AI学习笔记--Tensorflow--基本图形分类

本章主要学习一下关于官方文档中初学部分的 Keras 部分。首先开始的是第一部分，基础图形的分类。官网地址如下：

https://tensorflow.google.cn/tutorials/keras/classification

首先导入 TensorFlow 提供的学习数据集合，按照 Google 的文档提示，我们使用 Keras 提供的训练集作为训练对象。大致提供了四个训练集。

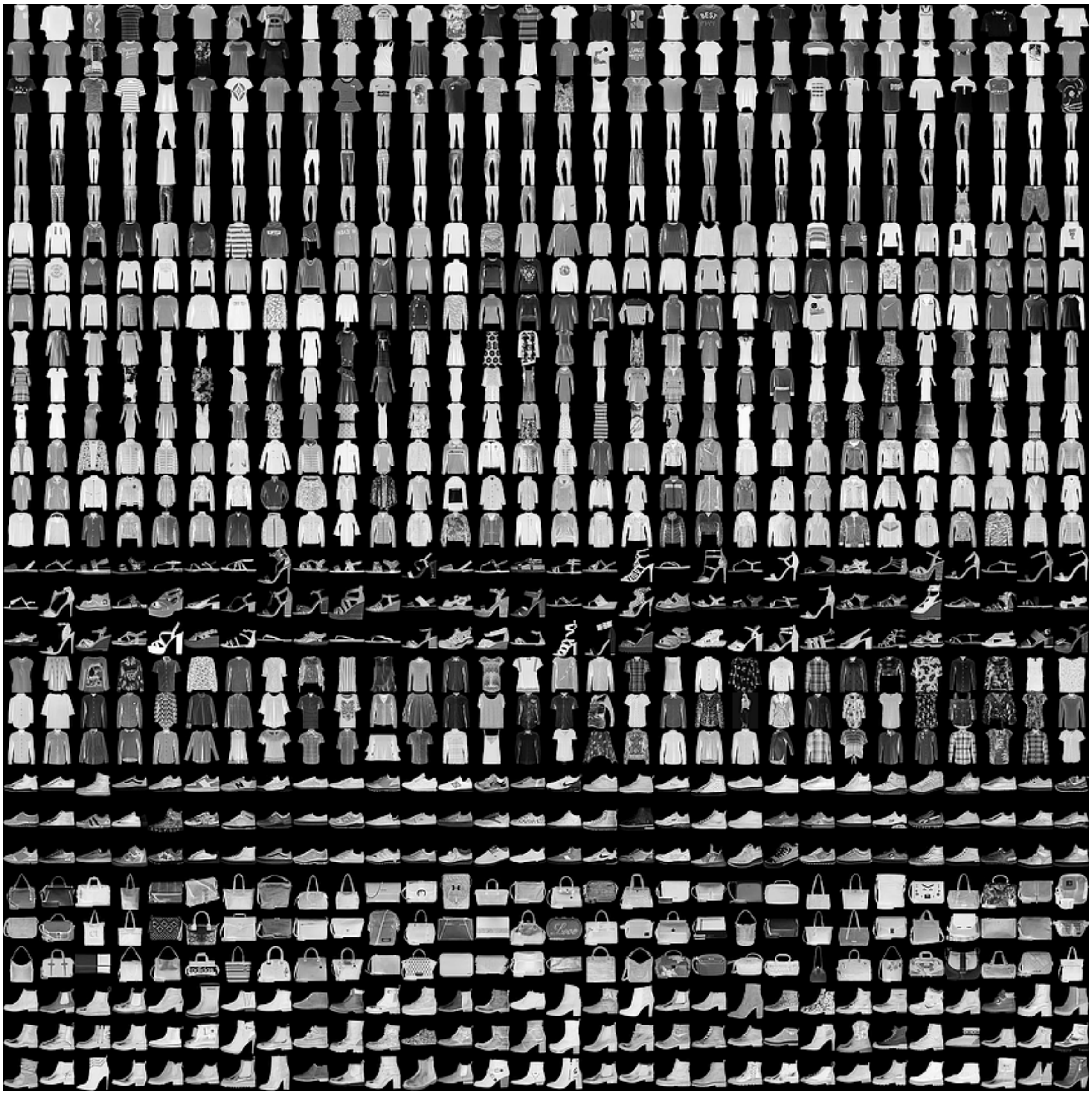
You can use direct links to download the dataset. The data is stored in the **same** format as the original MNIST data.

Name	Content	Examples	Size	Link
`train-images-idx3-ubyte.gz`	training set images	60,000	26 MBytes	Dowi
`train-labels-idx1-ubyte.gz`	training set labels	60,000	29 KBytes	Dowi
`t10k-images-idx3-ubyte.gz`	test set images	10,000	4.3 MBytes	Dowi
`t10k-labels-idx1-ubyte.gz`	test set labels	10,000	5.1 KBytes	Dowi

```
from __future__ import absolute_import, division, print_function,
unicode_literals

# TensorFlow and tf.keras
import tensorflow as tf
from tensorflow import keras

# Helper libraries
import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)

fashion_mnist = keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) =
```

```
fashion_mnist.load_data()

class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

按照官方文档的介绍，这些数据集合都对数据进行了分类和归纳，具体的数据集可以看下图：



Fashion-MNIST数据库一共提供了 60，000 个样本训练集合，并且可以测试将近10.000个测试样例。每一个图样都是一个 28*28 像素大小的图片，并且按照十种分类进行分类标注。上图可以看到，每一种分类包含了三列小型图例。

默认 PC 情况下，如果在使用上述代码去爬取数据时，系统可能会出现异常，主要原因是 Google 服务器无法访问的问题。

```
Exception: URL fetch failure on
https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz: None
-- [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to
get local issuer certificate (_ssl.c:1056)
```

出现这个情况，最简单的解决方案是修改源文件函数的指定向 URL：

```
# limitations under the License.
#
=============================================================================
===
"""Fashion-MNIST dataset.
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import gzip
import os

import numpy as np

from tensorflow.python.keras.utils.data_utils import get_file
from tensorflow.python.util.tf_export import keras_export


@keras_export('keras.datasets.fashion_mnist.load_data')
def load_data():
  """Loads the Fashion-MNIST dataset.

  Returns:
      Tuple of Numpy arrays: `(x_train, y_train), (x_test, y_test)`.

  License:
      The copyright for Fashion-MNIST is held by Zalando SE.
      Fashion-MNIST is licensed under the [MIT license](
      https://github.com/zalandoresearch/fashion-
mnist/blob/master/LICENSE).

  """
  dirname = os.path.join('datasets', 'fashion-mnist’)
  #将原有的地址修改成为 GitHub 提供的 Amazonaws.com 的下载地址
  base = 'http://fashion-mnist.s3-website.eu-central-
1.amazonaws.com/'
  files = [
      'train-labels-idx1-ubyte.gz', 'train-images-idx3-ubyte.gz',
      't10k-labels-idx1-ubyte.gz', 't10k-images-idx3-ubyte.gz'
  ]

  paths = []
  for fname in files:
```

```
    paths.append(get_file(fname, origin=base + fname,
cache_subdir=dirname))

  with gzip.open(paths[0], 'rb') as lbpath:
    y_train = np.frombuffer(lbpath.read(), np.uint8, offset=8)

  with gzip.open(paths[1], 'rb') as imgpath:
    x_train = np.frombuffer(
        imgpath.read(), np.uint8, offset=16).reshape(len(y_train), 28, 28)

  with gzip.open(paths[2], 'rb') as lbpath:
    y_test = np.frombuffer(lbpath.read(), np.uint8, offset=8)

  with gzip.open(paths[3], 'rb') as imgpath:
    x_test = np.frombuffer(
        imgpath.read(), np.uint8, offset=16).reshape(len(y_test), 28, 28)

  return (x_train, y_train), (x_test, y_test)
```

在将这些值输入到神经网络模型之前，需要把 value缩放到0到1的范围，所以每个像素权值需要除以255的预处理，训练集和测试集必须以相同的方式进行。为了验证数据的格式是否正确，以及是否准备好构建和训练网络，让我们显示训练集中的前25个图像，并在每个图像下面显示类名。如下图所示：

接下来，开始训练模型，新建一个模型。Layer 作为神经网络最基础的组成部分，Layers是从训练数据喂到模型中的特征点集合体，大部分深度学习都是用多个 layer 组合的形式，并且这类 layer 都可以配置一定的参数，例如 tf.Keras.layers。

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=10)
```

该模型的第一层tf.keras.layers.Flatten将图像的格式从二维数组（28×28像素）转换为一维数组（28×28=784像素）。把这一层想象成图像中一行行的像素并将它们进行连接。此层没有要学习的参数；它只重新格式化数据。

像素展开后，模型由两个tf.keras.layers.Dense层组成，这些数据Layer紧密相连，或完全相连。第一个 layer有128个节点（Node或神经元）第二层（也是最后一层）是一个10节点的softmax layer层，它返回一个由10个概率得分组成的数组，其总和为1。每个Node包含一个分数，表示当前图像属于10个类之一的概率。

然后开始训练模型，在开始准备训练模型之前，我们需要一些参数设置，也包括了 Compile 的步数。分别说明下参数的意义：

- *Loss function* —This measures how accurate the model is during training. You want to minimize this function to "steer" the model in the right direction. 损失函数方式
- *Optimizer* —This is how the model is updated based on the data it sees and its loss function. 依据损失函数来更新模型的方式
- *Metrics* —Used to monitor the training and testing steps. The following example uses *accuracy*, the fraction of the images that are correctly classified. 分类的概率

这里的损失函数判断方式使用了 adam 算法，具体的算法公式可以参考其他文档。并且使用了`sparse_categorical_crossentropy`作为（Cross Entropy）交叉熵 损失函数的方式。

之后，开始训练模型，TensorFlow 提供了 model.fit 方法用于训练模型。只需要把样本数据和标签，还有步长告诉函数即可开始训练过程，并且在最后会返回模型的识别概率。

```
model.fit(train_images, train_labels, epochs=10)
```

从终端，可以看到大致的训练过程和最后的结果概率：

```
1.14.0
trans image size  60000
trans image label size  60000
WARNING: Logging before flag parsing goes to stderr.
W1107 16:56:47.584466 140734891797952 deprecation.py:506] From
/Users/genesis/Workplace/Python_PyCharm_Workplace/Tensorflow/venv/lib/pytho
n3.7/site-packages/tensorflow/python/ops/init_ops.py:1251: calling
VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype
is deprecated and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to
the constructor
2019-11-07 16:56:47.979904: I
tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU supports
instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
Epoch 1/10
60000/60000 [==============================] - 4s 62us/sample - loss:
0.4962 - acc: 0.8244
Epoch 2/10
60000/60000 [==============================] - 4s 60us/sample - loss:
0.3721 - acc: 0.8654
Epoch 3/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.3346 - acc: 0.8780
Epoch 4/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.3131 - acc: 0.8852
```

```
Epoch 5/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.2954 - acc: 0.8919
Epoch 6/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.2797 - acc: 0.8972
Epoch 7/10
60000/60000 [==============================] - 4s 58us/sample - loss:
0.2689 - acc: 0.9000
Epoch 8/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.2568 - acc: 0.9034
Epoch 9/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.2461 - acc: 0.9084
Epoch 10/10
60000/60000 [==============================] - 4s 59us/sample - loss:
0.2394 - acc: 0.9105
```

每一步的损失和概率都会从终端有相应的返回。之后测试 10,000个测试数据用于判别模型。

```
test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)

print('\nTest accuracy:', test_acc)
```

输出结果：

```
10000/10000 - 0s - loss: 0.3373 - acc: 0.8804

Test accuracy: 0.8804
```

最后，只做一个 predictions，可以使用 model.predict 方法。做这一步之前，先让我们编写两个函数方法用于可视化输出的辅助性函数，以便于显示原始图形和投票结果。

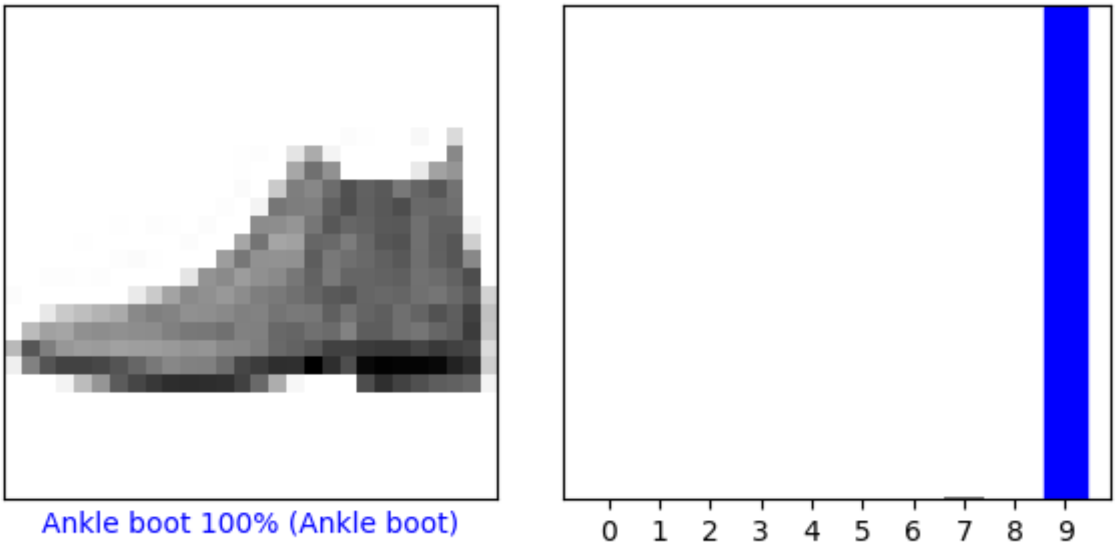
```
#用于绘画图形，并且显示概率
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array, true_label[i],
img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  plt.imshow(img, cmap=plt.cm.binary)
  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'
```

```
  plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                100*np.max(predictions_array),
                                class_names[true_label]),
                                color=color)
# 用于显示标签的直方图
def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array, true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

然后输入一个测试数据，并且将其概率识别信息显示到屏幕上。

```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

最后得到结果:



也可以输出前 5x5的识别结果:

```
num_rows = 5
```

```
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```