上海交通大學

机器学习
第二次大作业

论文题目：图像左右手兼伸出手指数识别
——基于卷积神经网络

指导老师：陈立　学院（系）：电子信息与电气工程学院

学生：冯绍庭　学院（系）：电子信息与电气工程学院

学生：水圣涛　学院（系）：电子信息与电气工程学院

图像左右手兼伸出手指数识别

——基于卷积神经网络

作者：电子信息与电气工程学院　　F2003402　　水圣涛
电子信息与电气工程学院　　F2003402　　冯绍庭

摘要

本文从解决图像左右手兼伸出手指数识别问题出发，以 Python 为工具，编写了以卷积神经网络为核心的算法，并成功建立了能有效解决此问题的模型。在得到有效模型后，本文从卷积神经网络参数出发，采用控制变量与交叉验证的方法对模型进行了优化。

关键词：卷积神经网络；控制变量；交叉验证

目录

1. 引言

1.1 神经网络简介

人工神经网络（artificial neural network，ANN），简称神经网络（neural network，NN），是一种模仿生物神经网络的结构和功能的数学模型或计算模型。神经网络由大量的人工神经元联结进行计算。大多数情况下人工神经网络能在外界信息的基础上改变内部结构，是一种自适应系统。现代神经网络是一种非线性统计性数据建模工具，常用来对输入和输出间复杂的关系进行建模，或用来探索数据的模式。

神经网络是一种运算模型，由大量的节点（或称“神经元”）和之间相互的联接构成。每个节点代表一种特定的输出函数，称为激励函数、激活函数（activation function）。每两个节点间的联接都代表一个对于通过该连接信号的加权值，称之为权重，这相当于人工神经网络的记忆。网络的输出则依网络的连接方式，权重值和激励函数的不同而不同。而网络自身通常都是对自然界某种算法或者函数的逼近，也可能是对一种逻辑策略的表达。

它的构筑理念是受到生物（人或其他动物）神经网络功能的运作启发而产生的。人工神经网络通常是通过一个基于数学统计学类型的学习方法得以优化，所以人工神经网络也是数学统计学方法的一种实际应用，通过统计学的标准数学方法我们能够得到大量的可以用函数来表达的局部结构空间，另一方面在人工智能学的人工感知领域，我们通过数学统计学的应用可以来做人工感知方面的决定问题(也就是说通过统计学的方法，人工神经网络能够类似人一样具有简单的决定能力和简单的判断能力)，这种方法比起正式的逻辑学推理演算更具有优势。

常见的神经网络模型（以 M-P 神经网络为例）和神经元模型如图 1.1，图 1.2 所示。

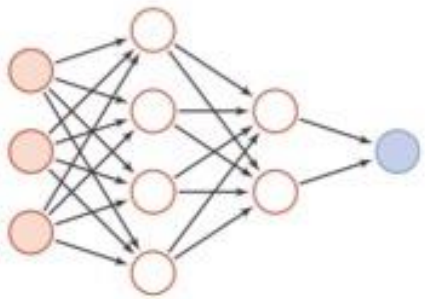

图 1.1 神经网络基本模型

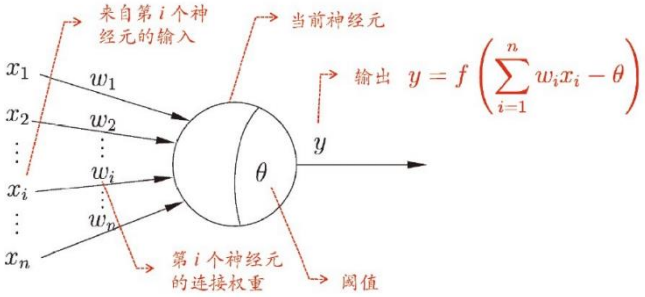

图 1.2 神经元模型

1.2 任务介绍

本团队的项目研究目标为建立一个能够计算手指数并区分左右手的模型（Task 5）。

研究的数据集包含 21600 张左手和右手手指的图像，其中训练集占 18000 张图像，测试集占 3600 张图像，图像大小均为 128*128 像素。所有的图像都以质心为中心，且背景中均有噪声。图 1.3 给出了 8 张数据集中的样例图片。

图片的名字的最后两位（eg.4L），L/R 表示左/右手，0、1、2、3、4、5 代表伸出的手指数。

需要注意的是，右手的图像是由左手的图像翻转得到的，通过这种方法，我们可以扩充数据集用来更充分地训练模型。

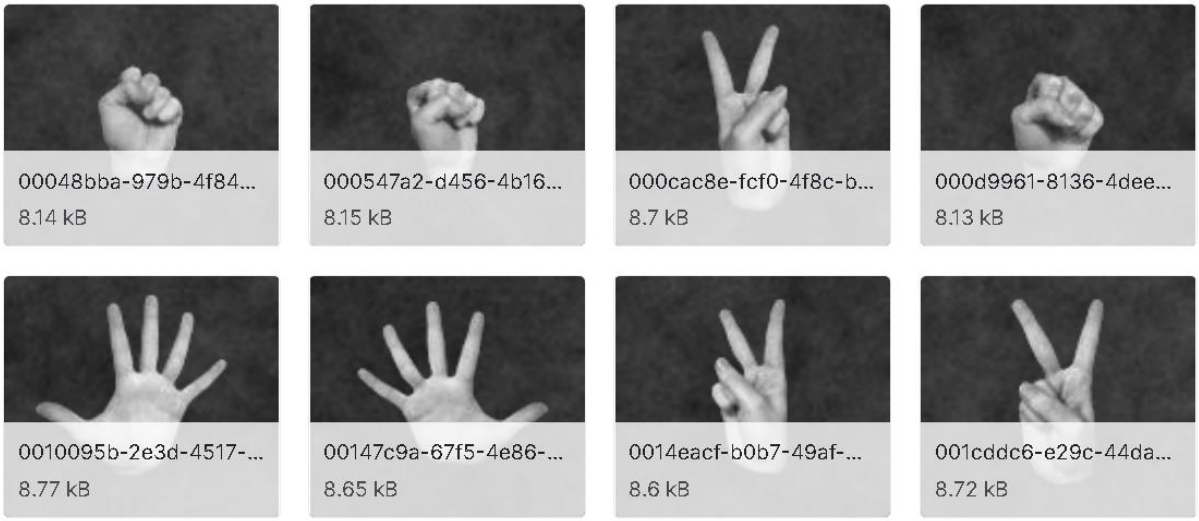

图 1.3 数据集样例

1.3 人员分工

本团队由水圣涛、冯绍庭两位成员组成。表 1.1 展示了团队的基本分工。

表 1.1 声呐系统评判指标

成员	主要任务
水圣涛	代码编写、结果分析、方案改进、报告撰写
冯绍庭	代码编写、代码调试、参数分析、模型优化

2. 解决方法

2.1 解决方法——卷积神经网络

卷积神经网络(Convolutional Neural Network, CNN)是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。对于图像识别问题，卷积神经网络也有着无与伦比的优势。

基础的 CNN 由 卷积层(convolution layer), 池化层(pooling layer)和全连接层(fully-connected layer)三层结构组成。CNN 输出的结果是每幅图像的特定特征空间。当处理图像分类任务时，我们会把 CNN 输出的特征空间作为全连接层或全连接神经网络(fully connected neural network, FCN)的输入，用全连接层来完成从输入图像到标签集的映射，即分类。当然，整个过程最重要的工作就是如何通过训练数据迭代调整网络权重，也就是后向传播算法（BP 算法）。

相比之下用全连接神经网络处理图像具有三个明显的缺点：

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。

而使用卷积神经网络可以很好地解决上面的三个问题。

与常规神经网络不同，卷积神经网络的各层中的神经元是 3 维排列的：宽度、高度和深度。其中的宽度和高度是很好理解的，因为本身卷积就是一个二维模板，但是在卷积神经网络中的深度指的是激活数据体的第三个维度，而不是整个网络的深度，整个网络的深度指的是网络的层数。图 2.1 展示了全连接神经网络与卷积神经网络的不同。

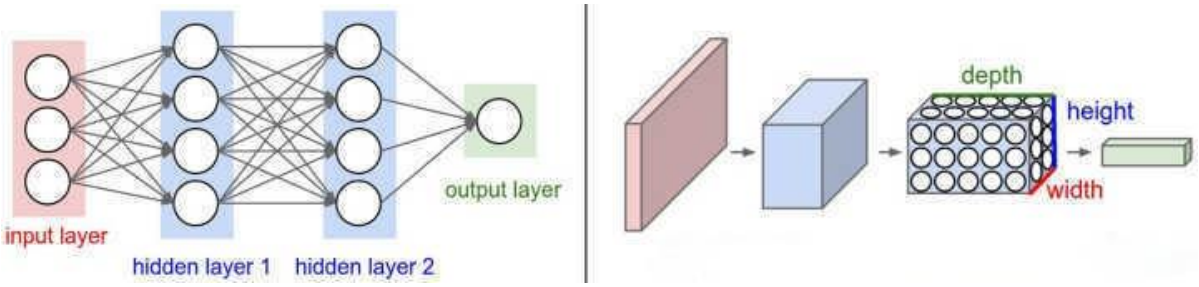

图 2.1 全连接神经网络（左）与卷积神经网络（右）

综上，本团队选择使用卷积神经网络作为模型来解决手势识别问题。

2.2 代码详解

2.2.1 下载与引用库

首先要确定并安装本模型需要使用的库。图2.2展示了下载效果。

```
!pip  install  Kaggle
```

```
Requirement already satisfied: Kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from Kaggle)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from Kaggl
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from Kaggl
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from Kag
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from Kagg
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (
```

图 2.2 库下载效果

接下来需要引用所有去要的库，具体代码如代码块 2.1 所示。

代码块 2.1 引用库

```
import tensorflow as tf
from zipfile import ZipFile
import os,glob
from skimage.io import imread
from skimage.transform import resize
import matplotlib.pyplot as plt
import random
import warnings
from scipy import ndarray
import skimage as sk
from skimage import transform
from skimage import util
from skimage import io
```

```
from sklearn import metrics
from tqdm._tqdm_notebook import tqdm_notebook as tqdm
import numpy as np
from keras.models import Sequential
from keras.layers import Convolution2D, Dropout, Dense
from keras.layers import BatchNormalization
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
from keras.layers import LeakyReLU
from numpy import asarray
from google.colab import files
from zipfile import ZipFile
import cv2
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_score
from numpy import *
```

2.2.2 下载数据

选择从 Kaggle 中下载数据，而不选择本地上传，因为数据量过于庞大，如果选择本地上传，会导致 Google 云端硬盘卡死。下载数据的代码见代码块 2.2，实现效果如图 2.3 所示。

代码块 2.2 下载数据

```
files.upload() # 上传了包含我的 kaggle 账户信息的 json 数据
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d koryakinp/fingers
```

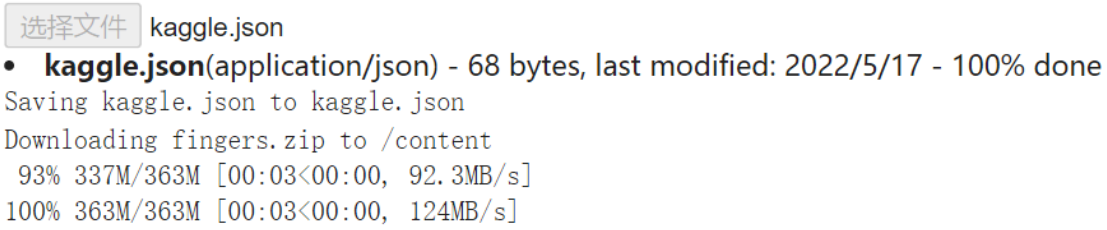

图 2.3 下载效果

2.2.3 解压文件

解压文件的代码比较简单，如代码块 2.3 所示。

代码块 2.3 解压文件

```
file_name = "fingers.zip"
with ZipFile(file_name,'r') as zip:
```

```
zip.extractall()
print('Done')
Done
```

2.2.4 图像处理

解使用 Opencv 进行图像处理，并放入相应的数据集。具体代码如代码块 2.4 所示。

代码块 2.4 解压文件

```
X_train = [] # 训练图像集
Y_train = [] # 训练标签集
os.chdir('/content/train')
print("Train:")
for i in tqdm(os.listdir()): # 一张一张图片处理，显示进度条
img = cv2.imread(i) # 读入图片
X_train.append(img) # 加入图像训练集
Y_train.append(i[-6:-4]) # 根据文件名提取标签加入标签训练集，
print("Shape of an image in X_train: ", X_train[0].shape) # 输入训练数据维度
print("Total categories: ", len(np.unique(Y_train))) # 训练集种类数
X_test = [] # 测试图像集
Y_test = [] # 测试标签集
os.chdir('/content/test')
print("Test:")
for i in tqdm(os.listdir()):
img = cv2.imread(i)
X_test.append(img)
Y_test.append(i[-6:-4])
print("Shape of an image in X_test: ", X_test[0].shape) # 输入测试数据维度
print("Total categories: ", len(np.unique(Y_test))) # 测试集种类数
```

2.2.5 数据集举例

本运行如下举例代码，得到如图 2.4 所示的效果。

代码块 2.5 数据集举例

```
%matplotlib inline
plt.figure(figsize=(10, 1))
for i in range(10):
plt.subplot(1, 10, i+1)
plt.imshow(X_test[i], cmap="gray")
plt.axis('off')
plt.show()
print('label for each of the above image: %s' % (Y_test[0:10]))
```

label for each of the above image: ['4L', '3L', '1L', '2R', '5R', '3R', '1R', '5L', '2L', '4

图 2.4 数据集展示

2.2.6 处理数据集

本部分要求对数据集进行标签序列化和标签集矩阵化，具体代码如代码块 2.6 所示。

代码块 2.6 数据集处理

```
# 标签序列化，把 12 个种类用 0-11 表示
le = preprocessing.LabelEncoder()
Y_train = le.fit_transform(Y_train)
Y_test = le.fit_transform(Y_test)
# 标签集矩阵化：每个标签对应一个 12 维向量，它是第 n 类，那么第 n 维为 1，其他维为 0
Y_train = tf.keras.utils.to_categorical(Y_train, num_classes=12)
Y_test = tf.keras.utils.to_categorical(Y_test, num_classes=12)
# list->array
Y_train = np.array(Y_train)
X_train = np.array(X_train)
Y_test = np.array(Y_test)
X_test = np.array(X_test)
```

2.2.7 搭建神经网络

本部分是所有代码中最重要的神经网络搭建部分，需要确定使用的卷积神经网络的层数与参数等，具体代码如代码块 2.7 所示。

代码块 2.7 搭建神经网络

```
myCNN = Sequential()
myCNN.add(BatchNormalization(input_shape=(128,128,3)))
myCNN.add(Convolution2D(32, (3,3), activation='relu', input_shape=(128, 128, 3)))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Convolution2D(filters=6, kernel_size=4, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Convolution2D(filters=128, kernel_size=3, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Convolution2D(filters=128, kernel_size=2, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Flatten())
  1 秒 完成时间：12:52
myCNN.add(Dense(units=128, activation='relu'))
myCNN.add(Dense(units=64, activation='relu'))
myCNN.add(Dense(units=32, activation='relu'))
myCNN.add(Dense(units=12, activation='softmax'))
myCNN.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

3. 结果展示

图 3.1 和图 3.2 给出了代码在训练集和测试集中的运行效果。

```
563/563 [==============================] - 11s 19ms/step - loss: 2.7667e-05 - accuracy: 1.00
```

图 3.1 训练集效果

```
113/113 [==============================] - 1s 8ms/step - loss: 6.1299e-05 - accuracy: 1.00
```

图 3.2 测试集效果

由上图可知，本模型的准确率相当之高，对于任何一张图片都能做到精准识别，但是因为参数过多等原因，代码运行速度较慢，仍可以进行优化。

为此，团队选择新建立一个较简单的网络结构，这将是第四部分（实验方案改进、模型优化）的主要内容。

4. 实验方案改进、模型优化

4.1 实验改进方案

团队决定从修改参数角度出发，优化实验方案，具体修改的参数为卷积层数、池化层数、全连接层数。

最终目的为在保证准确率的基础上，加快代码运行速度。

为了优化超参，我们采用控制变量+ cross-validation 的方法。控制变量意味着每次选一个超参改变，其他超参固定，固定此超参的最优取值。再调整下一个超参。

交叉验证是重复的使用数据，把训练集数据进行切分，组合为不同的子训练集和子测试集，用子训练集来训练模型，用子测试集来评估模型预测的好坏。在此基础上得到的多组不同的子训练集和子测试集，某次子训练集中的某样本在下次可能成为子测试集中的样本，即所谓“交叉”。比单次划分训练集和测试集的方法更稳定、全面，即泛化性更强。

原理图如图 4.1 所示，对应代码见代码块 4.1。

代码块 4.1 交叉验证

```
X_folds = np.array_split(X_train, 10)
Y_folds = np.array_split(Y_train, 10)
scores = list()
for k in range(10):
    # We use 'list' to copy, in order to 'pop' later on
    x_train = list(X_folds)
    x_test = x_train.pop(k)
    x_train = np.concatenate(x_train)
    y_train = list(Y_folds)
    y_test = y_train.pop(k)
    y_train = np.concatenate(y_train)
    myCNN = create_CNN()
    model = myCNN.fit(x_train, y_train, epochs=1, verbose=1, initial_epoch=0)
    loss_and_metrics = myCNN.evaluate(X_test, Y_test)
    scores.append(loss_and_metrics[0])
print("scores: ", scores)
```

```
17.  print("平均", mean(scores))
```

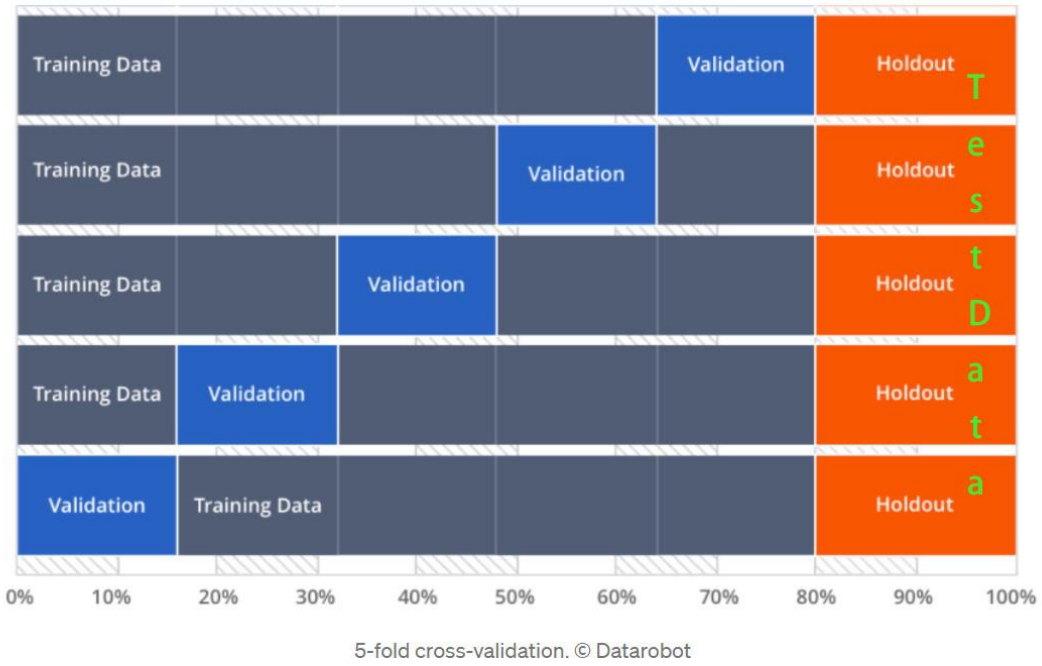

图 4.1 交叉验证原理图（实际操作中分了 10 组）

4.2 参数修改、优化

4.2.1 卷积层数

团队选择在一层或二层卷积层之间进行选择，以二层卷积层为例的模型示意图见图 4.2。

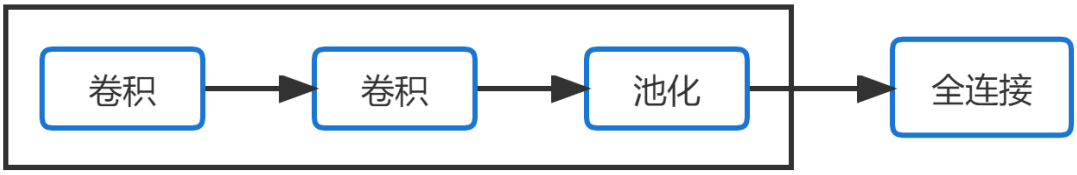

图 4.2 二层卷积层模型示意图

在代码中分别尝试一层卷积层与二层卷积层，实验结果如表 4.1 所示。

表 4.1 卷积层数运行效果对比

卷积层数	训练时间/s	平均错误率
1	8	2.623%
2	12	4.675%

由表 4.1 可知，相比二层卷积层，一层卷积层不但运行速度快，且平均错误率低。
综上，应当选择一层卷积层。

4.2.2 池化层数

在一层卷积层的基础上，团队选择在一层、二层或三层池化层之间进行选择，以二层池化层为例的模型示意图见图 4.3。

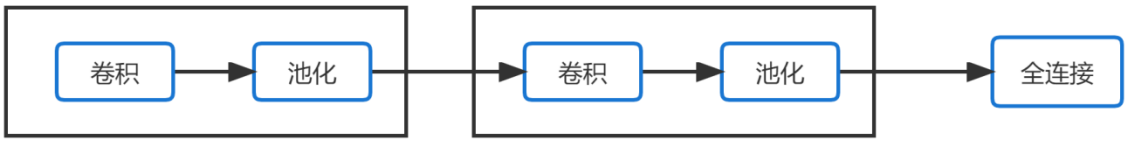

图 4.3 二层池化层模型示意图

在代码中分别尝试一层池化层、二层池化层与三层池化层，实验结果如表 4.2 所示。

表 4.2 池化层数运行效果对比

池化层数	训练时间/s	平均错误率
1	6	0.947%
2	8	0.262%
3	8	1.105%

由表 4.2 可知，一层池化层运行速度较快，但平均错误率略高。二层池化层运行速度为一层池化层的 1.33 倍，但是平均错误率仅为一层池化层的约 28%，相比之下更有优势。而三层池化层平均错误率回升，不予考虑。

综上，应当选择二层池化层。

4.2.3 全连接层数

在确定了一层卷积层和两层池化层的基础上，团队选择在一层或二层全连接层之间进行选择，以二层全连接层为例的模型示意图见图 4.4。

其中，全连接层使用的是 softmax 函数，两层全连接层也都使用了 softmax 函数。若决定采用两层全连接层时，前面一层使用 relu，后一层使用 softmax，应当可以获得更好的结果。

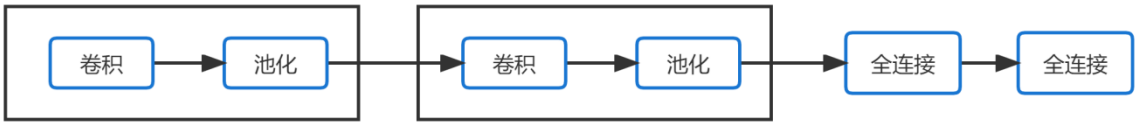

图 4.4 二层全连接层模型示意图

在代码中分别尝试一层全连接层与二层全连接层，实验结果如表 4.3 所示。

表 4.3 全连接层数运行效果对比

全连接层数	训练时间/s	平均错误率
1	8	0.262%
2	8	1.648%

由表 4.3 可知，相比二层全连接层，一层全连接层训练时间相近，但平均错误率低较多。

综上，应当选择一层全连接层。

4.3 最终结果

经综合以上分析，对模型的搭建神经网络部分进行修改优化。图 4.5 和图 4.6 给出了优化前后神经网络部分代码。

```
# The sequential API allows you to create models layer-by-layer for most problems.
# It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.
myCNN = Sequential()
myCNN.add(BatchNormalization(input_shape=(128,128,3))) # 数据预处理：标准化
myCNN.add(Convolution2D(filters=32, kernel_size=3, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Convolution2D(filters=6, kernel_size=4, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Convolution2D(filters=128, kernel_size=3, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Convolution2D(filters=128, kernel_size=2, padding='same', activation='relu'))
myCNN.add(MaxPooling2D(pool_size=2))
myCNN.add(Flatten())
myCNN.add(Dense(units=128, activation='relu'))
myCNN.add(Dense(units=64, activation='relu'))
myCNN.add(Dense(units=32, activation='relu'))
myCNN.add(Dense(units=12, activation='softmax'))
myCNN.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

图 4.5 优化前神经网络

```
def create_CNN():
    # The sequential API allows you to create models layer-by-layer for most problems.
    # It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.
    myCNN = Sequential()
    num_filter = 6
    myCNN.add(BatchNormalization(input_shape=(128,128,3))) # 数据预处理：标准化
    myCNN.add(Convolution2D(filters=num_filter, kernel_size=3, padding='same', activation='leaky_relu'))
    myCNN.add(MaxPooling2D(pool_size=2))
    myCNN.add(Convolution2D(filters=num_filter, kernel_size=3, padding='same', activation='leaky_relu'))
    myCNN.add(MaxPooling2D(pool_size=2))
    myCNN.add(Flatten())
    myCNN.add(Dense(units=12, activation='softmax'))
    myCNN.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return myCNN
```

图 4.6 优化后神经网络

表 4.4 给出了优化前后模型的结果对比。

表 4.3 全连接层数运行效果对比

模型	训练时间/s	准确率
优化前	23	100%
优化后	8	99.9%

由表 4.4 可知，优化后模型的训练时间大幅下降，下降为原来的约 34%，这对于大量数据的处理是十分关键的。至于准确率，优化后模型保持了原先的高准确率，几乎能够做到完美识别。

参考文献

[1] 《机器学习第二次大作业》
[2] CNN 卷积层、全连接层的参数量、计算量[OL].https://zhuanlan.zhihu.com/p/77471991.
[3] 详解卷积神经网络(CNN)[OL].https://blog.csdn.net/qq_25762497/article/details/51052861.
[4] 【机器学习】Cross-Validation（交叉验证）详解[OL].https://zhuanlan.zhihu.com/p/24825503.

附录

实验代码

```
"""Fingers.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1fp4cG4Hd-uoQqSxVbBF4QTwIyFwPhRHM
"""

!pip install Kaggle

import tensorflow as tf
from zipfile import ZipFile
import os,glob
from skimage.io import imread
from skimage.transform import resize
import matplotlib.pyplot as plt
import random
import warnings
from scipy import ndarray
import skimage as sk
from skimage import transform
from skimage import util
from skimage import io
from sklearn import metrics
from tqdm.notebook import tqdm_notebook as tqdm
import numpy as np
from keras.models import Sequential
from keras.layers import Convolution2D, Dropout, Dense
from keras.layers import BatchNormalization
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import SGD
from keras.layers import LeakyReLU
from numpy import asarray
from google.colab import files
from zipfile import ZipFile
import cv2
from sklearn import preprocessing
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold, cross_val_score
from numpy import *

"""从 Kaggle 里面下载数据。没有本地上传，因为太大了，Google 云端硬盘会直接卡死。"""

files.upload() # 上传了包含我的 kaggle 账户信息的 json 数据

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d koryakinp/fingers

"""解压文件"""

file_name = "fingers.zip"

with ZipFile(file_name,'r') as zip:
  zip.extractall()
  print('Done')

"""使用 Opencv 进行图像处理放入相应的数据集"""

X_train = [] # 训练图像集
Y_train = [] # 训练标签集
os.chdir('/content/train')
print("Train:")
for i in tqdm(os.listdir()): # 一张一张图片处理，显示进度条
  img = cv2.imread(i) # 读入图片
  X_train.append(img) # 加入图像训练集
  Y_train.append(i[-6:-4]) # 根据文件名提取标签加入标签训练集
print("Shape of an image in X_train: ", X_train[0].shape) # 输入训练数据维度
print("Total categories: ", len(np.unique(Y_train))) # 训练集种类数

X_test = [] # 测试图像集
Y_test = [] # 测试标签集
os.chdir('/content/test')
print("Test:")
for i in tqdm(os.listdir()):
  img = cv2.imread(i)
  X_test.append(img)
  Y_test.append(i[-6:-4])
```

```
print("Shape of an image in X_test: ", X_test[0].shape) # 输入测试数据维度
print("Total categories: ", len(np.unique(Y_test))) # 测试集种类数

"""数据集举例"""

# Commented out IPython magic to ensure Python compatibility.
# %matplotlib inline
plt.figure(figsize=(10, 1))
for i in range(10):
  plt.subplot(1, 10, i+1)
  plt.imshow(X_test[i], cmap="gray")
  plt.axis('off')
plt.show()
print('label for each of the above image: %s' % (Y_test[0:10]))

"""处理数据集"""

# 标签序列化，把 12 个种类用 0-11 表示
le = preprocessing.LabelEncoder()
Y_train = le.fit_transform(Y_train)
Y_test = le.fit_transform(Y_test)

# 标签集矩阵化：每个标签对应一个 12 维向量，它是第 n 类，那么第 n 维为 1，其他维为 0
Y_train = tf.keras.utils.to_categorical(Y_train, num_classes=12)
Y_test = tf.keras.utils.to_categorical(Y_test, num_classes=12)

# list->array
Y_train = np.array(Y_train)
X_train = np.array(X_train)
Y_test = np.array(Y_test)
X_test = np.array(X_test)

"""优化前的神经网络"""

def create_original_CNN():
  myCNN = Sequential()
  myCNN.add(BatchNormalization(input_shape=(128,128,3))) # 数据预处理：标准化
  myCNN.add(Convolution2D(filters=32, kernel_size=3, padding='same', activation='relu'))
  myCNN.add(MaxPooling2D(pool_size=2))
  myCNN.add(Convolution2D(filters=6, kernel_size=4, padding='same', activation='relu'))
  myCNN.add(MaxPooling2D(pool_size=2))
  myCNN.add(Convolution2D(filters=128, kernel_size=3, padding='same', activation='relu'))
  myCNN.add(MaxPooling2D(pool_size=2))
```

```
    myCNN.add(Convolution2D(filters=128, kernel_size=2, padding='same', activation='relu'))
    myCNN.add(MaxPooling2D(pool_size=2))
    myCNN.add(Flatten())
    myCNN.add(Dense(units=128, activation='relu'))
    myCNN.add(Dense(units=64, activation='relu'))
    myCNN.add(Dense(units=32, activation='relu'))
    myCNN.add(Dense(units=12, activation='softmax'))
    myCNN.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

"""优化后的神经网络"""

def create_CNN():
    # The sequential API allows you to create models layer-by-layer for most problems.
    # It is limited in that it does not allow you to create models that share layers or have multiple inputs o
r outputs.
    myCNN = Sequential()
    num_filter = 6
    myCNN.add(BatchNormalization(input_shape=(128,128,3))) # 数据预处理：标准化
    myCNN.add(Convolution2D(filters=num_filter, kernel_size=3, padding='same', activation='leaky_relu'))
    myCNN.add(MaxPooling2D(pool_size=2))
    myCNN.add(Convolution2D(filters=num_filter, kernel_size=3, padding='same', activation='leaky_relu'))
    myCNN.add(MaxPooling2D(pool_size=2))
    myCNN.add(Flatten())
    myCNN.add(Dense(units=12, activation='softmax'))
    myCNN.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return myCNN

"""参数优化实验代码：cross-validation"""

X_folds = np.array_split(X_train, 10)
Y_folds = np.array_split(Y_train, 10)
scores = list()
for k in range(10):
    # We use 'list' to copy, in order to 'pop' later on
    x_train = list(X_folds)
    x_test = x_train.pop(k)
    x_train = np.concatenate(x_train)
    y_train = list(Y_folds)
    y_test = y_train.pop(k)
    y_train = np.concatenate(y_train)
    myCNN = create_CNN()
    model = myCNN.fit(x_train, y_train, epochs=1, verbose=1, initial_epoch=0)
    loss_and_metrics = myCNN.evaluate(X_test, Y_test)
```

```
    scores.append(loss_and_metrics[0])
print("scores: ", scores)
print("平均", mean(scores))

"""训练"""

myCNN = create_CNN()
model = myCNN.fit(X_train, Y_train, epochs=1, verbose=1, initial_epoch=0)

"""测试"""

loss_and_metrics = myCNN.evaluate(X_test, Y_test, batch_size=1)
print(loss_and_metrics)

"""从测试集中随机输入图像到模型中，看看输出的分类结果是否真的有这么高"""

predicted_classes = myCNN.predict(X_test[:,:,:,:])
predicted_classes = np.argmax(np.round(predicted_classes),axis=1)
predicted_classes[0]
k=X_test.shape[0]
r=np.random.randint(k)
print(r)
print("Prediction:",predicted_classes[r])
print("\nActuals:   ",Y_test[r])
```