们是一个二维空间的一部分。在这种情况下，你可以使用箭头或射线在坐标 (x,y) 平面表示向量。

从一个标准位置（其中向量的起点为 $(0,0)$ ）的坐标平面出发，你可以通过查看该向量的第一行来推导 x 坐标，同时你也可以在第二行找到 y 坐标。当然，并不一定总是要维持这种标准位置，向量可以在平面内平行移动而不发生改变。

注：类似地，对于大小为 3×1 的向量，那就是在谈论一个三维空间。你可以将该向量表示成一个三维图形，带有指向其向量速度位置的箭头：它们被画在标准的 x, y 和 z 轴上。

将这些向量表示到坐标平面上是很好的，但本质上，这些向量可以用来执行运算，为了帮助做到这一点，你可以

将你的向量表示成基础或单位向量。

单位向量是指幅度为 1 的向量，通常用带有「帽子」的小写字母表示。如果你想将一个二维或三维向量表示成两个或三个正交分量（比如 x 轴、y 轴和 z 轴）的和，单位向量可以非常方便。

而且比如当你考虑将一个向量表示成分量的和时，你会发现你在讨论分量向量，其是两个或三个向量，它们的和即是原来给出的向量。

提示：这个视频用简单的家用物品解释了张量：<https://www.youtube.com/watch?v=f5liqUk0ZTw>

张量

平面向量以及余向量（covector）和线性运算符（linear operator）这三者有一个共同点：它们都是张量的特定案例。你仍然记得前一节中如何将向量特征化为带有一个方向的标量。那么，张量就是一种带有幅度和多个方向的物理实体的一种数学表征。

而且，就向你通过单个数字表示一个标量，3 个数字的序列表示一个三维空间中的向量一样，三维空间中的张量可以通过具有 $3R$ 个数字的数组表示。

这里的 R 表示张量的秩（rank）：比如在一个三维空间中，一个第二秩张量（second-rank tensor）可以用 $3^2=9$ 个数字表示。在一个 N 维空间中，标量仍然只需要一个数字，而向量则需要 N 个数字，张量则需要 N^R 个数字。这就是为什么你常会听到人们说标量是 0 秩的张量：因为没有方向，你可以使用一个数字表示它。

记住这一点，就能轻松识别和区分标量、向量和张量了：标量可以用单个数字表示，向量是一个有序的数字集合，张量是一个数字的阵列。

张量的独特之处在于分量和基向量的组合：基向量在参考系之间按一种方式变换，分量也只按这样一种方式变换以保证分量和基向量之间的组合不变。

安装 TensorFlow

现在你对 TensorFlow 已经有了一定程度的了解了，那就让我们开始上手安装这个库吧。这里需要说明一下，TensorFlow 提供了 Python、C++、Haskell、Java、Go、Rust 的 API，另外还有一个用于 R 语言的第三方软件包 tensorflow。

提示：如果你想知道更多 R 语言的深度学习软件包，可以参阅《keras: Deep Learning in R》：
<https://www.datacamp.com/community/tutorials/keras-r-deep-learning#gs.aLGxTlg>

在本教程中，你要下载可以用 Python 编写你的深度学习项目的 TensorFlow 版本。在 TensorFlow 安装页面 <https://www.tensorflow.org/install/>，你可以看到一些使用 virtualenv、pip、Docker 和 lastly 安装 TensorFlow 的最常见方式和最新说明，当然另外也有其它一些在你的个人计算机上安装 TensorFlow 的方法。

注：如果你用的 Windows，你也可使用 Conda 安装 TensorFlow。但是因为这种安装方式是社区提供支持的，最好还是参考官方的安装说明：https://www.tensorflow.org/install/install_windows

现在你应该已经完成了安装，现在需要再次检查你是否正确安装了 TensorFlow，你可以将其导入到别名 tf 之下的工作空间：

```
import tensorflow as tf
```

注：在上面的代码中使用的别名 tf 是一种惯例——使用这个别名既能让你与其他在数据科学项目中使用 TensorFlow 的开发者保持一致，也有助于开源 TensorFlow 项目。

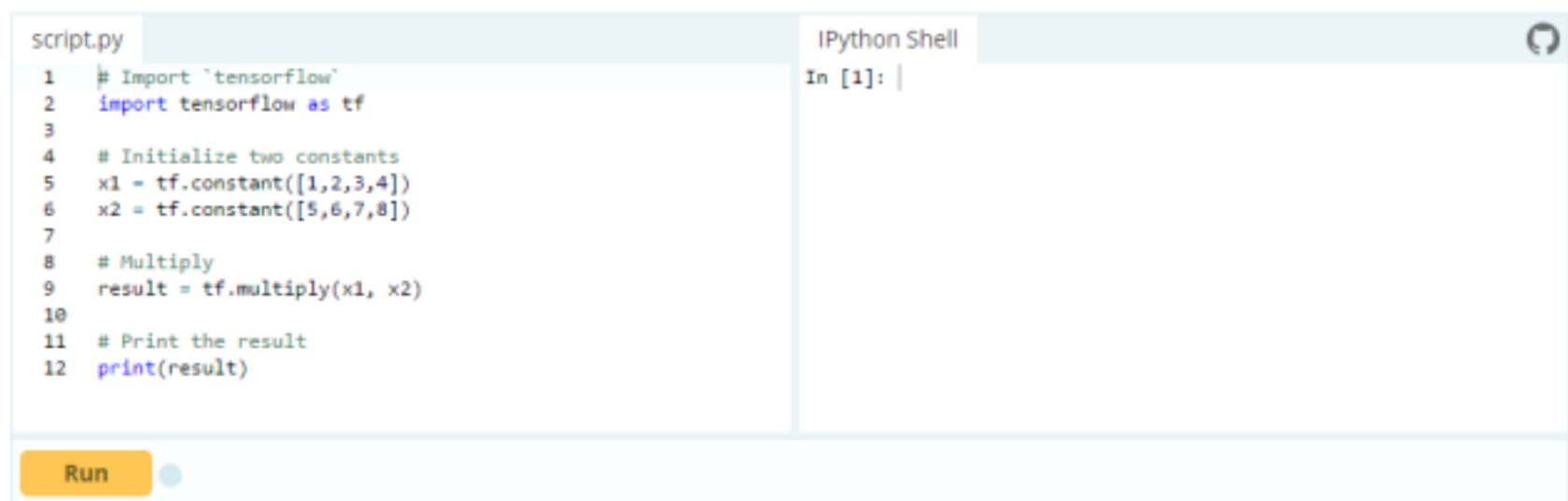
TensorFlow 入门：基础知识

你将能写一般的 TensorFlow 程序，你可以将其作为一个块（chunk）运行；当你使用 Python 时，一眼看上去这或许有点矛盾。但是，如果你愿意，你也可以使用 TensorFlow 的 Interactive Session，让你可以对 TensorFlow 进行更交互式的操作。当你习惯了 IPython 时，这会非常方便。

对于这个教程，我们的重点是第二个选项：这将有助于你通过 TensorFlow 上手深度学习。但在你深入之前，先让我们尝试一些轻量级的，之后再进入重量级挑战。

首先，在别名 `tf` 下导入 `tensorflow` 库，正如你在前一节看到的那样。然后初始化两个实际上是常量的变量。将一个 4 个数字的数组传递到 `constant()` 函数。

注：你可能也可以传入一个整数，但通常不会这样，你会发现你操作的是数组。正如引言中介绍的那样，张量就是关于数组！所以要确保你传递的是数组。接下来，你可以使用 `multiply()` 来将你的两个变量相乘。结果存储在 `result` 变量中。最后，在 `print()` 函数的帮助下显示 `result` 结果。



The screenshot shows a Jupyter Notebook interface. On the left, a code editor window titled 'script.py' contains the following Python code:

```
1 # Import 'tensorflow'
2 import tensorflow as tf
3
4 # Initialize two constants
5 x1 = tf.constant([1,2,3,4])
6 x2 = tf.constant([5,6,7,8])
7
8 # Multiply
9 result = tf.multiply(x1, x2)
10
11 # Print the result
12 print(result)
```

On the right, an IPython Shell window titled 'IPython Shell' shows the prompt 'In [1]: |'. At the bottom of the interface is a yellow 'Run' button with a circular arrow icon to its right.

注：你已经在上面的 DataCamp Light 代码块中定义了常量，但你也可以使用另外两种类型的值，即占位符（placeholder）和变量（variable）。占位符是指没有分配的值，在你开始运行该 `session` 时会被初始化。正如其名，它只是张量的一个占位符，在 `session` 运行时总是得到馈送。变量是指值可以变化的量。而前面你已经见过的常量的值则不会变化。

这几行代码的结果是计算图中的一个抽象张量。但是可能与你预期的相反，`result` 实际上没有得到计算；它只是定义了模型，但没有运行进程来计算结果。你可以在显示输出中看到，其中并没有你希望看到的结果（比如 30）。这意味着 TensorFlow 的评估很懒惰！

但是，如果你确实想看到这个结果，你就必须以一种交互式会话的方式运行这段代码。做到这一点的方式有好几个，下面展示了在 DataCamp Light 代码块中的情况：

script.py

```
1 # Import `tensorflow`
2 import tensorflow as tf
3
4 # Initialize two constants
5 x1 = tf.constant([1,2,3,4])
6 x2 = tf.constant([5,6,7,8])
7
8 # Multiply
9 result = tf.multiply(x1, x2)
10
11 # Initialize the Session
12 sess = tf.Session()
13
14 # Print the result
15 print(sess.run(result))
16
17 # Close the session
18 sess.close()
```

IPython Shell

In [1]: |

Run

注：你可以使用以下代码来启动一个交互式 Session，运行 result，并在显示输出 output 之后再次自动关闭 Session。

script.py

```
1 # Import `tensorflow`
2 import tensorflow as tf
3
4 # Initialize two constants
5 x1 = tf.constant([1,2,3,4])
6 x2 = tf.constant([5,6,7,8])
7
8 # Multiply
9 result = tf.multiply(x1, x2)
10
11 # Initialize Session and run `result`
12 with tf.Session() as sess:
13     output = sess.run(result)
14     print(output)
```

IPython Shell

In [1]: |

Run

在上面的代码块中，你刚刚定义了一个默认 Session，但要知道你也可以传递选项。比如说，你可以指定 config 参数，然后使用 ConfigProto 协议缓冲来为你的 session 增加配置选项。

比如说，如果你为你的 Session 增加了 config=tf.ConfigProto(log_device_placement=True)，你就可以确保你录入了运算分配到的 GPU 和 CPU 设备。然后你就可以了解在该 session 中每个运算使用了哪个设备。比如当你为设备配置使用软约束时，你也可使用下面的配

置 session: config=tf.ConfigProto(allow_soft_placement=True).

现在你已经装好了 TensorFlow，并且将其导入到了你的工作空间，你也了解了操作这个软件包的基础知识。现在是时候调整一下方向，关注一下数据了。就像了解其它事情一样，首先你要花时间探索和更好地了解你的数据，然后才能开始建模你的神经网络。

比利时交通标志：背景

尽管交通是一个通常你们都知道的主题，但不妨也简单了解一下这个数据集中的数据，看你在开始之前是否已经明白了一切。本质上，这一节将让你快速了解你进一步学习本教程所需的领域知识。

当然，因为我是比利时人，所以用了比利时的交通标志数据。这里给出了一些有趣的轶事：

- 比利时交通标志通常为荷兰语和法语。了解这一点当然不错，但对于你要操作的数据集，这并不重要！
- 比利时的交通标志分成六大类：警告标志、优先标志、禁令标志、强制性标志、与停车和在路上等待相关的标志、指示牌。
- 在 2017 年 1 月 1 日，比利时移除了超过 30,000 块交通标志。它们都是与速度相关的优先标志。
- 谈到移除，在比利时（而且可以扩展到整个欧盟），过多的交通标志一直以来都是一个讨论话题。

加载和探索数据

现在你已经得到了更多一些背景信息，该下载数据了：<http://btsd.ethz.ch/shareddata/>。你应该下载 BelgiumTS for Classification (cropped images) 右边的两个压缩包：BelgiumTSC_Training 和 BelgiumTSC_Testing。

提示：如果你已经下载了这些文件或者将会在完成本教程后下载，那就看看你下载的数据的文件夹结构！你将看到测试和训练数据文件夹包含了 61 个子文件夹，这是你将在本教程中使用的用于分类任务的 62 种类型的交通标志。另外，你会发现这些文件的文件扩展名是 .ppm，即 Portable Pixmap Format。你已经下载好了交通标志图片！

让我们先把这些数据导入到你的工作空间。让我们先从出现在用户定义函数（UDF）load_data() 下面的代码行开

始：

- 首先，设置你的 ROOT_PATH。这个路径是带有你的训练数据和测试数据的目录。
- 接下来，你可以借助 join() 函数为 ROOT_PATH 增加特定的路径。你将这两个特定的路径存储在 train_data_directory 和 test_data_directory 中
- 之后，你可以调用 load_data() 函数，并将 train_data_directory 作为它的参数。
- 现在 load_data() 启动并自己开始收集 train_data_directory 下的所有子目录；为此它借助了一种被称为列表推导式（list comprehension）的方法——这是一种构建列表的自然方法。基本上就是说：如果在 train_data_directory 中发现了一些东西，就双重检查这是否是一个目录；如果是，就将其加入到你的列表中。注意：每个子目录都代表了一个标签。
- 接下来，你必须循环遍历这些子目录。首先你要初始化两个列表：labels 和 imanges。然后你要收集这些子目录的路径以及存储在这些子目录中的图像的文件名。之后，你可以使用 append() 函数来收集这两个列表中的数据。

```
def load_data(data_directory):
    directories = [d for d in os.listdir(data_directory)
                    if os.path.isdir(os.path.join(data_directory, d))]
    labels = []
    images = []
    for d in directories:
        label_directory = os.path.join(data_directory, d)
        file_names = [os.path.join(label_directory, f)
                        for f in os.listdir(label_directory)
                        if f.endswith(".ppm")]
        for f in file_names:
            images.append(skimage.data.imread(f))
            labels.append(int(d))
    return images, labels

ROOT_PATH = "/your/root/path"
train_data_directory = os.path.join(ROOT_PATH, "TrafficSigns/Training")
test_data_directory = os.path.join(ROOT_PATH, "TrafficSigns/Testing")

images, labels = load_data(train_data_directory)
```

注意在上面的代码块中，训练数据和测试数据分别位于名为 Training 和 Testing 的文件夹中，这两个文件夹都是另一个目录 TrafficSigns 的子目录。在一个本地机器上，其路径可能是这样

的： /Users/yourName/Downloads/TrafficSigns，之后带有两个子文件夹 Training 和 Testing。

交通标志统计数据

加载好你的数据后，就可以做些数据检查（data inspection）了！首先你可以使用 images 数组的 ndim 和 size 属性做一些相当简单的分析：

注意 images 和 labels 变量都是列表，所以你可能需要使用 np.array() 将这些变量转换成你工作空间中的数组。这里已经为你做好了！

script.py

```
1 # Print the `images` dimensions
2 print(images.ndim)
3
4 # Print the number of `images`'s elements
5 print(images.size)
6
7 # Print the first instance of `images`
8 images[0]
```

IPython Shell

In [1]: |

Run

注：响应输出的 images[0] 实际上是由数组中的数组表示的单个图像。一开始这可能看起来与直觉相反，但随着你在机器学习或深度学习应用中对图像操作的进一步理解，你会习惯的。

接下来，你也可以简单看看 labels，但现在你应该不会太过惊讶了：

script.py

```
1 # Print the `labels` dimensions
2 print(labels.ndim)
3
4 # Print the number of `labels`'s elements
5 print(labels.size)
6
7 # Count the number of labels
8 print(len(set(labels)))
```

IPython Shell

```
1
4575
62

In [1]: |
```

Run

这些数字能让你了解你的导入有多成功以及你的数据的确切大小。大略看看，一切都是按照你预期的方式执行的，而且如果你考虑到你正在处理数组中的数组，那么你会看到数组的大小是相当大的。

提示：试试将以下属性添加到你的数组中，即使用 `flags`、`itemsize` 和 `nbytes` 属性获取关于内存布局（memory layout）、一个数组元素的字节长度和总消耗字节数的更多信息。你可以在上面的 DataCamp Light 块中的 IPython 控制台中测试这些属性！

接下来，你也可以看看这些交通标志数据的分布情况：

script.py

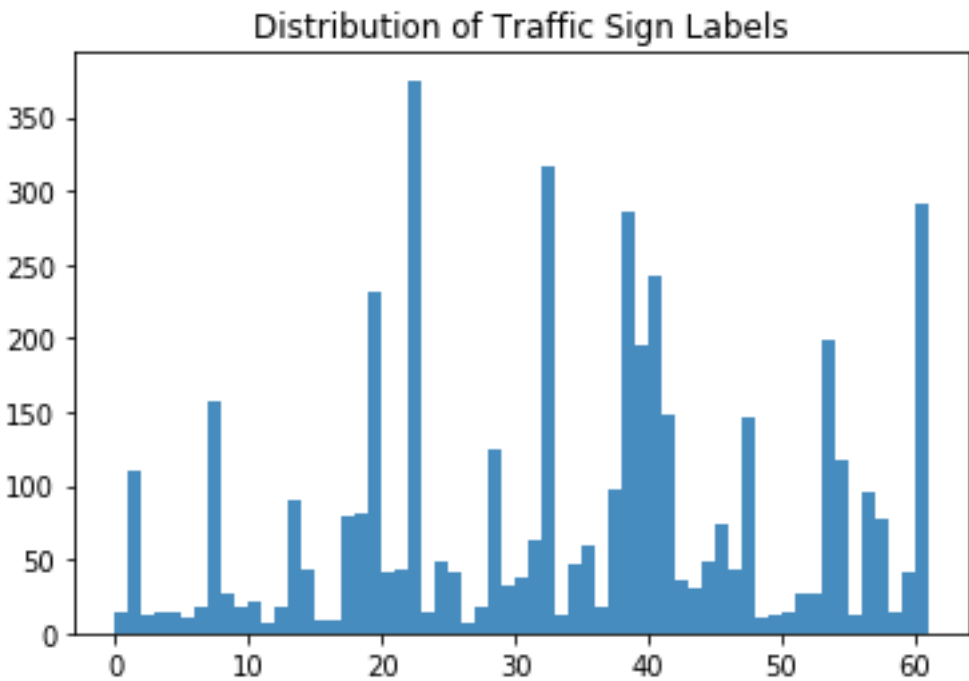
```
1 # Import the 'pyplot' module
2 import matplotlib.pyplot as plt
3
4 # Make a histogram with 62 bins of the 'labels' data
5 plt.hist(labels, 62)
6
7 # Show the plot
8 plt.show()
```

IPython Shell

In [1]: |

Run

干得漂亮！现在让我们仔细看看你做出的直方图！



你可以清楚地看到，所有类型的交通标志都在该数据集中得到了平等地表示。这是后面你开始建模神经网络之前操作你的数据时所需要处理的东西。

大略来看，你可以看到该数据集中有的标签的分量比其它标签更重：标签 22、32、38 和 61 显然出类拔萃。现在记住这一点，在下一节你会更深入地了解！

可视化交通标志数据

前面的小型分析和检查已经让你对这些数据有所了解了，但是当你的数据基本上是由图像构成时，你应该通过可视化的方式来探索你的数据。

我们先来看看一些随机的交通标志：

- 首先，确保你在常用别名 `plt` 下导入了 `matplotlib` 软件包的 `pyplot` 模块。
- 然后，你需要创建一个带有 4 个随机数字的列表。这些会被用于从 `images` 数组中选择你在前一节检查过的交通标志。在这个案例中，你会选择 300、2250、3650 和 4000。
- 接下来，对于列表长度中的每个元素，即从 0 到 4，你要创建一个没有轴的子图（subplot）（这样它们就不会关注所有东西，而只会单独关注于图像！）。在这些子图中，你将展示与索引 `i` 中数字相符的 `images` 数组中的特定图像。在第一次循环中，你会把 300 传递给 `images[i]`，在第二轮传递 2250，依此类推。最后你需要调整子图使它们之间具有足够的宽度。
- 最后使用 `show()` 函数展示你的图表！

代码如下：

```
# Import the `pyplot` module of `matplotlib`
import matplotlib.pyplot as plt

# Determine the (random) indexes of the images that you want to see
traffic_signs = [300, 2250, 3650, 4000]
```

```
# Fill out the subplots with the random images that you defined
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)

plt.show()
```

因为该数据集包含 62 个标签，你大概也猜到了这些交通标志彼此之间存在差异。

但你还注意到了什么？仔细看看下面的图像：



这四张图像的大小不一样！

显然你可以尝试调整包含在 `traffic_signs` 列表中的数字，并对这个观察进行更全面的跟踪。但实际上，这是一个重要的观察结果——当你开始操作你的数据以便将其输入神经网络时，你会需要将这个观察结果考虑进来。

让我们通过输出包含在这些子图中的特定图像的形状（`shape`）、最小值和最大值来确认图片大小不同的假设。

下面的代码与前面用来创建上面的图的代码非常相似，但不同的地方在于你将交替大小和图像，而不只是绘制接连的图像。

```
# Import `matplotlib`import matplotlib.pyplot as plt
```

```
# Determine the (random) indexes of the images
traffic_signs = [300, 2250, 3650, 4000]

# Fill out the subplots with the random images and add shape,
# min and max values
for i in range(len(traffic_signs)):

    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images[traffic_signs[i]])
    plt.subplots_adjust(wspace=0.5)
    plt.show()
    print("shape: {0}, min: {1}, max: {2}".format(images[traffic_signs[i]].shape,
                                                  images[traffic_signs[i]].min(),
                                                  images[traffic_signs[i]].max()))
```

注：你如何在 `shape: {0}, min: {1}, max: {2}` 串上使用 `format()` 方法来填充你定义的参数 `{0}`、`{1}` 和 `{2}`。



```
shape: (62, 61, 3), min: 3, max: 160
```



```
shape: (110, 96, 3), min: 3, max: 255
```



```
shape: (379, 153, 3), min: 0, max: 255
```



```
shape: (100, 68, 3), min: 17, max: 255
```

现在你已经看到了松散的图像，你可能还需要再看看你在数据探索的开始阶段输出的直方图：你可以通过绘制所有 62 个类的整体情况以及属于每个类的一张图像来轻松做到这一点。

```
# Import the `pyplot` module as `plt`import matplotlib.pyplot as plt
```

```
# Get the unique labels unique_labels = set(labels)
```

```
# Initialize the figureplt.figure(figsize=(15, 15))
```



```

# Set a counter i = 1
# For each unique label, for label in unique_labels:
# You pick the first image for each label
    image = images[labels.index(label)]
    # Define 64 subplots
    plt.subplot(8, 8, i)
    # Don't include axes
    plt.axis('off')
    # Add a title to each subplot
    plt.title("Label {0} ({1})".format(label, labels.count(label)))
    # Add 1 to the counter
    i += 1
    # And you plot this first image
plt.imshow(image)
# Show the plot plt.show()

```

注：即使你定义了 64 个子图，也并不是它们所有都会展示图像（因为总共只有 62 个标签！）。再注意一下，你不需要包含任何轴以确保阅读器的注意不会偏离主要主题：交通标志！



正如你基本上在上述直方图中猜到的那样，具有标签 22、32、38 和 61 的交通标志要多得多。上面的假设在这幅图中得到了确认：你可以看到标签 22 有 375 个实例，标签 32 有 316 实例，标签 38 有 285 个实例，标签 61 有 282 个实例。

你可能要问一个有趣的问题：所有这些实例之间有什么联系吗——也许它们都是指示标志？

让我们仔细看看：你可以看到标签 22 和 32 是禁令标志，而标签 38 和 61 则分别是指示标志和优先标志。这意味着这四者之间并没有直接的联系，只是禁令标志大量存在，占到了该数据集的一半。

特征提取

现在你已经全面探索了你的数据，是时候开始实战了！让我们简要回顾一下你的发现，以确保你没有忘记操作中的任何步骤。

- 图像的大小是不同的；
- 存在 62 个标签或目标值（因为你的标签始于 0，终于 61）；
- 交通标志值的分布相当不平衡；数据集中大量存在的标志之间实际上并没有任何联系。

现在你已经清楚了解了你需要改进的内容，你可以从加工你的数据开始，以使其可以用于神经网络或任何你想让用来处理该数据的模型中——你将重新调整图像的大小，并将这些保存在 `images` 数组中的图像转换成灰度图像。进行这种颜色转换的主要原因是颜色在你正尝试解决的这种分类问题上作用不大。但是对于检测任务来说，颜色确实发挥了很大作用！所以在这些案例中，不需要进行这样的转换！

重调图像大小

为了解决不同图像大小的问题，你需要重调图像大小。你可以使用 `skimage` 或 `Scikit-Image` 库轻松实现这一目标；`Scikit-Image` 是一个用于图像处理的算法的集合。

在这个案例中，`transform` 模块用起来很方便，因为其提供了一个 `resize()` 函数；你将看到你会使用列表推导式（再一次！）来将每张图像的大小调整为 `28×28` 像素。再一次，你将看到你实际构建列表的方式：对于每一张你在 `images` 数组中找到的图像，你都可以执行从 `skimage` 库借用的变换运算。最后，你将结果存储在 `images28` 变量中：

```
# Import the `transform` module from `skimage`
from skimage import transform

# Rescale the images in the `images` array
images28 = [transform.resize(image, (28, 28)) for image in images]
```

是不是很简单？

注：图像现在是 4 维的：如果你将 `images28` 转换成一个数组，并且如果你将属性 `shape` 连接到它，你就会看到输出的结果表明 `images28` 的维度为 `(4575, 28, 28, 3)`。这些图像是 784 维的（因为你的图像是 28×28 像素的）。

你可以借助 `traffic_signs` 变量，通过复用上面你用过的代码来给出 4 张随机图像的图表，从而对大小调整操作进行检查。只是不要忘记将所有的参考都从 `images` 改成 `images28`。

结果如下：



```
shape: (28, 28, 3), min: 0.061764705882353076, max: 0.6161764705882353
```



```
shape: (28, 28, 3), min: 0.07634053621448501, max: 1.0
```



```
shape: (28, 28, 3), min: 0.08464760904361845, max: 1.0
```



```
shape: (28, 28, 3), min: 0.08907563025210051, max: 1.0
```

注：因为你调整过大小了，所以你的 min 和 max 值都已经改变了；现在它们似乎全部都在同一个范围内，这是一件非常好的事，因为你不再需要规范化你的数据了！

将图像转换成灰度图像

正如本节介绍中提到的那样，在尝试解决分类问题时，图像的颜色作用更小。所以你也需要麻烦一下，将图像转换成灰度图像。但是注意，你也可以自己试试看如果不做这一步，对你的模型的最终结果有什么影响。就像调整图像大小一样，你也可以使用 Scikit-Image 库来解决；在这个案例中，我们要在你需要的地方使用 color 模块中的 `rgb2gray()` 函数。这非常好用，而且很简单！

但是，不要忘记将 image28 变量转换回数组，因为 rgb2gray() 函数并不使用数组作为参数。

```
# Import `rgb2gray` from `skimage.color`
from skimage.color import rgb2gray
# Convert `images28` to an array
images28 = np.array(images28)
# Convert `images28` to grayscale
images28 = rgb2gray(images28)
```

再通过输出一些图像的图表来检查一下你的灰度转换；这里，你可以再次复用并稍微调整一些代码来显示调整后的图像：

```
import matplotlib.pyplot as plt

traffic_signs = [300, 2250, 3650, 4000]
for i in range(len(traffic_signs)):
    plt.subplot(1, 4, i+1)
    plt.axis('off')
    plt.imshow(images28[traffic_signs[i]], cmap="gray")
    plt.subplots_adjust(wspace=0.5)
# Show the plotplt.show()
```

注：你确实必须指定颜色图（即 cmap），并将其设置为 gray 以给出灰度图像的图表。这是因为 imshow() 默认使用一种类似热力图的颜色图。更多信息参阅：<https://stackoverflow.com/questions/39805697/skimage-why-does-rgb2gray-from-skimage-color-result-in-a-colored-image>

提示：因为在本教程中你已经复用了这个函数很多次，所以你可以看看你是如何使其成为一个函数的：)

这两个步骤是非常基础的；其它你可以在你的数据上尝试的操作包括数据增强（旋转、模糊、位移、改变亮度.....）等。如果你愿意，你也可以通过你发送图像的方式来设置整个数据操作运算的流程。

使用 TensorFlow 做深度学习

现在你已经探索并操作好了你的数据，该使用 TensorFlow 软件包来构建你的神经网络架构了！

建模神经网络

就像你可以用 Keras 做的那样，现在是时候一层层构建你的神经网络了。

如果你还没试过，就先将 tensorflow 导入到惯例别名 tf 的工作空间中。然后你可以借助 Graph() 对图进行初始化。你可以使用这个函数来定义计算。注意如果只有这个图，那你就不能计算任何东西，因为它不保留任何值。它只是定义你想在之后运行的运算。

在这个案例中，你可以使用 as_default() 设置一个默认背景，该函数会返回一个背景管理器，可使这个特定图成为默认的图。如果你想在这同一个流程中创建多个图，你也可以使用这种方法：使用这个函数，如果你不明确创建一个新图，那你就为所有将被加入的运算设置了一个全局默认图。

接下来，你就可以将运算加入到你的图中了。在 Keras 中，你需要先构建你的模型，然后在编译它，你要定义一个损失函数、一个优化器和一个指标。而使用 TensorFlow，你一步就能完成上述操作：

- 首先，你要为输入和标签定义占位符，因为你尚未输入「真实」数据。记住占位符是未分配的值，可以在你运行它时被 session 初始化。所以当你运行该 session 结束时，这些占位符会获取你在 run() 函数中传递的数据集中的值。
- 然后构建你的网络。首先使用 flatten() 函数展平输入，其会给你一个形状为 [None, 784] 的数组，而不是 [None, 28, 28]——这是你的灰度图像的形状。
- 在你展平了输入之后，构建一个全连接层，其可以生成大小为 [None, 62] 的 logits。logits 是运行在早期层未缩放的输出上的函数，其使用相对比例来了解单位是否是线性的。
- 构建出多层感知器后，就可以定义损失函数了。损失函数的选择取决于当前的任务：在这个案例中，你可以使用 sparse_softmax_cross_entropy_with_logits()，其可以计算 logits 和标签之间的稀疏 softmax 交叉熵。换句话说，它测量的是离散分类任务中的概率误差，在这些任务中，类别是互斥的。这意味着每一个条目（entry）都是一个单独的类别。在这种情况下，一个交通标志只会有一个标签。记住这一点，回归（regression）被用于预测连续值，而分类（classification）则被用于预测离散值或数据点的类别。你可以使用 reduce_mean() 来包裹这个函数，它可以计算一个张量的维度上各个元素的均值。

- 你也需要定义一个训练优化器；最流行的优化算法包括随机梯度下降（SGD）、ADAM 和 RMSprop。取决于你选择的算法，你需要调节特定的参数，比如学习率或动量。在这个案例中，你要选择 ADAM，并将其学习率定义为 0.001。
- 最后，你可以在进入训练之前初始化要执行的运算。

现在你已经用 TensorFlow 创造出了你的第一个神经网络！

```
# Import `tensorflow`
import tensorflow as tf

# Initialize placeholders
x = tf.placeholder(dtype = tf.float32, shape = [None, 28, 28])
y = tf.placeholder(dtype = tf.int32, shape = [None])

# Flatten the input data
images_flat = tf.contrib.layers.flatten(x)

# Fully connected layer
logits = tf.contrib.layers.fully_connected(images_flat, 62, tf.nn.relu)

# Define a loss function
loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(labels = y,
                                                                    logits = logits))

# Define an optimizer
train_op = tf.train.AdamOptimizer(learning_rate=0.001).minimize(loss)

# Convert logits to label
indexescorrect_pred = tf.argmax(logits, 1)

# Define an accuracy metric
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))
```

如果你想，你可以显示输出大部分变量的值以快速了解和检查你刚刚编写的东西：

```
print("images_flat: ", images_flat)
```



```
print("logits: ", logits)
print("loss: ", loss)
print("predicted_labels: ", correct_pred)
```

提示：如果看到了「module 'pandas' has no attribute 'computation'」这样的错误提示，可以通过在命令行运行 `pip install --upgrade dask` 来升级 `dask` 软件包。参见这个帖子了解更多：
<https://stackoverflow.com/questions/43833081/attributeerror-module-object-has-no-attribute-computation>

运行神经网络

现在你已经一层层地构建好了你的模型，可以跑起来了！要做到这一点，首先你需要使用 `Session()` 初始化一个 `session`，你可以将你在之前一节定义的图 `graph` 传递给该函数。接下来，将同样在前一节定义的初始化运算 `init` 变量传递给 `run()`，并通过该函数运行该 `session`。

接下来，你可以使用这些初始化的 `session` 来启动 `epoch` 或训练循环。在这个案例中，你可以选择 201，因为你需要注册最新的 `loss_value`；在训练循环中，使用你在前一节定义的训练优化器和损失（或准确度）指标，运行这个 `session`。你也可以传递一个 `feed_dict` 参数，你可以使用它将数据传递给模型。每 10 个 `epoch` 之后，你会获得一个日志，能给你提供更多有关模型的损失或成本的信息。

正如你在 TensorFlow 基础知识一节看到的那样，我们不需要人工关闭 `session`；TensorFlow 会自动完成。但是，如果你想尝试不同的设置，当你将你的 `session` 定义为 `sess` 时，你可能将需要使用 `sess.close()`，就像在下列代码块中一样：

```
tf.set_random_seed(1234)
sess = tf.Session()

sess.run(tf.global_variables_initializer())for i in range(201):
    print('EPOCH', i)
    accuracy_val = sess.run([train_op, accuracy], feed_dict={x: images28, y: labels})
    if i % 10 == 0:
        print("Loss: ", loss)
    print('DONE WITH EPOCH')
```

记住，你也可以运行下面这段代码，但这段代码会在之后立即关闭 session，就像你在本教程的引言中看到的那样：

```
tf.set_random_seed(1234)with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(201):
        _, loss_value = sess.run([train_op, loss], feed_dict={x: images28, y: labels})
        if i % 10 == 0:
            print("Loss: ", loss)
```

注：你使用了 `global_variables_initializer()`，因为不提倡使用 `initialize_all_variables()` 函数。

现在你已经成功训练了你的模型！并不是很难吧？

评估你的神经网络

训练完了并不是结束；你还需要对你的神经网络进行评估。在这个案例中，你可以通过选取随机 10 张图像并比较预测标签和真实标签来简单了解你的模型的表现。

你首先可以显示输出它们，但为什么不使用 `matplotlib` 来得到交通标志本身的图表并进行视觉比较呢？

```
# Import `matplotlib`
import matplotlib.pyplot as pltimport random

# Pick 10 random images
sample_indexes = random.sample(range(len(images28)), 10)
sample_images = [images28[i] for i in sample_indexes]
sample_labels = [labels[i] for i in sample_indexes]

# Run the "correct_pred" operation
predicted = sess.run([correct_pred], feed_dict={x: sample_images})[0]
# Print the real and predicted labelsprint(sample_labels)
print(predicted)
```

```
# Display the predictions and the ground truth
visually.fig = plt.figure(figsize=(10, 10))for i in range(len(sample_images)):
    truth = sample_labels[i]
    prediction = predicted[i]
    plt.subplot(5, 2,1+i)
    plt.axis('off')
    color='green' if truth == prediction else 'red'
    plt.text(40, 10, "Truth:          {0}\nPrediction: {1}".format(truth, prediction),
            fontsize=12, color=color)
    plt.imshow(sample_images[i],  cmap="gray")

plt.show()
```



Truth: 0
Prediction: 1



Truth: 13
Prediction: 13



Truth: 38
Prediction: 38



Truth: 47
Prediction: 19



Truth: 26
Prediction: 24



Truth: 3
Prediction: 13



Truth: 39
Prediction: 39



Truth: 35
Prediction: 22



Truth: 61
Prediction: 31



Truth: 38
Prediction: 38

但是，仅仅查看随机图像并不能让你了解你的模型的表现究竟如何。所以你需要加载测试数据。

```
# Import `skimage` from sk
image import transform

# Load the test data
test_images, test_labels = load_data(test_data_directory)

# Transform the images to 28 by 28 pixels
test_images28 = [transform.resize(image, (28, 28)) for image in test_images]
```

```

# Convert to grayscale
from skimage.color import rgb2gray
test_images28 = rgb2gray(np.array(test_images28))

# Run predictions against the full test set.
predicted = sess.run([correct_pred], feed_dict={x: test_images28})[0]

# Calculate correct matches
match_count = sum([int(y == y_) for y, y_ in zip(test_labels, predicted)])

# Calculate the accuracy
accuracy = match_count / len(test_labels)

# Print the accuracy
print("Accuracy: {:.3f}".format(accuracy))

```

注：你要使用 `load_data()` 函数，你在本教程开始时定义了这个函数。

记住使用 `sess.close()` 关闭 session，以防你没有使用 `with tf.Session() as sess:` 来启动你的 TensorFlow session。

接下来看什么？

如果你想继续使用这个教程中的数据集和模型，可以尝试下面的东西：

- 在你的数据上应用正则化 LDA，之后再将其应用于你的模型。这个建议来自于一篇原始论文：<http://btsd.ethz.ch/shareddata/publications/Mathias-IJCNN-2013.pdf>，是由收集并分析这个数据集的研究者写的。
- 正如在本教程中说的那样，你也可以在这些交通标志数据上执行一些其它的数据增强操作。此外，你也可以尝试进一步调整这个网络——你目前创造的这个网络还相当简单。
- 早停（early stopping）：在你训练神经网络时，要一直关注训练和测试误差。当两个误差都下降又突然升高时要停止训练——这是神经网络开始过拟合训练数据的征兆。

一定要看 Nishant Shukla 写的书《Machine Learning With TensorFlow》：

<https://www.manning.com/books/machine-learning-with-tensorflow>

提示：另外也看看 TensorFlow Playground（<http://playground.tensorflow.org>）和 TensorBoard（https://www.tensorflow.org/get_started/summaries_and_tensorboard）。

如果你想继续处理图像，肯定不能错过 DataCamp 的 scikit-learn 教程：<https://www.datacamp.com/community/tutorials/machine-learning-python>，在这个教程中你将了解如何用 PCA、K-均值和支持向量机（SVM）来处理 MNIST 数据集。你也可以看看其它使用了比利时交通标志数据集的教程，比如：<https://github.com/waleedka/traffic-signs-tensorflow/blob/master/notebook1.ipynb>

原文地址：<https://www.datacamp.com/community/tutorials/tensorflow-tutorial>

本文为机器之心编译，转载请联系本公众号获得授权。



加入机器之心（全职记者/实习生）：hr@jiqizhixin.com

投稿或寻求报道：editor@jiqizhixin.com

广告&商务合作：bd@jiqizhixin.com

[阅读原文](#)