人工智能实践：Tensorflow 笔记

第三讲

神经网络搭建八股

本节课目标：分享神经网络的搭建八股，用“六步法”， 不到 20 行代码，写出手写数字识别训练模型。

1　tf.keras 搭建网络八股

1.1 keras 介绍

tf.keras 是 tensorflow2 引入的高封装度的框架，可以用于快速搭建神经网络模型，keras 为支持快速实验而生，能够把想法迅速转换为结果，是深度学习框架之中最终易上手的一个，它提供了一致而简洁的 API，能够极大地减少一般应用下的工作量，提高代码地封装程度和复用性。

Keras 官方文档：

https://tensorflow.google.cn/api_docs/python/tf

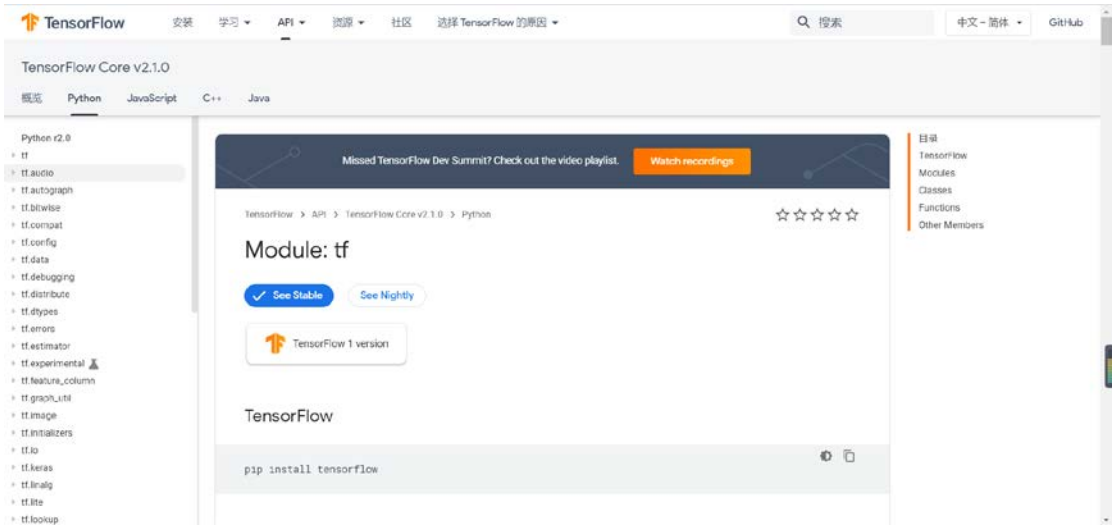
深度学习编程框架中的 API 众多，就算是从业很久的算法工程师也不可能记住所有的 API。由于本课程时间有限，只覆盖了 tensorflow2 系列中的最常用的几个 API，仍然还有很多需要在今后的实践中继续学习，这时我们就需要参考 tensorflow 的官方文档，通过阅读源码和注释的方法学习 API。通常有两种方法，以下将分别介绍。

第一种：在 pycharm 集成开发环境中查看框架源码

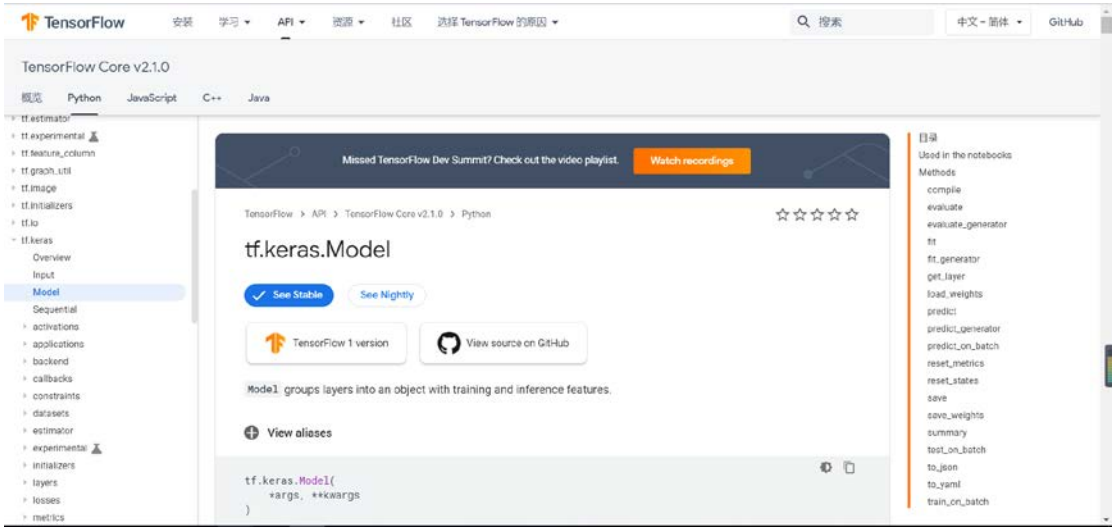
如上图，将鼠标放置**在函数上按住 Ctrl 键**，会显示函数的基本信息，包括封装函数的类，函数入口参数，函数功能等。上图中显示的提示框就是显示出的函数信息，第一行表示函数属于 Model 类，第二三四行列出了函数的参数，第五行说明了函数的功能。可以看到，model.fit()的功能是执行训练过程，是本节课搭建神经网络六部法中十分重要的一步，后面会进一步介绍。

按住 Ctrl 键点击函数会跳转到函数的源代码部分，使用者可以根据源码和注释进一步了解函数的实现方法。

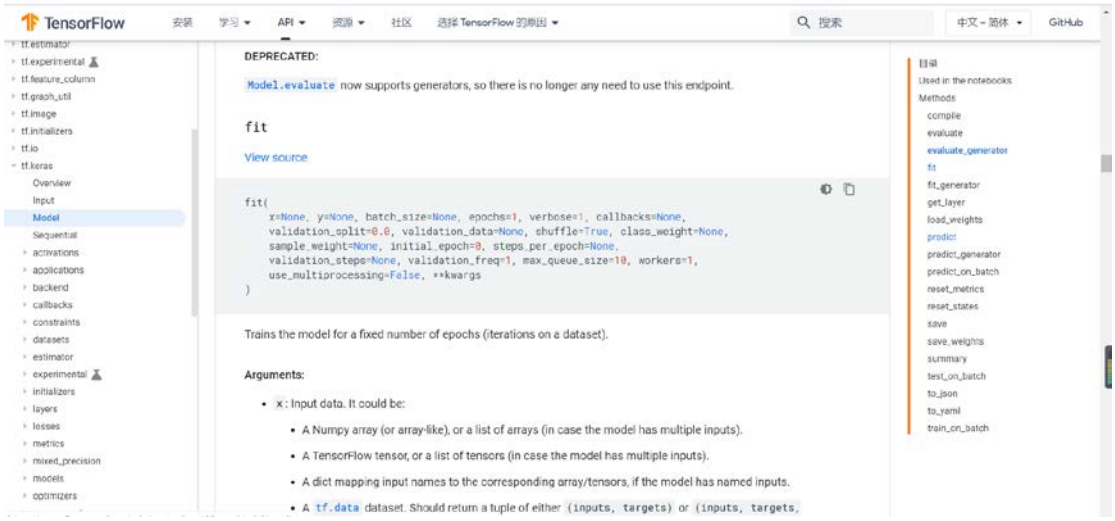
第二种：在 tensorflow 官网中查询函数文档



上图是 tensorflow 官方文档的网站页面。查询时可以通过左边的检索寻找目标函数。以下以查找 model.fit()函数为例。



打开 tf.keras 中的 Model 类，可以看到右方目录列出了 Model 类所包含的函数。



点击 fit()函数可以看到对于函数的介绍，包括输入参数具体介绍，函数功能等。

1.2 搭建神经网络六部法

tf.keras 搭建神经网络六部法

第一步： import 相关模块，如 import tensorflow as tf。

第二步：指定输入网络的训练集和测试集，如指定训练集的输入 x_train 和标签 y_train，测试集的输入 x_test 和标签 y_test。

第三步：逐层搭建网络结构，model = tf.keras.models.Sequential()。 相当于走了一遍前向传播

第四步：在 model.compile()中配置训练方法，选择训练时使用的优化器、损失函数和最终评价指标。

第五步：在 model.fit()中执行训练过程，告知训练集和测试集的输入值和标签、每个 batch 的大小（batchsize）和数据集的迭代次数（epoch）。

第六步：使用 model.summary()打印网络结构，统计参数数目。

1.3 函数用法介绍

——tf.keras.models.Sequential()

Sequential 函数是一个容器，描述了神经网络的网络结构，在 Sequential 函数的输入参数中描述从输入层到输出层的网络结构。

如：

拉直层：tf.keras.layers.Flatten() **只是形状转换**

拉直层可以变换张量的尺寸，把输入特征拉直为一维数组，是不含计算参数的层。

全连接层：tf.keras.layers.Dense(神经元个数,

activation=”激活函数”,

kernel_regularizer=”正则化方式”)

其中：

activation（字符串给出）可选 relu、softmax、sigmoid、tanh 等

kernel_regularizer 可选 tf.keras.regularizers.l1()、

tf.keras.regularizers.l2()

卷积层：tf.keras.layers.Conv2D(filter = 卷积核个数,

kernel_size = 卷积核尺寸,

strides = 卷积步长,

padding = “valid” or “same”)

循环层

LSTM 层：tf.keras.layers.LSTM()。

本章只使用拉直层和全连接层，卷积层和循环神经网络层将在之后的章节介绍。

——Model.compile(optimizer = 优化器,

loss = 损失函数,

metrics = [“准确率”])

Compile 用于配置神经网络的训练方法，告知训练时使用的优化器、损失函数和准确率评测标准。

其中：

optimizer 可以是字符串形式给出的优化器名字，也可以是函数形式，使用函数形式可以设置学习率、动量和超参数。

可选项包括：

‘sgd’ or tf.optimizers.SGD(lr=学习率,

建议入门时，先使用左边这些字符串形式的优化器名字，等掌握了整个框架后，可以通过TensorFlow官网查询这些函数的具体用法，调节超参数

decay=学习率衰减率,

momentum=动量参数)

‘adagrad’ or tf.keras.optimizers.Adagrad(lr=学习率,

decay=学习率衰减率)

‘adadelta’ or tf.keras.optimizers.Adadelta(lr=学习率,

decay=学习率衰减率)

‘adam’ or tf.keras.optimizers.Adam (lr=学习率,

decay=学习率衰减率)

Loss 可以是字符串形式给出的损失函数的名字，也可以是函数形式。

可选项包括：

‘mse’ or tf.keras.losses.MeanSquaredError()

‘sparse_categorical_crossentropy

是否是原始输出，即未经过概率分布的输出

or tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False)

损失函数常需要经过 softmax 等函数将输出转化为概率分布的形式。from_logits 则用来标注该损失函数是否需要转换为概率的形式，取 False 时表示转化为概率分布，取 True 时表示没有转化为概率分布，直接输出。

Metrics 标注网络评测指标。

可选项包括：

标签　　网络输出结果

‘accuracy’：y_和 y 都是数值，如 y_=[1] y=[1]。

‘categorical_accuracy’：y_和 y 都是以独热码和概率分布表示。

如 y_=[0, 1, 0], y=[0.256, 0.695, 0.048]。

‘sparse_ categorical_accuracy’：y_是以数值形式给出，y 是以~~独热码~~概率分布形式给出。

如 y_=[1],y=[0.256, 0.695, 0.048]。

每次喂入神经网络的样本数

——model.fit(训练集的输入特征， 训练集的标签， batch_size, epochs,

要迭代多少次数据集

validation_data和validation_split二者选一

validation_data = (测试集的输入特征，测试集的标签)，

validataion_split = 从测试集划分多少比例给训练集，

validation_freq = 测试的 epoch 间隔次数)

每多少次epoch迭代，使用测试集验证一次结果

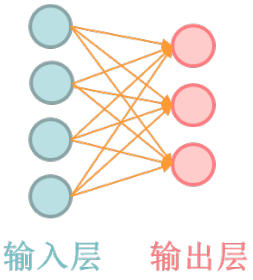
fit 函数用于执行训练过程。

——model.summary()

summary 函数用于打印网络结构和参数统计

全连接

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                multiple                  15
=================================================================
Total params: 15
Trainable params: 15
Non-trainable params: 0
_________________________________________________________________
```



上图是 model.summary()对鸢尾花分类网络的网络结构和参数统计，对于一个输入为 4 输出为 3 的全连接网络，共有 15 个参数。

2 iris 数据集代码复现

```python
# -*- coding: UTF-8 -*-
import tensorflow as tf
import os
from sklearn import datasets
import numpy as np

x_train = datasets.load_iris().data
y_train = datasets.load_iris().target

np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)

model = tf.keras.models.Sequential([
                                    tf.keras.layers.Dense(3, activation='sigmoid')
                                    ])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

model.fit(x_train,y_train, epochs=500, validation_freq=20,validation_split=0.2)
model.summary()
```

第一步：import 相关模块：

```
import tensorflow as tf

from sklearn import datasets
```

```
import numpy as np
```

第二步：指定输入网络地训练集和测试集：

```
x_train = datasets.load_iris().data
y_train = datasets.load_iris().target
```

其中测试集的输入特征 x_test 和标签 y_test 可以像 x_train 和 y_train 一样直接从数据集获取，也可以如上述在 fit 中按比例从训练集中划分，本例选择从训练集中划分，所以只需加载 x_train，y_train 即可。

```
np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)

tf.random.set_seed(116)
```

以上代码实现了数据集的乱序。

第三步：逐层搭建网络结构：

```
model = tf.keras.models.Sequential([
        tf.keras.layers.Dense(3, activation='softmax',
        kernel_regularizer=tf.keras.regularizers.l2())
])
```

如上所示，本例使用了单层全连接网络，第一个参数表示神经元个数，第二个参数表示网络所使用的激活函数，第三个参数表示选用的正则化方法。

使用 Sequential 可以快速搭建网络结构，但是如果网络包含跳连等其他复杂网络结构，Sequential 就无法表示了。这就需要使用 class 来声明网络结构。

```
class MyModel(Model):
        def __init__(self):
                super(MyModel, self).__init__()
                //初始化网络结构
```

```
        def call(self, x):
                y = self.d1(x)
                return y
```

使用 class 类封装网络结构，如上所示是一个 class 模板，MyModel 表示声明的神经网络的名字，括号中的 Model 表示创建的类需要继承 tensorflow 库中的 Model 类。类中需要定义两个函数，__init__()函数为类的构造函数用于初始化类的参数，spuer(MyModel,self).__init__()这行表示初始化父类的参数。之后便可初始化网络结构,搭建出神经网络所需的各种网络结构块。call()函数中调用__init__()函数中完成初始化的网络块，实现前向传播并返回推理值。使用 class 方式搭建鸢尾花网络结构的代码如下所示。

```
class IrisModel(Model):
        def __init__(self):
                super(IrisModel, self).__init__()
                self.d1 = Dense(3, activation='sigmoid',
kernel_regularizer=tf.keras.regularizers.l2())

        def call(self, x):
                y = self.d1(x)
                return y
```

鸢尾花分类的单层网络是含有三个神经元的全连接

d1是给这一层起的名字，每一层都用self.引导

搭建好网络结构后只需要使用 Model=MyModel()构建类的对象，就可以使用该模型了。

和iris_sequential相比只改了两块：
1、添加了Model模块（2-3行）
2、定义了IrisModel类（16-25行）

```python
# -*- coding: UTF-8 -*-
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras import Model
from sklearn import datasets
import numpy as np

x_train = datasets.load_iris().data
y_train = datasets.load_iris().target

np.random.seed(116)
np.random.shuffle(x_train)
np.random.seed(116)
np.random.shuffle(y_train)

class IrisModel(Model):
    def __init__(self):
        super(IrisModel, self).__init__()
        self.d1 = Dense(3, activation='sigmoid')

    def call(self, x):
        y=self.d1(x)
        return y

model = IrisModel()


model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

model.fit(x_train,y_train, epochs=500, validation_freq=20,validation_split=0.2)
model.summary()
```

对比使用 Sequential()方法和 class 方法，有两点区别：

①import 中添加了 Model 模块和 Dense 层、Flatten 层。

②使用 class 声明网络结构，model = IrisModel()初始化模型对象。

第四步：在 model.compile()中配置训练方法：

```
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.1),
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False,
              metrics=['sparse_categorical_accuracy'])
```

如上所示，本例使用 SGD 优化器，并将学习率设置为 0.1，选择 SparseCategoricalCrossentrop 作为损失函数。由于神经网络输出使用了 softmax 激活函数，使得输出是概率分布，而不是原始输出，所以需要将 from_logits 参数设置为 False。鸢尾花数据集给的标签是 0，1，2 这样的数值，而网络前向传播的输出为概率分布，所以 metrics 需要设置为 sparse_categorical_accuracy。

第五步：在 model.fit()中执行训练过程：

训练集输入特征　　训练时一次喂入神经网络多少组数据　　从训练集中选择20%的数据作为测试集

```
model.fit(x_train,y_train,batch_size=32,epochs=500, validation_split
= 0.2,validation_freq=20)
```

训练集标签　　数据集迭代循环多少次

每迭代20次训练集要在测试集中验证一次准确率

在 fit 中执行训练过程,x_train,y_train 分别表示网络的输入特征和标签，batch_size 表示一次喂入神经网络的数据量，epochs 表示数据集的迭代次数 validation_split 表示数据集中测试集的划分比例,validation_freq 表示每迭代 20 次在测试集上测试一次准确率。

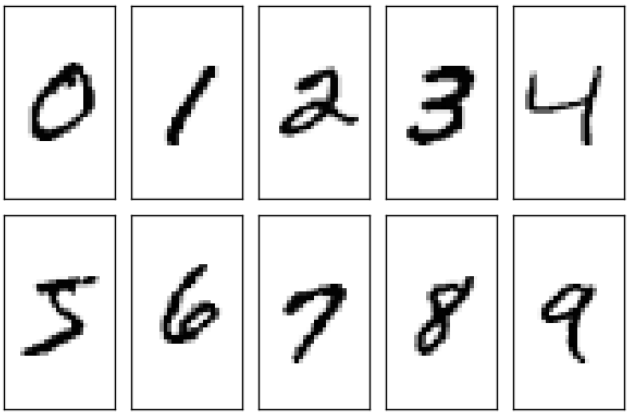
第六步：使用 model.summary()打印网络结构，统计参数数目：

```
model.summary()
```

3 MNIST 数据集

3.1 介绍

MNIST 数据集一共有 7 万张图片，是 28×28 像素的 0 到 9 手写数字数据集，其中 6 万张用于训练，1 万张用于测试。每张图片包括 784（28×28）个像素点，使用全连接网络时可将 784 个像素点组成长度为 784 的一维数组，作为输入特征。数据集图片如下所示。



3.2 导入数据集

keras 函数库中提供了使用 mnist 数据集的接口，代码如下所示，可以使用 load_data()直接从 mnist 中读取测试集和训练集。

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

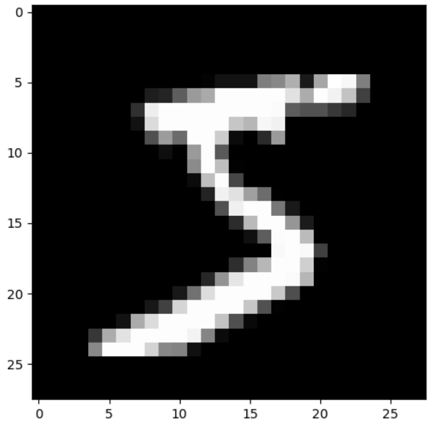
输入全连接网络时需要先将数据拉直为一维数组，把 784 个像素点的灰度值作为输入特征输入神经网络。

```
tf.keras.layers.Flatten()
```

使用 plt 库中的两个函数可视化训练集中的图片。

```
plt.imshow(x_train[0],cmap='gray')
plt.show()
```

把训练集中的第一个样本x_train[0]可视化出来



使用 print 打印出训练集中第一个样本以二位数组的形式打印出来，如下所示。

```
print("x_train[0]:",x_train[0])
```

用print函数把训练集中第一个样本的输入特征打印出来，
是一个28*28的二维数组，是手写数字5的28行28列个像素值
（0表示纯黑色，255表示纯白色）

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136 175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253 225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251  93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119  25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253 150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252 253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249 253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253 253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253 250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201  78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

打印出第一个样本的标签，为 5。

```
print("y_train[0]:",y_train[0])
y_train[0]:5
```

打印出测试集样本的形状，共有 10000 个 28 行 28 列的三维数据。

```
print("x_test.shape:" x_test.shape)
x_test.shape:(10000,28,28)
```

3.3 训练 MNIST 数据集

使用 Sequential 实现手写数字识别

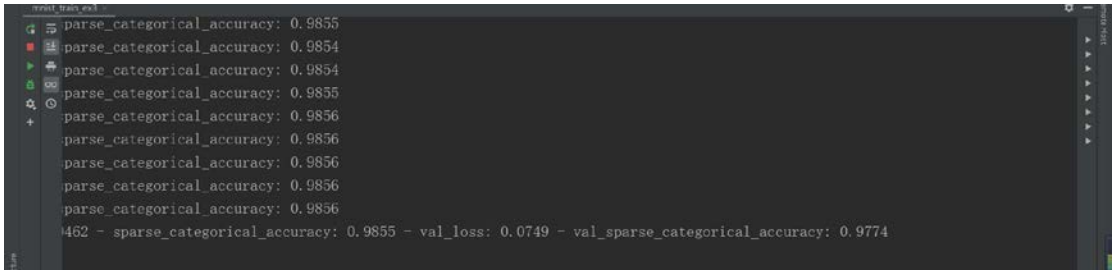
```python
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
                                    tf.keras.layers.Flatten(input_shape=(28, 28)),
                                    tf.keras.layers.Dense(128, activation='relu'),
                                    tf.keras.layers.Dense(10, activation='softmax')
                                    ])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

model.fit(x_train, y_train, epochs=10, validation_data=(x_test,y_test),validation_freq=2)
model.summary()
```

使用 class 实现手写数字识别

```python
import tensorflow as tf
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras import Model

mnist = tf.keras.datasets.mnist
(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

class MnistModel(Model):
    def __init__(self):
        super(MnistModel, self).__init__()
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.flatten(x)
        x = self.d1(x)
        y = self.d2(x)
        return y

model = MnistModel()


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['sparse_categorical_accuracy'])

model.fit(x_train, y_train, epochs=10, validation_data=(x_test,y_test),validation_freq=2)
model.summary()
```

值得注意的是训练时需要将输入特征的灰度值归一化到[0,1]区间，这可以使网络更快收敛。

训练时每个 step 给出的是训练集 accuracy 不具有参考价值，有实际评判价值的是 validation_freq 中设置的隔若干轮输出的测试集 accuracy。如下图所示



4 Fashion_mnist 数据集

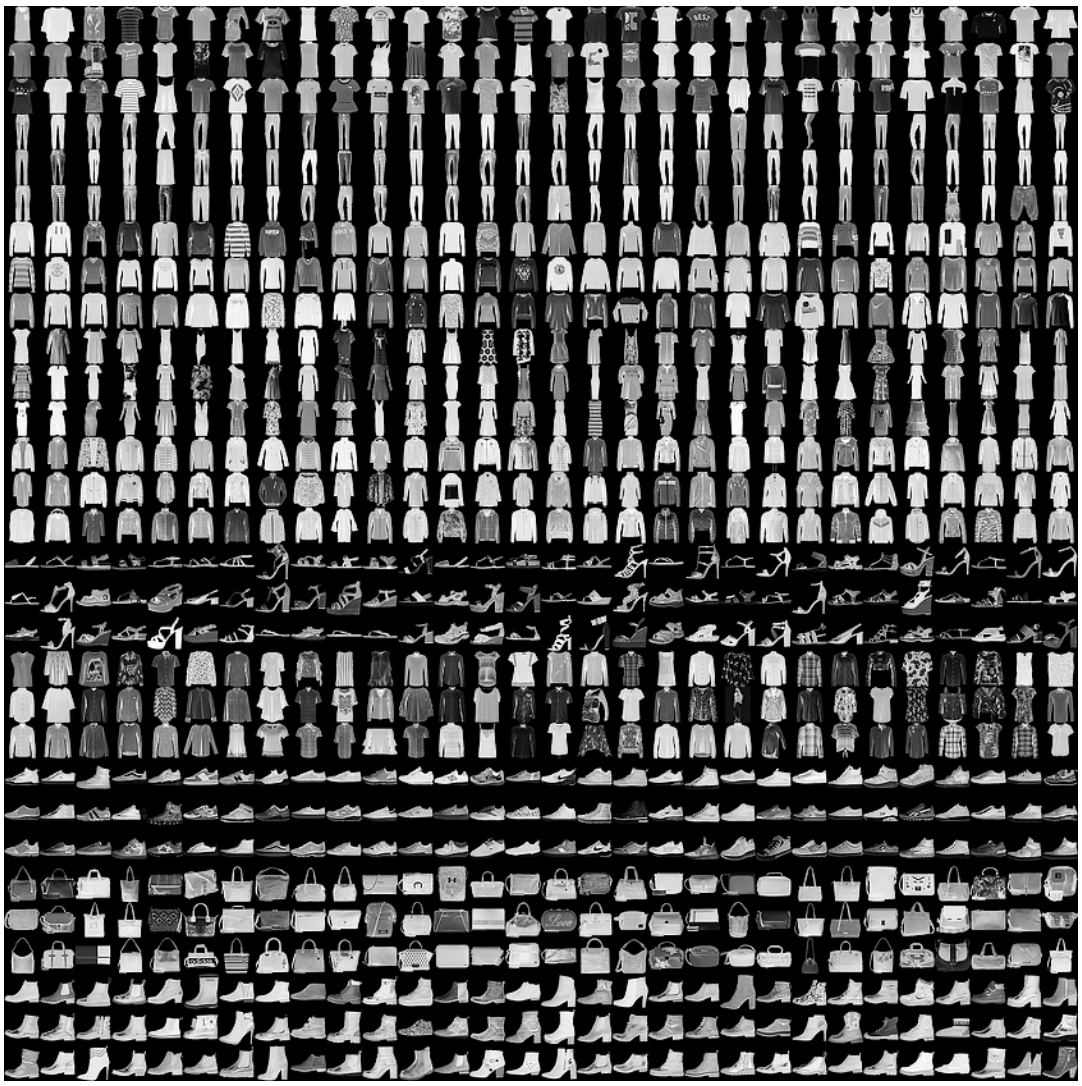
Fashion_mnist 数据集具有 mnist 近乎所有的特征，包括 60000 张训练图片和 10000 张测试图片，图片被分为十类，每张图像为 28×28 的分辨率。

像素点的灰度值数据

类别如下所示：

Label	Description
0	T恤（T-shirt/top）
1	裤子（Trouser）
2	套头衫（Pullover）
3	连衣裙（Dress）
4	外套（Coat）
5	凉鞋（Sandal）
6	衬衫（Shirt）
7	运动鞋（Sneaker）
8	包（Bag）
9	靴子（Ankle boot）

图片事例如下所示：



可以使用load_data()直接从FASHION数据集中读取训练集和测试集

由于 Fashion_mnist 数据集和 mnist 数据集具有相似的属性，所以对于 mnist 只需讲 mnist 数据集的加载换成 Fashion_mnist 就可以训练 Fashion 数据集了。

代码如下所示：

```python
import tensorflow as tf

mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['sparse_categorical_accuracy'])

history = model.fit(x_train, y_train, batch_size=32, epochs=5, validation_data=(x_test, y_test), validation_freq=1)
model.summary()

```