数据挖掘

课程设计报告

设计题目：服装分类

专　　业　信息与计算科学

班　　级　2018 级 2 班

学　　生　曾晶晶、汪璐

学　　号 631810040303、641810040321

指导教师　韩逢庆

起止时间　2021/07/01~2021/07/09

设计报告	优	良	中	及格	不及格
平时表现	优	良	中	及格	不及格
程序演示	优	良	中	及格	不及格
综合成绩					

2021 年 7 月

一、背景与挖掘目标

在图像分类学习中，MNIST 数据集常用来作为入门教学的数据集。但是 MNIST 数据集对于现在的神经网络来说过于简单，模型的精度达到了饱和，几乎没有提升的空间，不便学习到能运用到真是计算机视觉任务的方法中。随着时代的进步，人们对于穿着打扮的需求也在增加，服装的种类也在不断地因为创新而增加，比如 19 世纪 90 年代，随着黄金热的出现，牛仔服也被创造了出来。一家的德国的时尚科技公司旗下的研究部门提供了涵盖来自 10 中类别的共 7 万个不同商品的正面图片，用来代替 MNIST 手写数字集的图像数据集，命名为 FashionMNIST 数据集。数据集的大致情况如下：

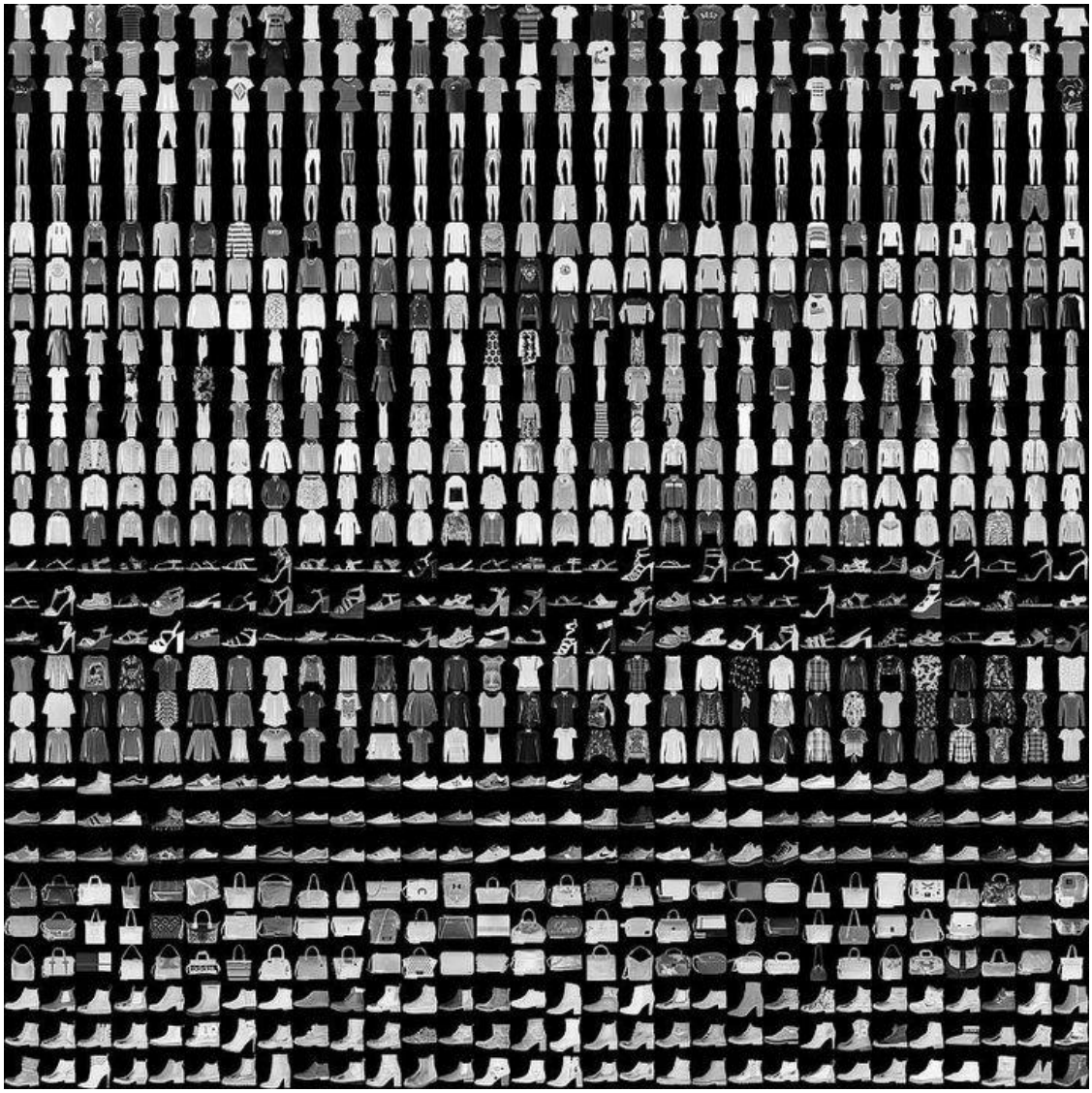


图 1 Fashion-Mnist 数据集

服装按照穿着组合大致可以分为十类，分别是：t-shirt（T 恤），trouser

（牛仔裤），pullover（套衫），dress（裙子），coat（外套），sandal（凉鞋），shirt（衬衫），sneaker（运动鞋），bag（包），ankle boot（短靴）。

FashionMNIST 数据集的大小、格式和训练集、测试集划分与原始的 MNIST 完全一致，采用了 28*28 的灰度图片，每个像素的值为 0~255 之间，用 784 个数据作为评判标准，因此我们需要通过研究该数据集中各个维度与类别的关系，构建一种机器学习算法模型，建立分类和图片像素状态之间的关系，通过一系列手段，使得模型具有更高的准确率、更好的鲁棒性和泛化性。

原始数据情况：服装分类训练集包含约 6 万条数据、服装分类测试集包含约 1 万条数据。

二、分析方法与过程

总体流程：

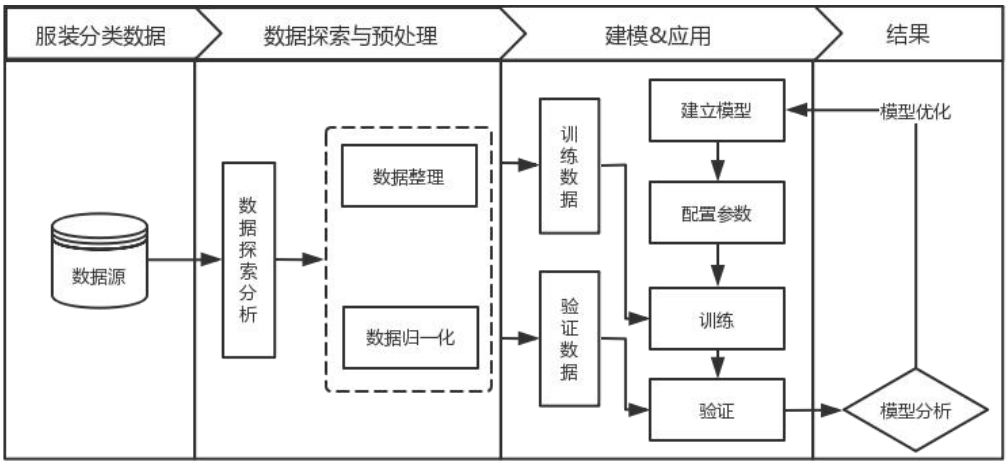


图 2 总体流程图

初步分析：

Fashion-MNIST 由 60000 张训练集图像、10000 张测试集图像及对应的标签构成，每张图像是分辨率为 28x28 的灰度图像，包含 10 种分类：T 恤、裤子、套头衫、连衣裙、大衣、凉鞋、衬衫、运动鞋、包、短靴。但数据中图像的格式并不是传统意义上的 28x28 的像素组，而是已经展开了的 748 个像素集合。

该问题因此我们组的任务是利用训练集来构建一种机器学习算法模型，而在分类任务中，正确率是更为直观的一种衡量方法，即统计样本预测值与实际值一致的情况占整个样本的比例（衡量样本被正确标注的数量），即 score = 正确

数/总数。因此我们需要使用测试集中 10,000 个图像来评估网络学习对图像分类的准确率，准确率越高，说明该模型分类效果越好。

考虑到在计算机视觉领域中，神经网络的使用较为广泛，且效果较好，因此我们首先决定建立一个多层全连接神经网络，之后在验证集的基础上考虑是否需要进行正则化。

第 1 步：数据探索分析

训练集中有 60,000 个图像，每个图像由 28 x 28 的像素表示，同样，训练集中有 60,000 个标签，每个标签都是一个 0 到 9 之间的整数，代表类别如下表 1；测试集中有 10,000 个图像。同样，每个图像都由 28x28 个像素表示，然而没有图像标签，只有 ID。

标签	0	1	2	3	4	5	6	7	8	9
类	T 恤/上衣	裤子	套头衫	连衣裙	外套	凉鞋	衬衫	运动鞋	包	短靴

表 1 类别与标签对应表

首先需要将原始的二维数据形式转换为三维数据形式，以便以图像形式展示。检查训练集中的第一个图像，发现像素值处于 0 到 255 之间。

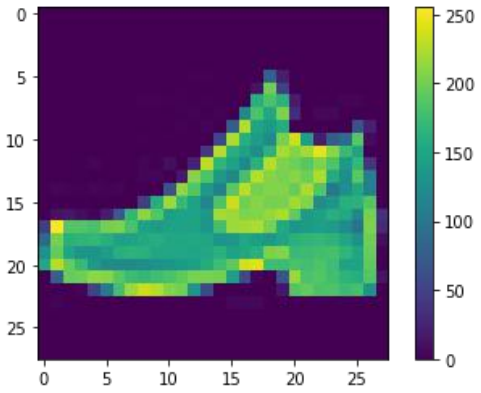


图 3 训练集第一个图像

第 2 步：数据预处理

因为使用神经网络模型需要进行多次迭代，为了提升模型的收敛速度和精度，我们首先对数据进行归一化处理，将这些值缩小至 0 到 1 之间：

train2 = train1 / 255.0

test2 = test1 / 255.0

为了验证数据的格式是否正确，显示训练集中的前 25 个图像，并在每个图像下方显示类名称。

图 4 图像及其类名

由于在此次比赛数据中，测试集没有所对应的类标签，我们并不能进行模型的评估，因此我们从训练集中分出 25%的数据作为验证集。

第 3 步：构建模型

1. 构建全连接神经网络模型

（1）设置层

第一层：该层为输入层，原始数据的图像格式已经是展开后的一维数组形式（28 x 28 = 784 像素）。将该层视为图像中未堆叠的像素行并将其排列起来。该层没有要学习的参数，它只会重新格式化数据。
第二层：该层为隐藏层，有 128 个节点（或神经元）。为了降低计算量以及避免出现梯度消失的情况，采用“Relu”作为激活函数。
第三层：该层为输出层，也是全连接神经层。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。

（2）编译模型：

选取为“Adam”为梯度下降优化器，指定学习率为 0.001；由于该问题是图像分类，损失函数采用交叉熵损失；监控训练和测试步骤的指标采用准确率，即被正确分类的图像的比率。

2. 训练全连接模型

（1）将训练数据馈送给模型。训练数据位于 train3 和 train_label 数组中。

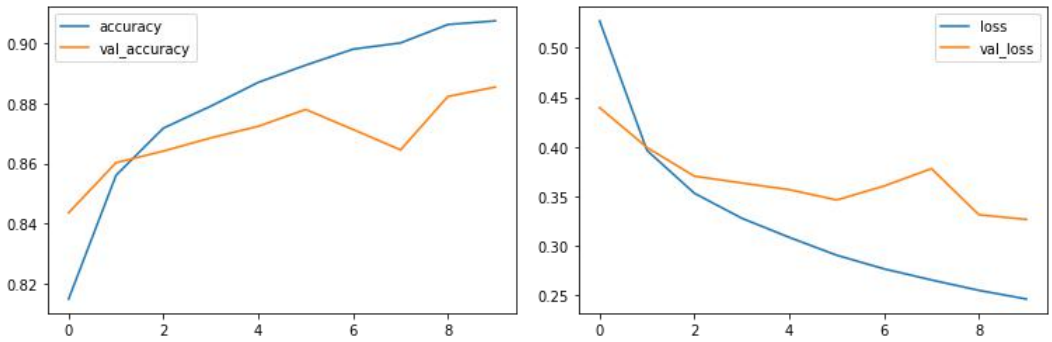
（2）模型学习将图像和标签关联起来。设定迭代次数为 10，batch_size 为 32。即每迭代一次训练时，从样本中选择 32 个样本进行训练。

（3）同时要求模型对验证集（vali 数组）进行预测

（4）验证预测是否与 vali_label 数组中的标签相匹配。

3. 模型分析（1）

模型 1 训练及验证结果：模型训练准确率：0.9075，模型验证准确率：0.8855。



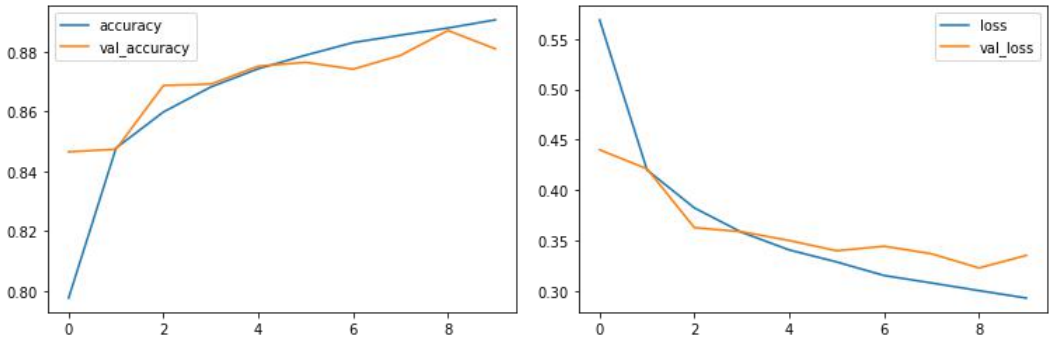
上述结果表明，模型在验证数据集上的准确率略低于训练数据集。表明该模型过拟合，过拟合是指机器学习模型在新的、以前未曾见过的输入上的表现不如在训练数据上的表现。因此下面我们需要对模型进行优化避免过拟合。

4. 模型优化——加入 Dropout 正则化

为了避免模型过拟合，我们对模型进行正则化操作，这里我们采用 Dropout 正则化，随机的对神经网络每一层进行去弃部分神经元操作。设置丢弃率为 0.2，即每一层有 20%的神经元会被丢弃，再次进行模型编译与训练。

5. 模型分析（2）

模型 2 训练及验证结果：模型训练准确率：0.8907，模型验证准确率：0.8810。



由上面结果，我们可以看到虽然模型训练准确率较第一次有所下降，但是模型的验证准确率也有所提高，说明加了 Dropout 的效果要比加了 Dropout 的效果要好。但是总体而言，这个模型比较简单，为了得到更精准的模型，那么下面我们又构建一个卷积神经网络进行图像分类。

6. 数据预处理

由于卷积神经网络的输入需要是一个四维数组，因此首先需要将原始二维数据通过循环赋值转换为三维数组(60000 * 28 * 28 像素)后,再重塑为四维数组（60000 * 28 * 28 * 1 像素）。

7. 构建卷积神经网络模型

（1）设置层

第一层：该层为卷积层 1，有 32 个卷积核，卷积核的大小为 3*3，步长为 1，在图片像素的最外层加上 1 层 0 值，激活函数依旧是“Relu”。
第二层：该层为池化层 1，最大池化，池化窗口为 2*2，步长为 2，目的是减少图片的特征数量，避免全连接层参数过多，同时提高 Feature Map 的鲁棒性，防止过拟合。
第三层：该层为卷积层 2，有 64 个卷积核，卷积核的大小为 3*3，步长为 1，在图片像素的最外层加上 1 层 0 值，激活函数：“Relu”。
第四层：该层为池化层 2，最大池化，池化窗口为 2*2，步长为 2。
第五层：该层为卷积层 2，有 128 个卷积核，卷积核的大小为 3*3，步长为 1，在图片像素的最外层加上 1 层 0 值，激活函数：“Relu”。
第六层：该层为池化层 2，最大池化，池化窗口为 2*2，步长为 2。
第七层：该层重新格式化数据，将图像格式从三维数组（3 * 3 * 128 像素）转换成一维数组（3 * 3 * 128 = 1,152 像素）。

第八层：该层为全连接层 1，有 128 个神经元。激活函数：“Relu”，采用了 Dropout 正则化，丢弃概率为 0.2。
第九层：该层为全连接层 2，有 64 个神经元。激活函数：“Relu”，采用了 Dropout 正则化，丢弃概率为 0.2。
第十层：该层为全连接层 3，有 32 个神经元。激活函数：“Relu”，采用了 Dropout 正则化，丢弃概率为 0.2。
第十一层：该层为输出层，也是全连接神经层。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。

（2）编译模型：

同样，梯度下降优化器：“Adam”，学习率为 0.001；损失函数：交叉熵损失；指标：准确率。

8. 训练卷积神经网络模型

（1）将训练数据馈送给模型。训练数据位于 train_4D 和 train_label 数组中。

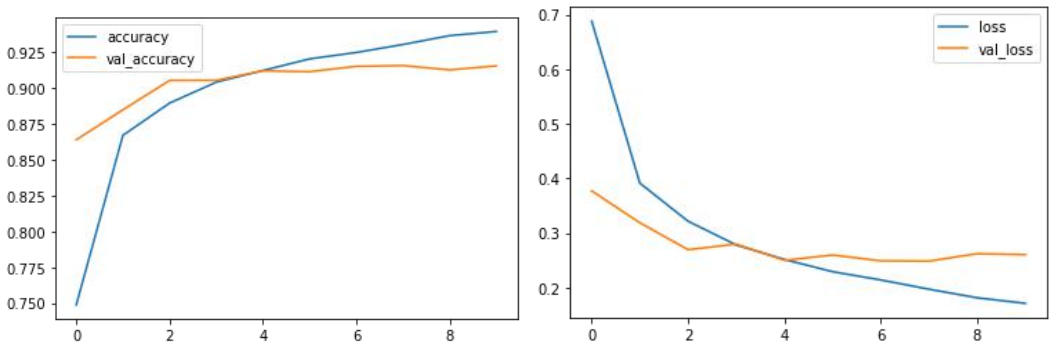
（2）模型学习将图像和标签关联起来。设定迭代次数为 10，batch_size 为 32。即每迭代一次训练时，从样本中选择 32 个样本进行训练。

（3）同时要求模型对验证集（vali 数组）进行预测。

（4）验证预测是否与 vali_label 数组中的标签相匹配。

9. 模型分析

模型 3（卷积神经网络模型）训练及验证结果：模型训练准确率：0.9396，模型验证准确率：0.9156。



由上面结果，无论是准确率还是验证准确率都达到了 90%以上，还是比较理想的。

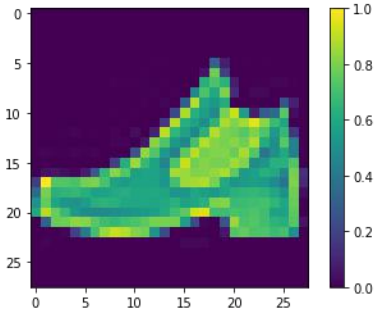
10. 进行预测

在每个模型的最后加一个全连接层，激活函数为“Softmax”，返回该图片属于各个类别的概率。利用最终得到的卷积神经网络模型对测试集进行预测，进行整理后输出提交到比赛平台。

三、挖掘算法实现

```
import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

train = pd.read_csv('fashion-mnist_train.csv') #  读取训练集
test = pd.read_csv('fashion-mnist_test_data.csv') #  读取测试集
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                        'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot'] #类别名

##数据预处理
#整理数据集
train_labels=train['label'].copy()
del train['label']
del test['iD']
# dataframe -> ndarray
train1 = train.values
train_labels = train_labels.values
test1 = test.values

#数据归一化（做特征值的归一化，所有像素值的范围为 0~225）
train2 = train1 / 255.0
test2 = test1 / 255.0

#从训练集中分出验证集
train3 = train2[0:45000].copy() #训练集
vali = train2[45000:60000].copy() #验证集
train_label = train_labels[0:45000].copy()#训练集
vali_label = train_labels[45000:60000].copy()#验证集

#将数据转换为三维数组形式
#训练集
train_3D = np.zeros([45000,28,28])
for i in range(0,45000):
```

```
        j=0
        k=0
        while(k<28 and j<28):
            train_3D[i,j,k]=train3[i,j*28+k]
            k=k+1
            if k==28:
                j=j+1
                k=0
#验证集
vali_3D = np.zeros([15000,28,28])
for i in range(0,15000):
        j=0
        k=0
        while(k<28 and j<28):
            vali_3D[i,j,k]=vali[i,j*28+k]
            k=k+1
            if k==28:
                j=j+1
                k=0
#测试集
test_3D = np.zeros([10000,28,28])
for i in range(0,10000):
        j=0
        k=0
        while(k<28 and j<28):
            test_3D[i,j,k]=test2[i,j*28+k]
            k=k+1
            if k==28:
                j=j+1
                k=0

#将三维数组转换为四维数组形式
vali_4D = vali_3D.reshape((15000,28,28,1))
train_4D = train_3D.reshape((45000,28,28,1))
test_4D =test_3D.reshape((10000,28,28,1))

##画图
plt.figure()
plt.imshow(train_3D[1])
plt.colorbar()
plt.grid(False)
plt.show()
```



```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_3D[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_label[i]])
plt.show()
```


```
#全连接神经网络模型 1
model1 = keras.Sequential([
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),    #定义隐藏层，128 个神经元的网络层
    keras.layers.Dense(10) #10 个类别，则全连接层输出神经元个数必须跟总类别数相同
])

#配置训练相关参数
model1.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
#选择优化器，指定学习率 0.001
                loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
#计算损失的损失函数
                metrics=['accuracy'])    #评估：准确率

#模型训练拟合并验证
fit1 = model1.fit( train3, train_label , batch_size=32, epochs=10, validation_data = ( vali ,
```

vali_label))

#训练样本的特征值和目标值：batch_size 每次迭代样本数,epochs 迭代次数

#在每迭代一次训练时，6w 个数据中，每次选择 batch_size=32 个样本进行训练，在下一次迭代时，6w 个数据中心，每次选择 batch_size=32 个样本进行训练。

#训练中 loss 一般为 0.几，如果超过个位数，表明损失特别大。

```
#模型训练拟合并验证
fit1 = model1.fit(train3, train_label , batch_size=32, epochs=10, validation_data=(vali,vali_label))
```

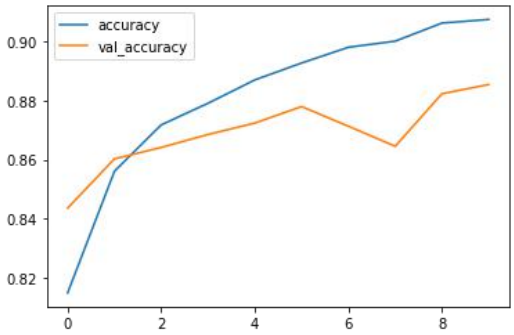
```
Epoch 1/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.5272 - accuracy: 0.8149 - val_loss: 0.4395 - val_accuracy: 0.8436
Epoch 2/10
1407/1407 [==============================] - 1s 898us/step - loss: 0.3962 - accuracy: 0.8561 - val_loss: 0.3993 - val_accuracy: 0.8603
Epoch 3/10
1407/1407 [==============================] - 1s 848us/step - loss: 0.3530 - accuracy: 0.8718 - val_loss: 0.3703 - val_accuracy: 0.8641
Epoch 4/10
1407/1407 [==============================] - 1s 851us/step - loss: 0.3278 - accuracy: 0.8791 - val_loss: 0.3634 - val_accuracy: 0.8685
Epoch 5/10
1407/1407 [==============================] - 1s 961us/step - loss: 0.3085 - accuracy: 0.8870 - val_loss: 0.3567 - val_accuracy: 0.8724
Epoch 6/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.2904 - accuracy: 0.8928 - val_loss: 0.3462 - val_accuracy: 0.8780
Epoch 7/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.2767 - accuracy: 0.8981 - val_loss: 0.3604 - val_accuracy: 0.8713
Epoch 8/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.2654 - accuracy: 0.9002 - val_loss: 0.3779 - val_accuracy: 0.8645
Epoch 9/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.2549 - accuracy: 0.9063 - val_loss: 0.3313 - val_accuracy: 0.8823
Epoch 10/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.2462 - accuracy: 0.9075 - val_loss: 0.3266 - val_accuracy: 0.8855
```

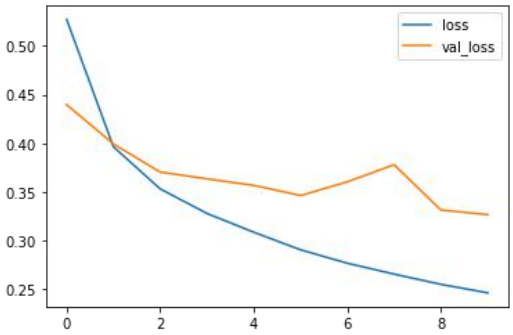
#画出准确率图

```
plt.figure()
plt.plot(fit1.epoch,fit1.history.get('accuracy'),label="accuracy")
plt.plot(fit1.epoch,fit1.history.get('val_accuracy'),label="val_accuracy")
plt.legend()
```

#画出损失率图

```
plt.figure()
plt.plot(fit1.epoch,fit1.history.get('loss'),label="loss")
plt.plot(fit1.epoch,fit1.history.get('val_loss'),label="val_loss")
plt.legend()
```





#全连接神经网络模型 2

```
model2 = keras.Sequential([
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.2), #Dropout 正则化，丢失率为 0.2
    keras.layers.Dense(10)
])
```

#配置训练相关参数

```
model2.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
               loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
               metrics=['accuracy'])
```

#模型训练拟合并验证

fit2 = model2.fit(train3, train_label , batch_size=32, epochs=10, validation_data = (vali , vali_label))

```
#模型训练拟合并验证
fit2 = model2.fit(train3, train_label , batch_size=32, epochs=10, validation_data=(vali,vali_label))
```

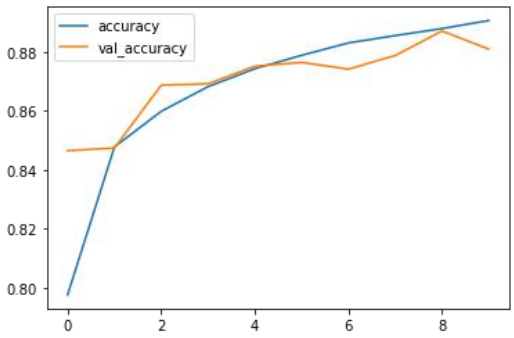
```
Epoch 1/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.5687 - accuracy: 0.7976 - val_loss: 0.4400 - val_accuracy: 0.8465
Epoch 2/10
1407/1407 [==============================] - 1s 1ms/step - loss: 0.4200 - accuracy: 0.8478 - val_loss: 0.4211 - val_accuracy: 0.8474
Epoch 3/10
1407/1407 [==============================] - 1s 1ms/step - loss: 0.3824 - accuracy: 0.8599 - val_loss: 0.3628 - val_accuracy: 0.8687
Epoch 4/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.3582 - accuracy: 0.8682 - val_loss: 0.3588 - val_accuracy: 0.8692
Epoch 5/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.3408 - accuracy: 0.8743 - val_loss: 0.3501 - val_accuracy: 0.8752
Epoch 6/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.3288 - accuracy: 0.8789 - val_loss: 0.3399 - val_accuracy: 0.8765
Epoch 7/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.3154 - accuracy: 0.8831 - val_loss: 0.3444 - val_accuracy: 0.8742
Epoch 8/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.3081 - accuracy: 0.8856 - val_loss: 0.3370 - val_accuracy: 0.8788
Epoch 9/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.3004 - accuracy: 0.8880 - val_loss: 0.3229 - val_accuracy: 0.8871
Epoch 10/10
1407/1407 [==============================] - 2s 1ms/step - loss: 0.2931 - accuracy: 0.8907 - val_loss: 0.3354 - val_accuracy: 0.8810
```

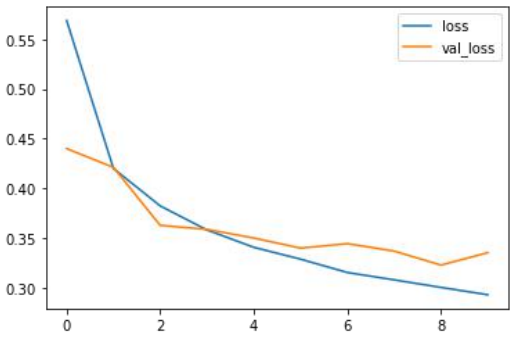
#画出准确率图

```
plt.figure()
plt.plot(fit2.epoch,fit2.history.get('accuracy'),label="accuracy")
plt.plot(fit2.epoch,fit2.history.get('val_accuracy'),label="val_accuracy")
plt.legend()
```

#画出损失率图

```
plt.figure()
plt.plot(fit2.epoch,fit2.history.get('loss'),label="loss")
plt.plot(fit2.epoch,fit2.history.get('val_loss'),label="val_loss")
plt.legend()
```



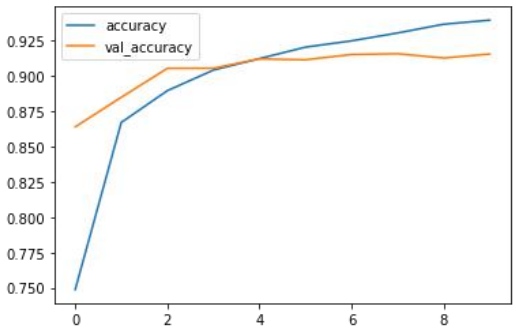


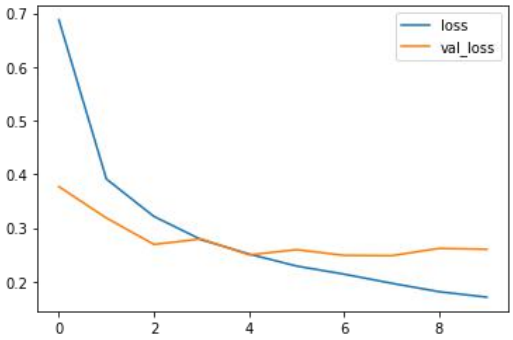
```
    #优化遇到的问题：梯度消失、局部最优
    #卷积神经网络的组成：由一个或多个卷积层（conbolutions）、池化层（subsampling）和全连接层（full connection）、激活函数等组成。需要考量的参数更小
    #卷积层：目的是提取输入的不同特征；参数：size，卷积核、过滤器大小；padding：零填充 valid 与 same， stride，步长，通常默认为 1

    #卷积神经网络模型
    model_CNN = keras.Sequential([      #读取数据集， 编写两层+两层全连接层网络模型
        keras.layers.Conv2D(32,kernel_size=3,strides=1,padding="same", data_format="channels_last", activation=tf.nn.relu),
    #卷积层 1：32 个，3*3 ，步长为 1 ，channel 在最后一个维度
        keras.layers.MaxPool2D(pool_size=2,strides=2,padding = 'same'),
    #池化：2*2 窗口，步长为 2
        keras.layers.Conv2D(64,kernel_size=3,strides=1,padding="same", data_format="channels_last", activation=tf.nn.relu),
    #卷积层 2：64 个，3*3 ，步长为 1 ，channel 在最后一个维度
        keras.layers.MaxPool2D(pool_size=2,strides=2,padding = 'same'),
    #池化：2*2 窗口，步长为 2
        keras.layers.Conv2D(128,kernel_size=3,strides=1,padding="same", data_format="channels_last", activation=tf.nn.relu),
    #卷积层 3：128 个，3*3 ，步长为 1 ，channel 在最后一个维度
        keras.layers.MaxPool2D(pool_size=2,strides=2,padding = 'same'),
    #池化：2*2 窗口，步长为 2
        keras.layers.Flatten(),      #全连接层神经网络
        keras.layers.Dense(128, activation='relu'),
        #relu 函数，修正线性单元，128 个神经元网络层
        keras.layers.Dropout(0.2),
        keras.layers.Dense(64, activation='relu'),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(32, activation='relu'),
        keras.layers.Dropout(0.2),
        keras.layers.Dense(10)     #10 个神经元的神经网络
    ])

    #配置训练相关参数
    model_CNN.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                      loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])

    #模型训练拟合并验证
    fit_CNN = model_CNN.fit( train_4D, train_label, batch_size = 32 , epochs = 10 , validation_data = ( vali_4D , vali_label ))
```

```
fit_CNN = model_CNN.fit(train_4D, train_label, batch_size=32 ,epochs=10,validation_data=(vali_4D,vali_label))
```

```
Epoch 1/10
1407/1407 [==============================] - 32s 23ms/step - loss: 0.6879 - accuracy: 0.7487 - val_loss: 0.3768 - val_accuracy: 0.8640
Epoch 2/10
1407/1407 [==============================] - 35s 25ms/step - loss: 0.3910 - accuracy: 0.8671 - val_loss: 0.3184 - val_accuracy: 0.8849
Epoch 3/10
1407/1407 [==============================] - 37s 26ms/step - loss: 0.3212 - accuracy: 0.8896 - val_loss: 0.2691 - val_accuracy: 0.9055
Epoch 4/10
1407/1407 [==============================] - 39s 28ms/step - loss: 0.2777 - accuracy: 0.9042 - val_loss: 0.2793 - val_accuracy: 0.9055
Epoch 5/10
1407/1407 [==============================] - 39s 28ms/step - loss: 0.2511 - accuracy: 0.9124 - val_loss: 0.2496 - val_accuracy: 0.9121
Epoch 6/10
1407/1407 [==============================] - 38s 27ms/step - loss: 0.2287 - accuracy: 0.9205 - val_loss: 0.2593 - val_accuracy: 0.9115
Epoch 7/10
1407/1407 [==============================] - 38s 27ms/step - loss: 0.2136 - accuracy: 0.9249 - val_loss: 0.2487 - val_accuracy: 0.9153
Epoch 8/10
1407/1407 [==============================] - 39s 28ms/step - loss: 0.1966 - accuracy: 0.9305 - val_loss: 0.2481 - val_accuracy: 0.9158
Epoch 9/10
1407/1407 [==============================] - 40s 28ms/step - loss: 0.1809 - accuracy: 0.9367 - val_loss: 0.2617 - val_accuracy: 0.9128
Epoch 10/10
1407/1407 [==============================] - 43s 31ms/step - loss: 0.1707 - accuracy: 0.9396 - val_loss: 0.2600 - val_accuracy: 0.9156
```

```
#画出准确率图
plt.figure()
plt.plot(fit_CNN.epoch,fit_CNN.history.get('accuracy'),label="accuracy")
plt.plot(fit_CNN.epoch,fit_CNN.history.get('val_accuracy'),label="val_accuracy")
plt.legend()

#画出损失率图
plt.figure()
plt.plot(fit_CNN.epoch,fit_CNN.history.get('loss'),label="loss")
plt.plot(fit_CNN.epoch,fit_CNN.history.get('val_loss'),label="val_loss")
plt.legend()
```






```
#进行预测
probability_model2 = tf.keras.Sequential([model2,tf.keras.layers.Softmax()])
predictions2 = probability_model2.predict(vali)#用模型 2 预测
probability_model3 = tf.keras.Sequential([model_CNN,tf.keras.layers.Softmax()])
predictions3 = probability_model3.predict(vali_4D)#用模型 3 预测

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)

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')

num_rows = 5
num_cols = 3
num_images = num_rows*num_cols

#全连接神经网络预测结果
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, predictions2[i], vali_label, vali_3D)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions2[i], vali_label)
plt.tight_layout()
plt.show()

#卷积神经网络预测结果
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, predictions3[i], vali_label, vali_3D)
    plt.subplot(num_rows, 2*num_cols, 2*i+2)
    plot_value_array(i, predictions3[i], vali_label)
plt.tight_layout()
plt.show()
```

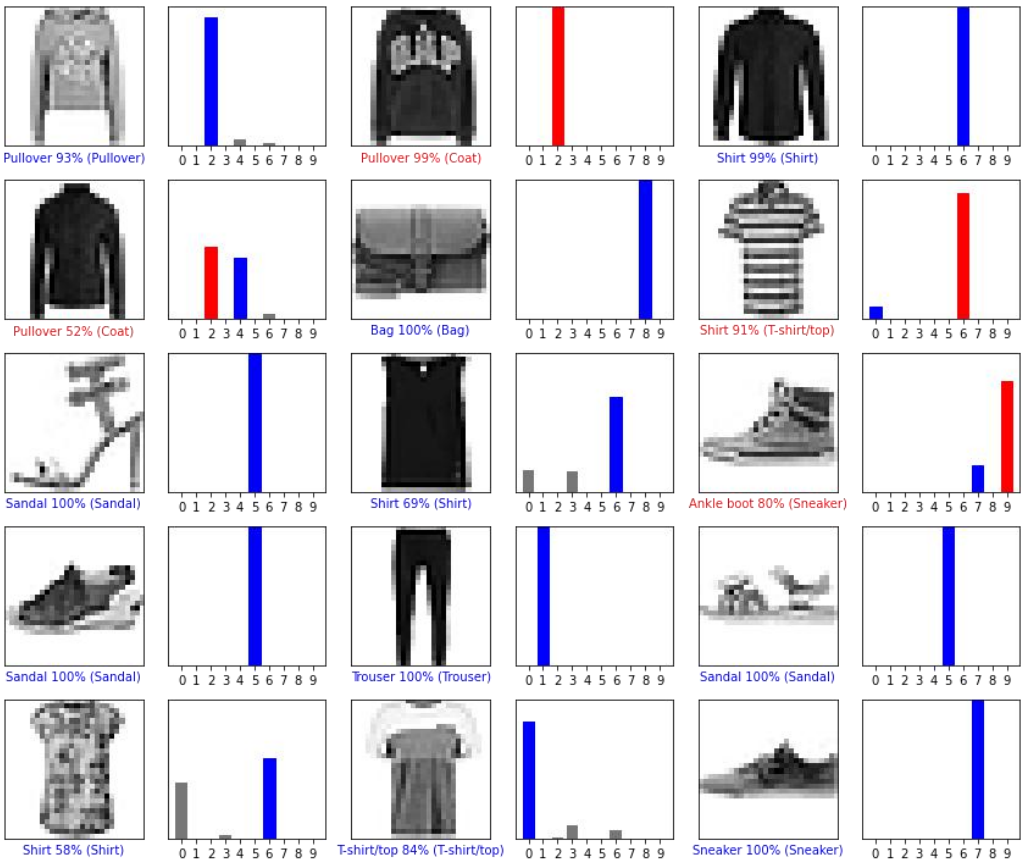


图 全连接神经网络预测结果

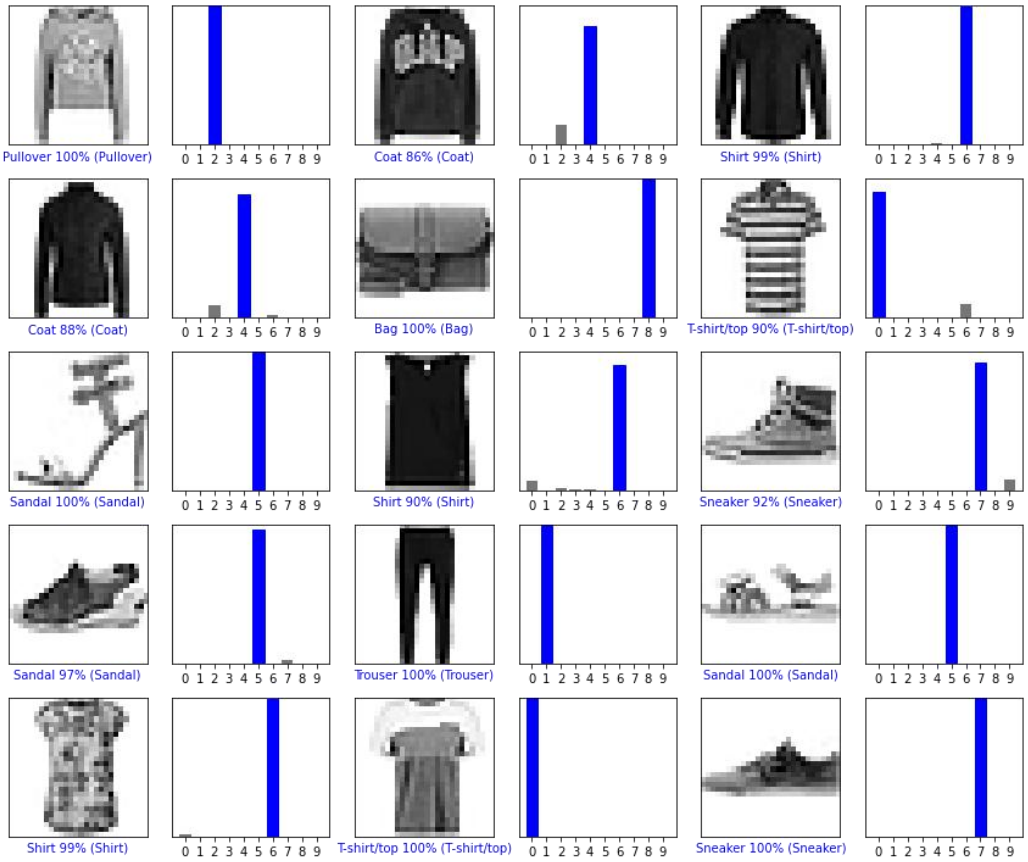


图 全连接神经网络预测结果

```
#用卷积神经网络模型预测测试集
probability_model_CNN = tf.keras.Sequential([model_CNN,tf.keras.layers.Softmax()])
predictions_CNN = probability_model_CNN.predict(test_4D)

#整理预测结果
result = pd.DataFrame(columns=['id','class'])
for i in range(10000):
    a = str(i)+'.jpg'
    b =   np.argmax(predictions_CNN[i])
    result.loc[i+1]={'id':a,'class':b}

#保存预测结果到 result.csv
result.to_csv('result.csv', sep=',', header=False, index=False)
```

1	0. jpg	0
2	1. jpg	1
3	2. jpg	2
4	3. jpg	2
5	4. jpg	3
6	5. jpg	6
7	6. jpg	8
8	7. jpg	6
9	8. jpg	5
10	9. jpg	0
11	10. jpg	3
12	11. jpg	2
13	12. jpg	4
14	13. jpg	6
15	14. jpg	8
16	15. jpg	5
17	16. jpg	4
18	17. jpg	3
19	18. jpg	6
20	19. jpg	4
21	20. jpg	4
22	21. jpg	4

比赛提交结果：

四、总结与推广

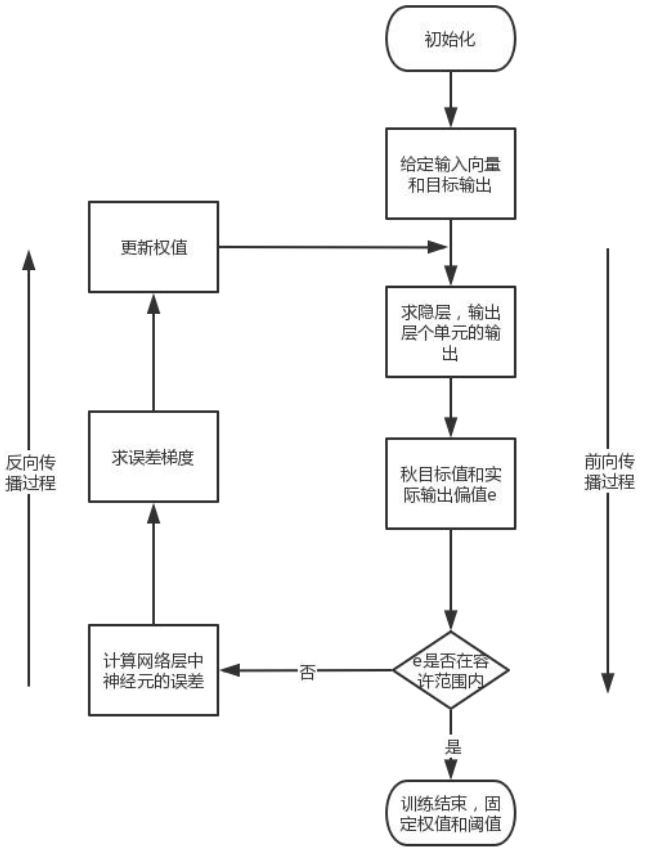
总结：

通过这个课程设计，我们发现优化改变模型可以训练得到一个好的神经网络，因此在模型不尽人意的时候，可以进行网络优化调整，同时对数据和约束的优化调整也能训练得到一个满意的神经网络模型。

神经网络中最常见的形式就是全连接层，深度学习神经网络中，一般都有多个隐藏层顺序排列组成完整的神经网络。但是全连接存在一些缺点，现实生活中的很多任务，比如目标检测，分割等不要求提取全图的特征，只需要提取能够覆盖目标物体大小的感受野内的特征，当目标物体特别小时，感受野很小，如果还采用全连接去提取全图的特征，那么我们的目标就会被淹没在和其他背景的平均特征之中变得不可识别。而卷积神经网络就可以很好地提取到目标的特征，哪怕目标特别小，同时卷积神经网络可以不要求输入尺寸为固定尺寸，只要求输入图片不太小到网络下采样到尺寸不够用就行，输入图像采样到 1*1 则和全连接一样的效果。全连接的计算形式不如卷积神经网络通用，所以可以被卷积形式替换，且卷积神经网络对比全连接神经网络更加的精确。

值得一提的是，开始的全连接神经网络过拟合了，通过查阅资料，我们了解到了 Dropout，在标准 Dropout 的每轮迭代中，网络中的每个神经元以 p 的概率被丢弃。Dropout 能够有效地改善归你和的情况，提升泛化能力。当然 Dropout 的实现有小要点的：1.一般是实施在分类器之前；2.Dropout 以概率 p 置零神经元，这种情况下，保留的神经元的输出要除以 1-p；3.通常 p 初始值为 0.2。

卷积神经网络（CNN）主要是用于图像识别领域，它指的是一类网络，而不是某一种，其包含很多不同种结构的网络。不同的网络结构通常表现会不一样。卷积神经网络的训练过程流程图：



不断循环这个过程，最后得到一个稳定的权值和阈值。

其中池化可以理解为定义一个选择框，将输入数据某个范围（矩阵）的所有数值进行相应计算，得到一个新的值，作为结果的一个像素点；池化也有步长和补齐的概念，但是很少使用，通常选择框以不重叠的方式，在 padding=0 的输入数据上滑动，生成一张新的特征图。

推广：

卷积神经网络模型不仅可以用于服装分类，还可以用于其他多维度数据的分类识别分类；也可以用于相似图搜索，比如人脸识别，对比两张照片是否是同一个人，当两张照片是同一个人是，他的欧氏距离会非常接近。

同时这次所运用的两个神经网络也可以进行推广，如果测试机中出现了未被分类的数据样本，我们可以通过孪生神经网络对于没有参与训练的类别进行识别，孪生神经网络的泛化能力对于复杂的现实有着广大的应用前景。