## **TensorFlow处理FashionMnist—主要代码说明+神经网络流程**

### 第一步：定义网络

#### 代码

· 创建 Sequential 类的实例：然后按照层的连接顺序创建所需要的网络层。

|  |
| --- |
| model = keras.Sequential([     keras.layers.Flatten(input\_shape=(28, 28)), #Flatten层用来将输入“压平”，即把多维的输入一维化，常用在从卷积层到全连接层的过渡。     keras.layers.Dense(128, activation='relu'), #添加全连接层，输出空间维度（节点）为128，激活函数为relu，作用是分类     keras.layers.Dense(10) #添加全连接层，输出空间维度为10，激活函数为softmax ]) |

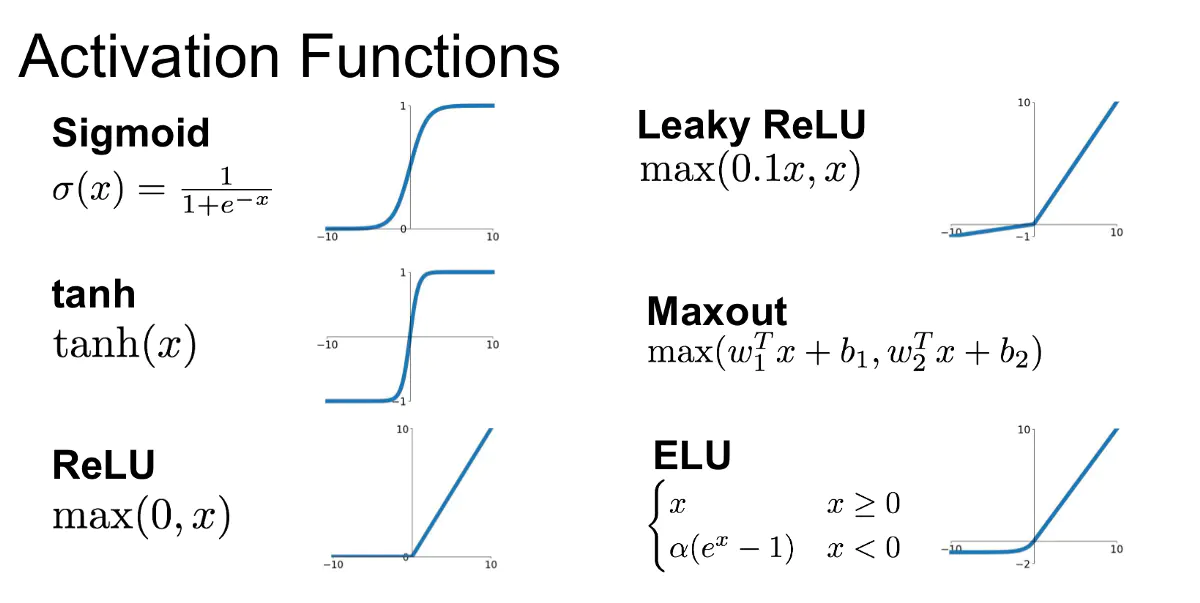
tf.keras.layers.Flatten 将图像格式从二维数组转换成一维数组（28 x 28 = 784 像素）。

两个 tf.keras.layers.Dense 层的序列。它们是密集连接或全连接神经层。

第一个 Dense 层有 128 个节点（或神经元）。

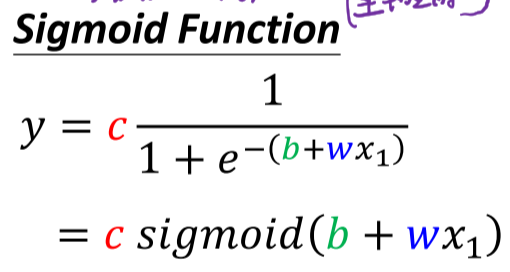
第二个 Dense 层会返回一个长度为 10 的 logits 数组。每个节点都包含一个得分，用来表示当前图像属于 10 个类中的哪一类。

#### 激活函数



* **Sigmoid函数**：下图中的蓝图中的曲线。

非线性的Sigmoid函数对中央区的信号增益较大，对两侧区的信号增益小，在信号的特征空间映射上，有很好的效果

* ReLu函数：



* SoftMax函数——将预测结果转化为10个模型的概率（每个输出对应一个神经元



1. softmax第一步就是将模型的预测结果转化到指数函数上，这样保证了概率的非负性。
2. 归一化处理。方法就是将转化后的结果除以所有转化后结果之和，可以理解为转化后结果占总数的百分比。这样就得到近似的概率。

### 第二步：编译网络

编译是一个高效的步骤。它会将我们定义的层序列通过一系列高效的矩阵转换，根据 Keras 的配置转换成能在 GPU 或 CPU 上执行的格式。

需要专门针对网络的训练设定一些参数，设定训练网络使用的优化算法 以及用于评价网络通过优化算法最小化结果的损失函数。

|  |
| --- |
| model.compile(optimizer='adam',               loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),               metrics=['accuracy']) |

#### 损失函数

* Fashion-Minst数据集为什么只用categorical\_crossentropy损失函数？

当使用categorical\_crossentropy损失函数时，标签应为多类模式，Fashion-Minst有10个类别，每一个样本的标签应该是一个10维的向量，该向量在对应有值的索引位置为1其余为0。

交叉熵，一般搭配softmax激活函数使用

sparse的含义是，真实的标签值 y\_true 可以直接传入 int 类型的标签类别

#### 优化算法

Adam优化算法

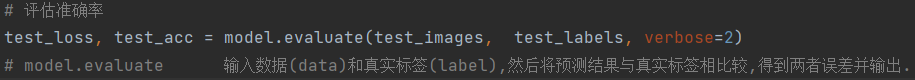
* AdaGrad：能够对每个不同的参数调整不同的学习率，对频繁变化的参数以更小的步长进行更新，而稀疏的参数以更大的步长进行更新。
* RMSProp：能够在不稳定（Non-Stationary）的目标函数情况下进行很好地收敛。能够克服AdaGrad梯度急剧减小的问题，尤其在不稳定的目标函数下表现良好。

Adam吸收了Adagrad（自适应学习率的梯度下降算法）和动量梯度下降算法的优点，既能适应稀疏梯度（即自然语言和计算机视觉问题），又能缓解梯度震荡的问题

### 第三步：训练网络

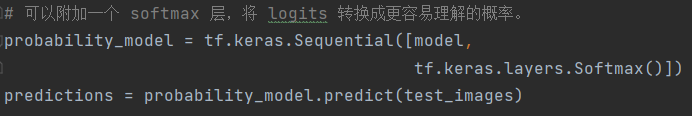
|  |
| --- |
| model.fit( 训练集的输入特征，  训练集的标签，  batch\_size, #每一个batch的大小  epochs, #迭代次数  validation\_data = (测试集的输入特征，测试集的标签），  validation\_split = 从测试集中划分多少比例给训练集，  validation\_freq = 测试的epoch间隔数） |

### 第四步：评估与预测

****

基于训练集的accuracy评估是没有用的，因为模型的参数本身就是基于这些数据训练得到的，所以要用模型还没有见过的测试集去测试，这将提供网络在未来对没有见过的数据进行预测的性能时的估测。

### 第五步：进行预测



* + 如果对训练后的模型的性能满意的话，就能用它来对新的数据做预测。

model.evaluate   需要真实标签，因为需要比较预测结果与真实标签的误差

model.predict   不需要真实标签，只是单纯输出预测结果，全程不需要标签的参与。

### 全连接神经网络缺点

（1）首先将图像展开为向量会丢失空间信息；

（2）其次参数过多效率低下，训练困难；

（3）同时大量的参数也很快会导致网络过拟合。

## 优化：卷积神经网CNN模型

卷积神经网络主要由这几类层构成：输入层、卷积层，ReLU层、池化（Pooling）层和全连接层（全连接层和常规神经网络中的一样）。模型结构如下：

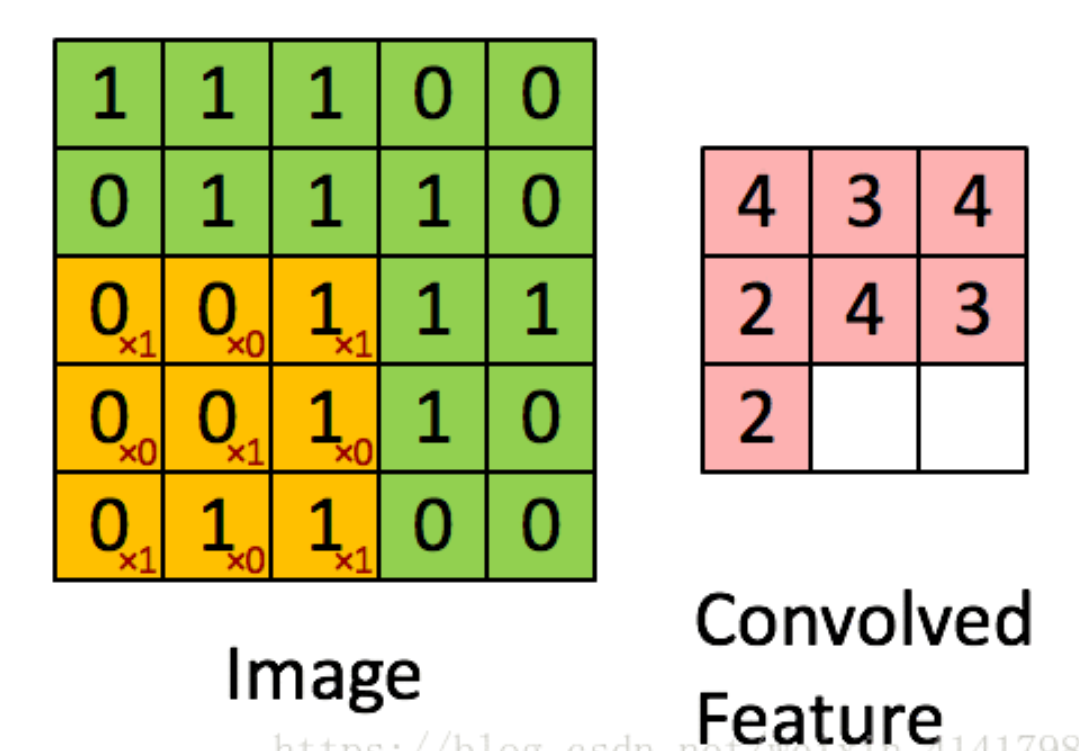
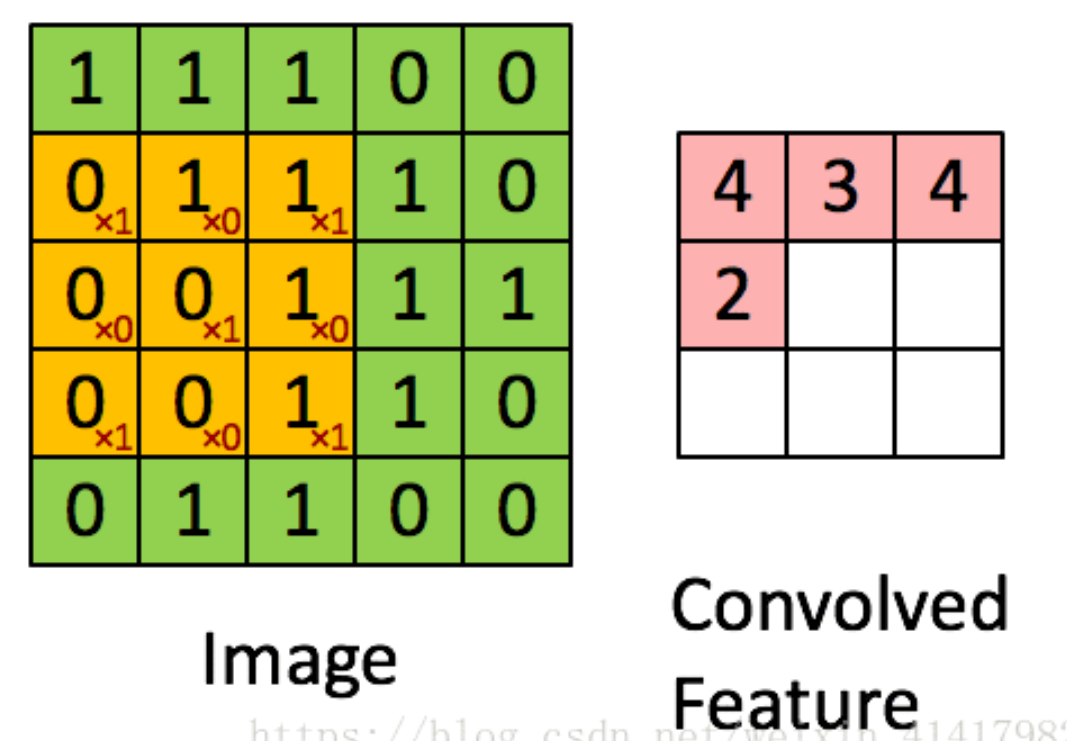
|  |
| --- |
| Model: "sequential"  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Layer (type) Output Shape Param #  =================================================================  conv2d (Conv2D) (None, 26, 26, 64) 640  max\_pooling2d (MaxPooling2D (None, 13, 13, 64) 0  )  conv2d\_1 (Conv2D) (None, 11, 11, 64) 36928  max\_pooling2d\_1 (MaxPooling (None, 5, 5, 64) 0  2D)  flatten (Flatten) (None, 1600) 0  dense (Dense) (None, 128) 204928  dense\_1 (Dense) (None, 10) 1290  =================================================================  Total params: 243,786  Trainable params: 243,786  Non-trainable params: 0 |

### 卷积层

卷积有助于我们找到特定的局部图像特征（如边缘）

**卷积核Kernal/filter：**每个卷积核具有长宽深三个维度，长宽为（3，3）的矩阵，每九个像素得到一个与卷积核矩阵相乘的解。深度为RBG=3，三个矩阵算数求和。

**Feature map：**在输入层，如果是灰度图片，那就只有一个feature map；如果是彩色图片，一般就是一个卷积核得到3个feature map（红绿蓝），算数求和得到新的featuremap。



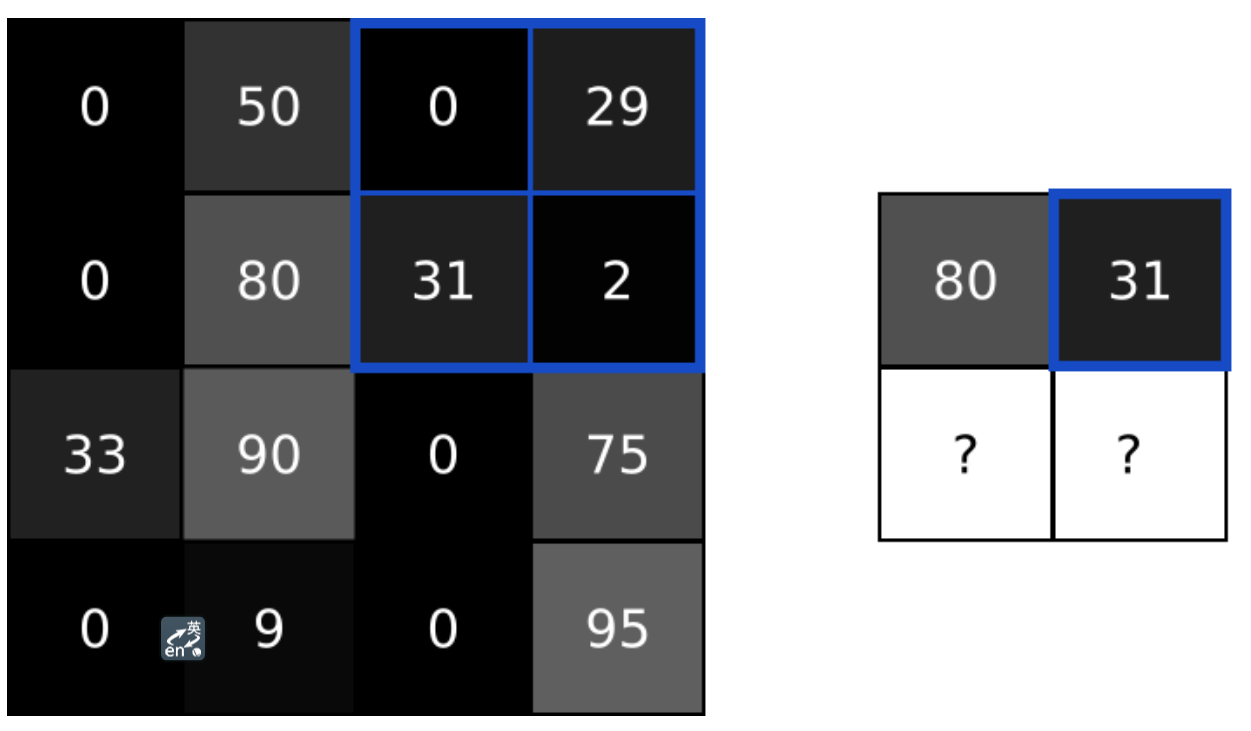
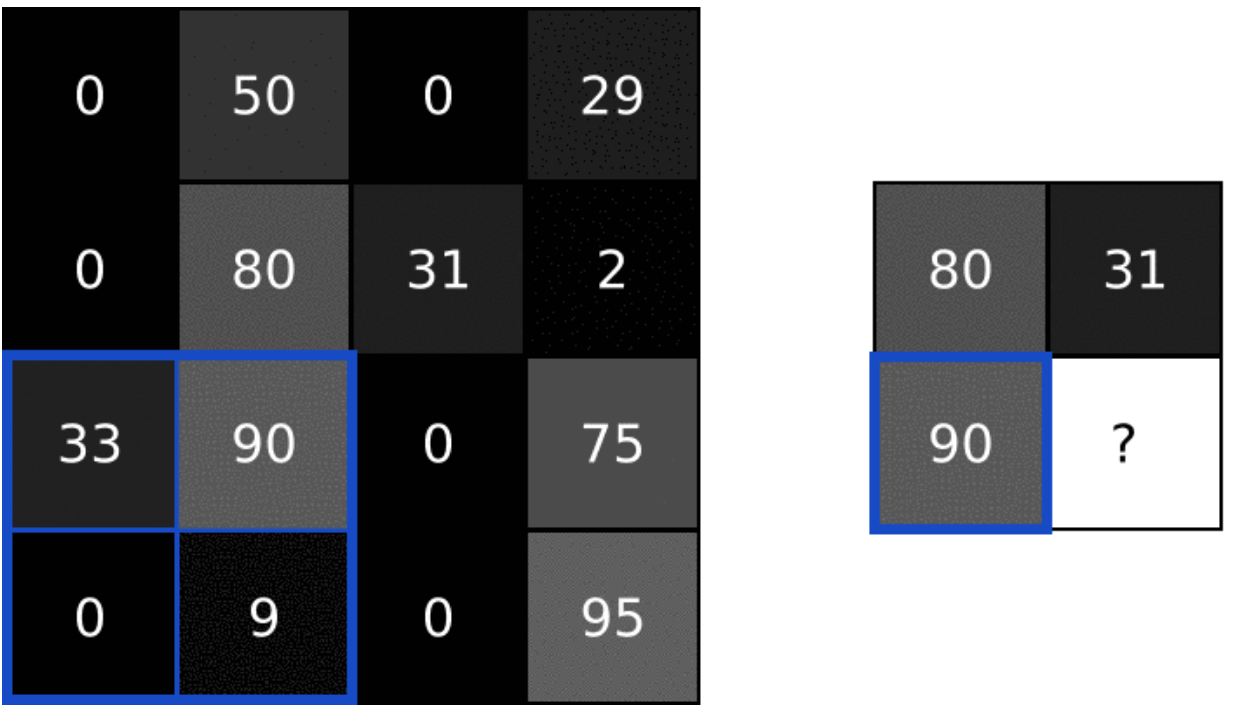
**卷积核个数Filter=64：**64个卷积核，3×3×3（RGB）每个卷积核分别与原图卷积得到一个featuremap

### 池化层

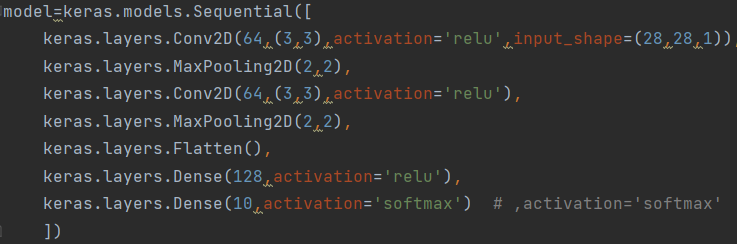
空间池化的主要目的是降维，在保持原有空间特征的基础上最大限度将数组的维度变小。空间池化有下面几种方式：最大化、平均化、加和等等。

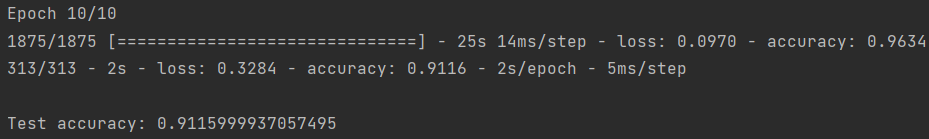
* 图像中的相邻像素倾向于具有相似的值，因此通常卷积层相邻的输出像素也具有相似的值。这意味着，卷积层输出中包含的大部分信息都是冗余的。如果我们使用边缘检测滤波器并在某个位置找到强边缘，那么我们也可能会在距离这个像素1个偏移的位置找到相对较强的边缘。但是它们都一样是边缘，我们并没有找到任何新东西。

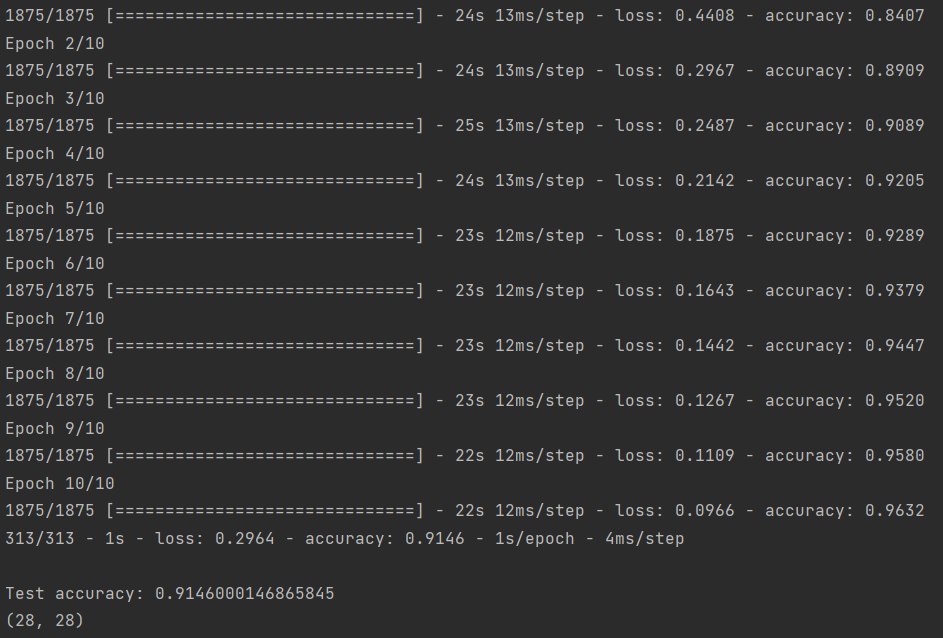
如最大池化就是在每个2x2的空间邻域取出最大值。Maxpooling如下

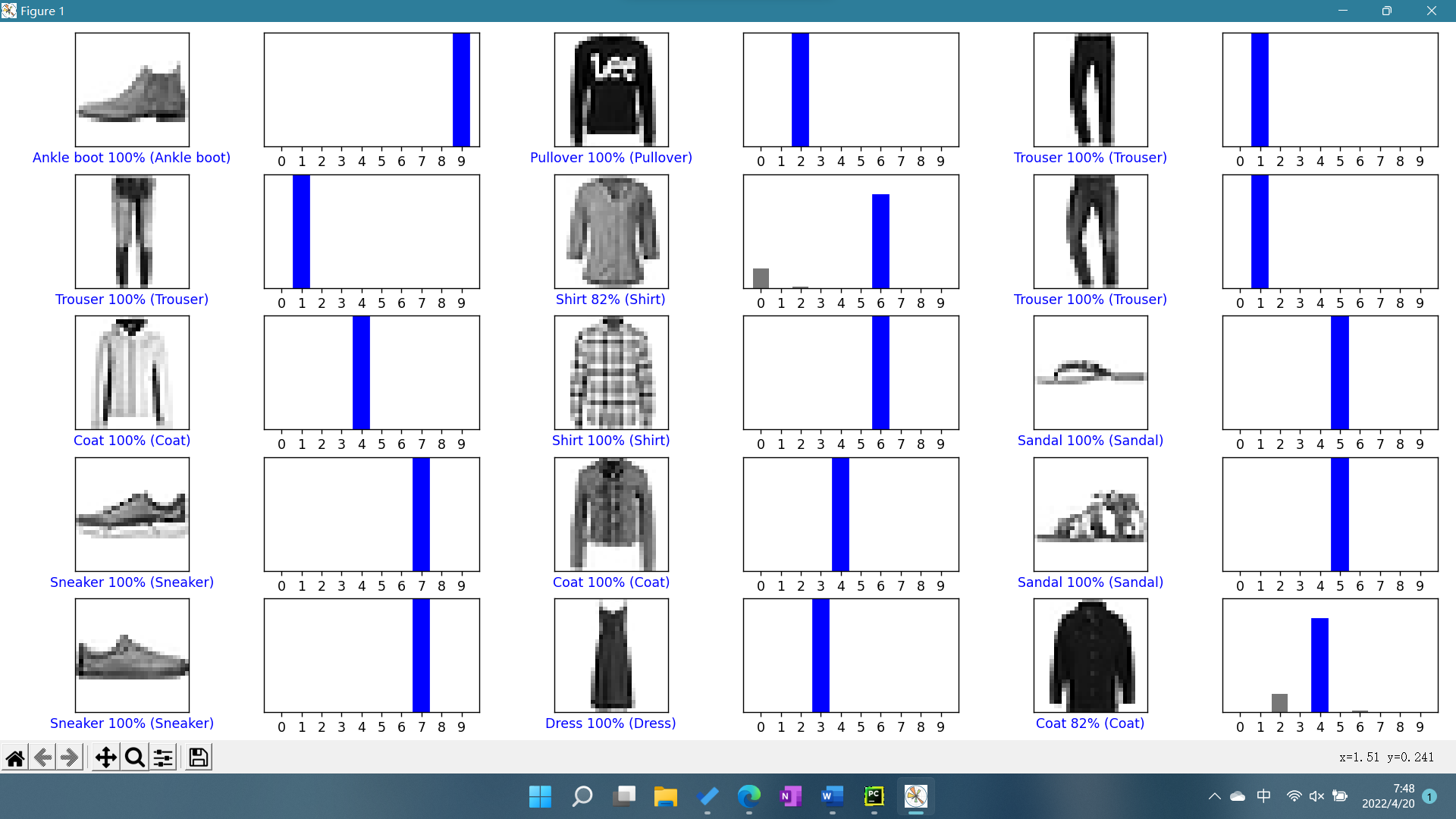
 

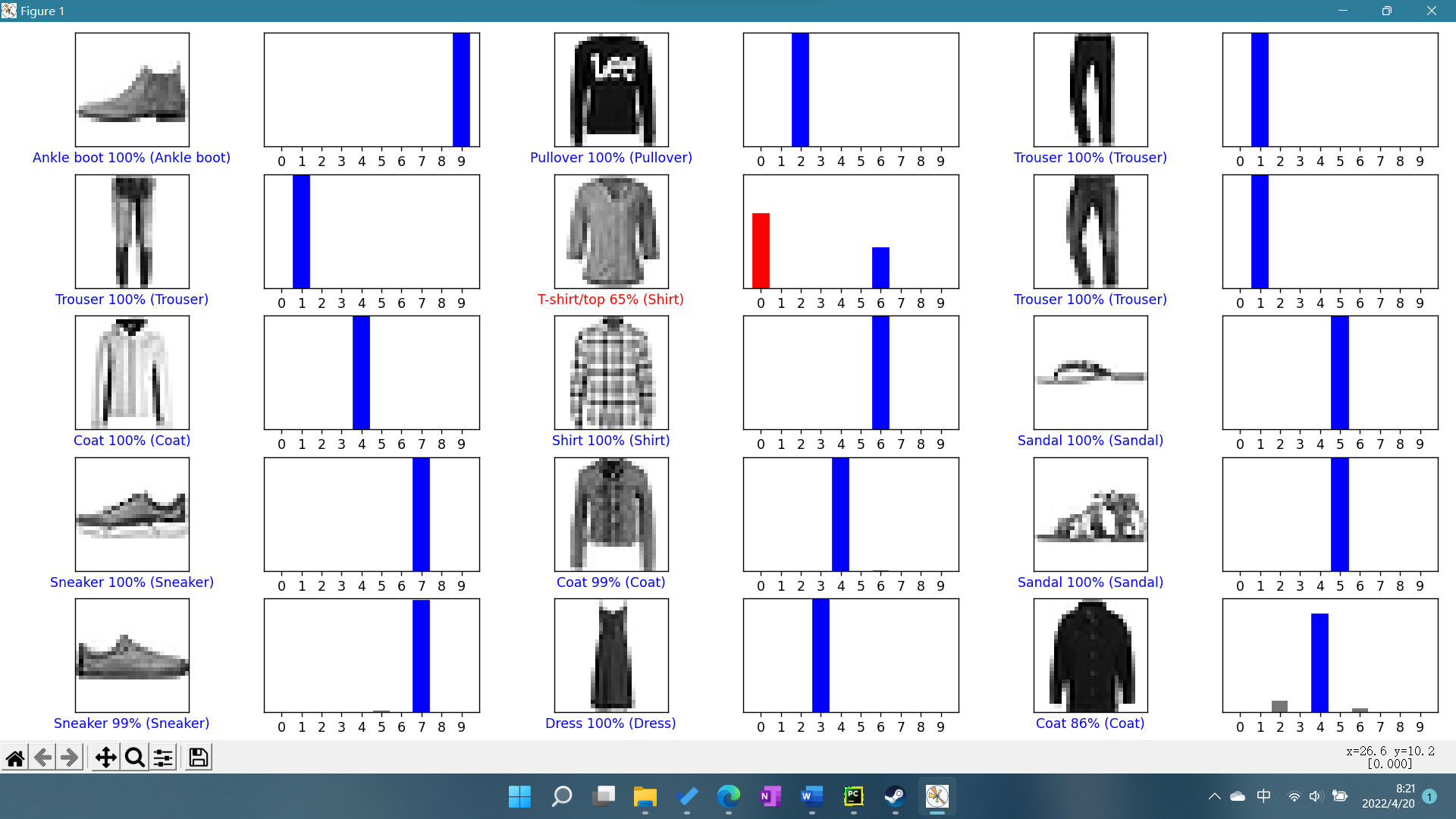
### 定义网络代码如下

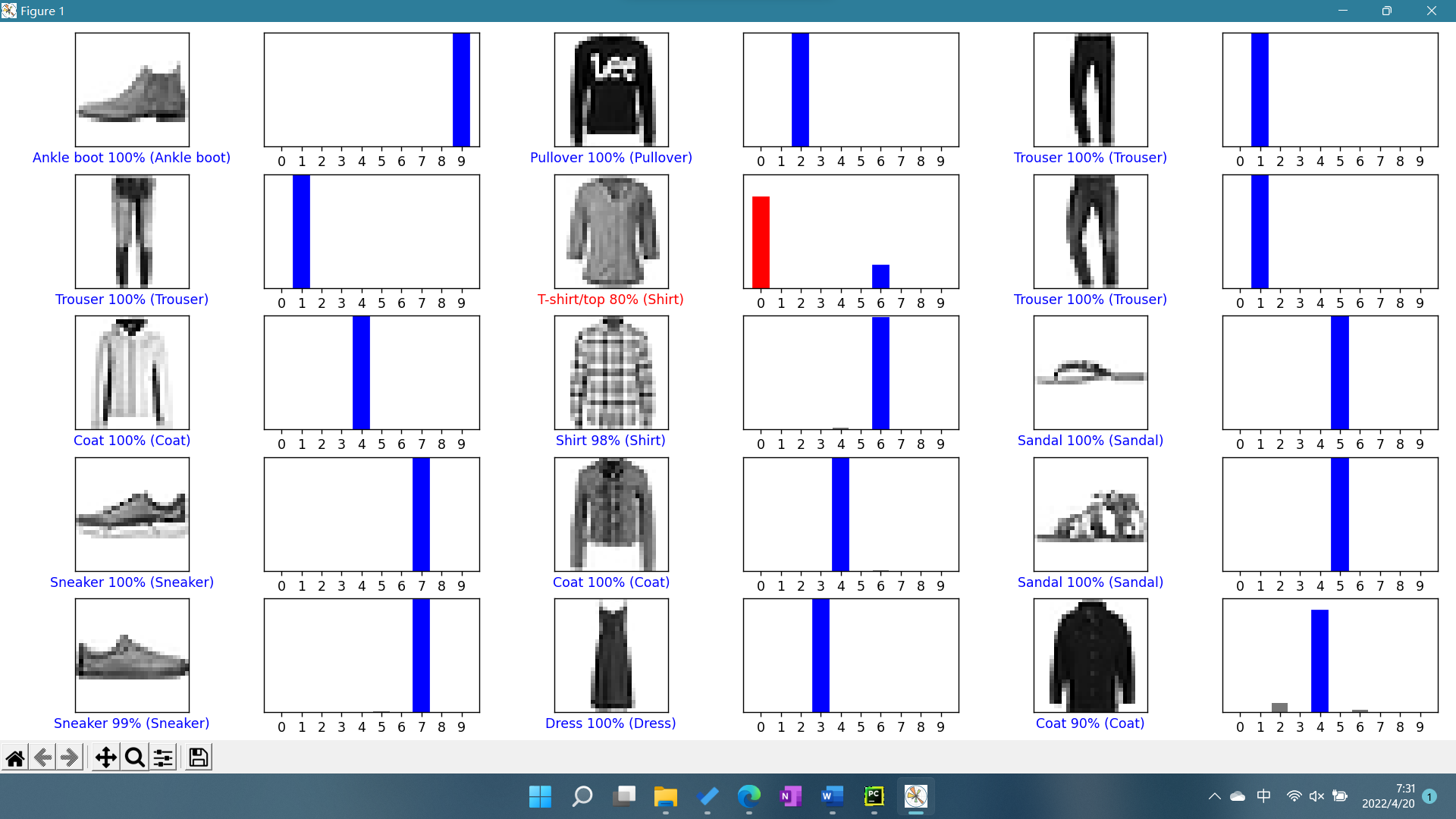












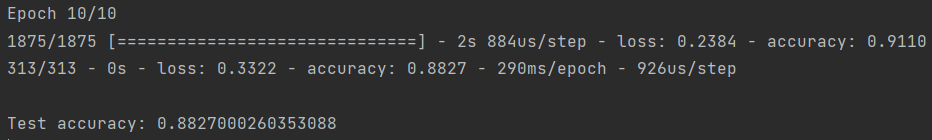
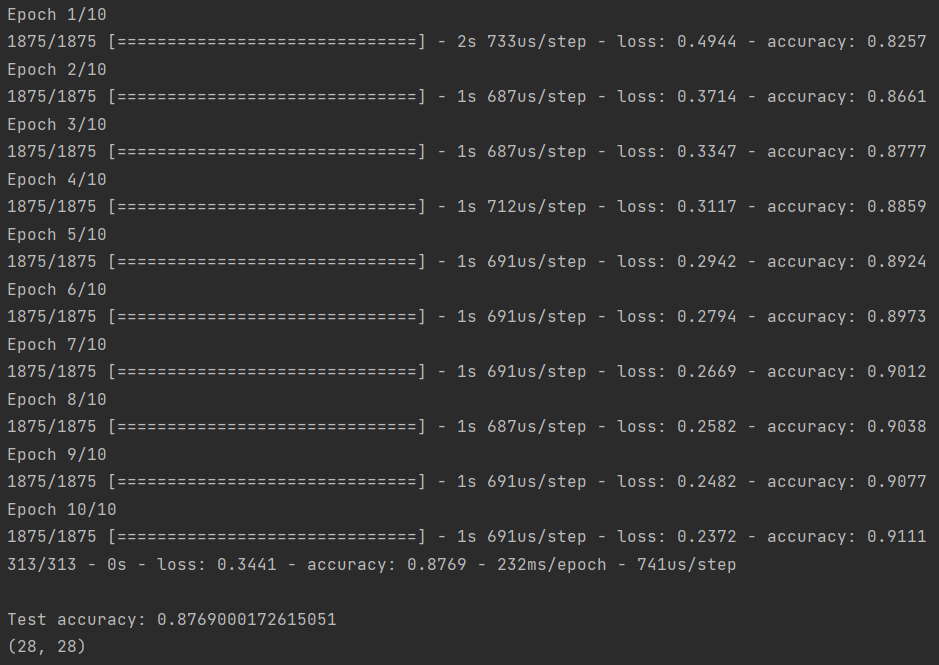
## 优化：EarlyStopping防止过拟合——Callbacks函数

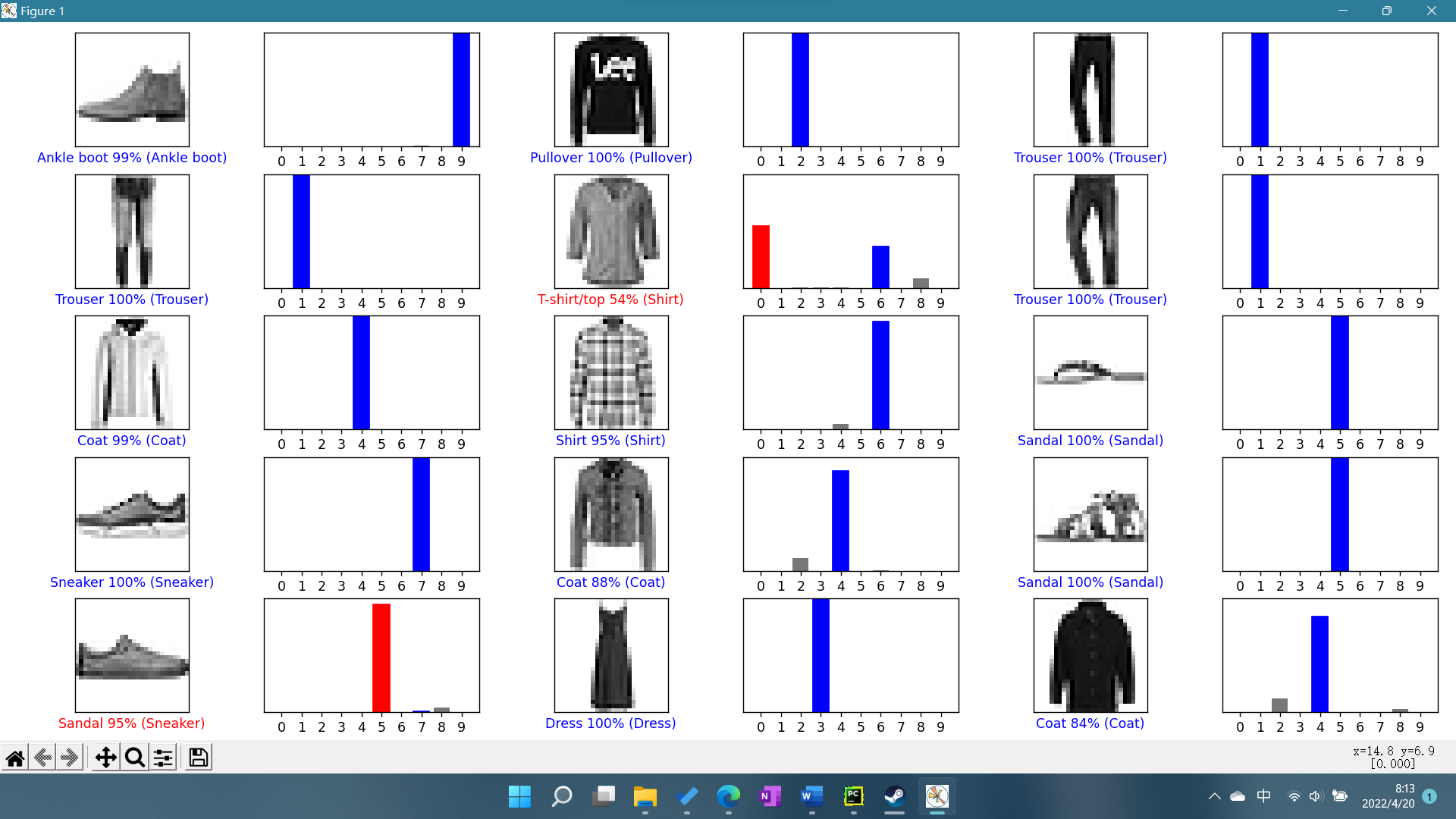
|  |
| --- |
| class Callback1(tf.keras.callbacks.Callback):  def on\_epoch\_end(self,epoch,logs={}):  if(logs.get('loss')<0.3):  self.model.stop\_training=True  ...  model.fit(train\_images,train\_labels,epochs=10,callbacks=Callback1()) |

使用callback防止过拟合进行优化，提前结束训练

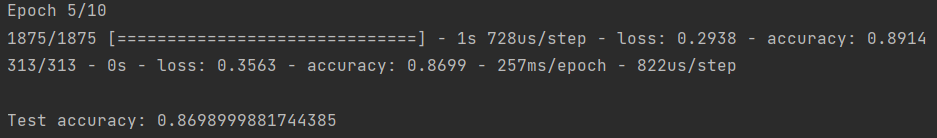
## 实验分析

### 全连接

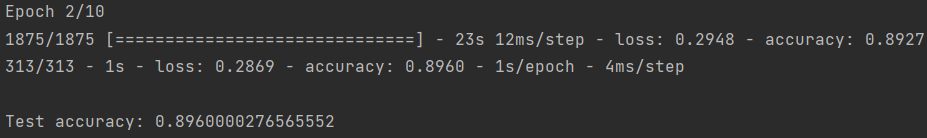




加入callback后，loss<0.3则停止训练



将参数调整为Loss<0.2，可以看到test accuracy变高了



### CNN

