学 号：**201614590130**

华北理工大学
NORTH CHINA UNIVERSITY OF SCIENCE AND TECHNOLOGY

毕 业 论 文

GRADUATE THESIS

论文题目：基于卷积神经网络识别手写数字

学生姓名：魏兴源

专业班级：海洋技术 1 班

学　　院：人工智能学院

指导教师：黄晓红　教授

2020 年 03 月 11 日

摘 要

机器学习是人工智能领域中一个重要的分支和研究内容,一般对它的研究都是从训练的过程中不断提高计算机程序的性能。使得计算机具备一定智能,通过机器学习模型的训练，从而使得机器学在很多场景被大量的应用。其中手写数字体的识别是在模式识别之中的一个重要且有意义的课题。神经网络和深度学习在图像处理领域很早就有了大量的应用，识别效果很好的认知网络模型常常是人们的研究对象。但是,复杂的网络模型使训练变得困难和耗时间。但是神经网络的执行要矩阵和向量的运算，随着可编程的 GPU 发展，基于图形处理器的通用计算已成为研究重点。

本文阐述的是卷积神经网络算法在机器学习中识别手写数字的应用，并且将手写数字过程移植到 GPU 上执行，利用并行计算的能力，极大的提高了手写数字识别速度，综合考虑学习率的变化特性，增强了网络的泛化性能。训练机器学习模型之后，把模型运用到 Android 手机上。在讲述理论的同时给出实例，并且附上应用的实例与代码。

关键字：卷积神经网络　深度学习　GPU 加速　手写数字识别　Android 应用

术语(nomenclature)：

- NN　Neural network　神经网络
- Ann　Artificial neural Network　人工神经网络
- CNN　Convolution neural Network　卷积神经网络
- OCR　Optical character recognition　光学字符识别
- PCA　Principal component analysis　主成分分析
- GPU　Graphics processing unit　图形处理器
- MLP　Multilayer Perceptron　多层感知机
- BP　Back Propagation　反向传播

Abstract

It is main topic in the field of artificial intelligence, machine learning mainly studies how to improve the performance of computer programs in the process of accumulating experience.Through machine learning, computers are equipped with intelligent means and are widely used in many fields. Character recognition is an important branch of pattern recognition and handwritten numeral recognition is a meaningful subject.Neural network and deep learning have been widely used in the field of image processing.However, the complex network model makes training difficult and time-consuming. However, the execution of neural network requires the operation of matrix and vector. With the development of programmable GPU, the general computation based on graphics processor has become the focus of research.

This paper describes the application of convolutional neural network algorithm in recognition of handwritten Numbers in machine learning, and the process of handwritten Numbers is transplanted to GPU for execution. By utilizing the ability of parallel computing, the recognition speed of handwritten Numbers is greatly improved, and the variation of learning rate is comprehensively considered to enhance the generalization performance of the network. After completing the model training, transplant it to an Android phone.An example is given along with the theory, and an application example and code are attached.

Keywords: deep learning of convolutional neural network; GPU; acceleration; handwritten number recognition; Android application

目 录

引 言

早在二十世纪 50 年代的时候研究员就模仿一些在动物神经网络之中发现的一些功能。Hubeld&Wiesel 察觉了动物视觉皮层细胞负责检测光学信号。由此受到启发，1980 年日本学者 Kunihiko Fukushima（福岛邦彦）提出了 CNN(Convolution Neural Network)的前身——Necogonitron[1][2][3]。

神经网络和深度学习在图像处理领域得到了广泛的应用。对于复杂的网络模型,通常需要良好的识别结果。但复杂的网络模型使训练变得困难,需要很长时间。为了获得较高的识别率,采用简单的模型,对 BP(Back Propagation)神经网络和卷积神经网络进行了单独的研究,并在 MNIST 的数据集上进行了验证。为了进一步提高识别结果,提出了一种综合深度网络,并对其进行了验证 MNIST 数据集[3][4]。

在之前的研究中，提出了一种新的识别阿拉伯数字方法，其中一组有 88 个功能，该特征集分为 72 个阴影特征和 16 个八分区特征。对于这项工作，使用从一个数据库中获得的 3000 个样本。将图像缩放到 32x32 像素大小，然后用灰度像素值定义。这种使用了 MLP 分类器有一个输入层，一个输出层，和一个隐藏层之间，因为 a 单隐层就足够计算均匀的了给定数据集的近似值。这个层的设置是如图 2 所示。每一层的神经元数量都是通过反复试验来选择的。每个神经元都有 sigmoid 作为激活函数。监督培训这个方法是在 2000 年的帮助下执行的训练样本。采用反向传播算法进行训练的输入-输出关系的识别手写数字的问题[5][6]。

第1章　绪论

1.1 研究背景

古往今来，最基本社交方式就是口语，但人们总是将自己的想法诉诸于文字，将自己的思想留下给后人，尽管在计算机发展如此迅速的当下，手写也是我们必不可少的社交方式。但如果将手写和计算机结合起来，既保持手写的“真”，又保持计算机的“便捷”，变成了当下亟待解决的一个重要问题。

在现代中，以人工神经网络和深度学习为代表的现代算法在图像处理领域得到了广泛的应用。对于复杂的网络模型，通常需要良好的识别结果。但是复杂的网络模型使训练变得困难，并且需要很长时间。为了用简单的模型获得更高的识别率，分别对 BP 神经网络和卷积神经网络进行了研究，并对 MNIST 数据集进行了验证[1][3]。

人工神经网络具有很强的自学习能力、自适应能力、分类能力、容错能力和快速识别能力，在字符识别中得到了广泛的应用。近年来，随着深度学习的发展，卷积神经网络(CNN)在图像处理中得到了广泛的应用。CNN 具有局部连接、权值共享和汇聚等特点。CNN 模型提取的特征具有良好的描述能力。

自早期模式识别以来,人们已经知道自然数据的可变性和丰富性,即它的语言、符号或其他类型的模式,几乎不可能完全通过手工建立一个准确的识别系统。因此,大多数模式识别系统都是使用自动学习技术和手工制作的算法组合起来的。

新世纪以来，多媒体技术以及游戏产业中 3D 渲染促进了 GPU(Graphics Processing Unit)的快速发展。如今的 GPU 逐渐走向成熟，不仅在晶体管的规模、时钟频率、并行能力指标上有了很大的飞跃，可编程性更是让人们看到 GPU 的应用前景蕴含巨大的潜力，其针对像素的运算甚至使用 GPU 来做通用计算成为可能。通用计算的领域包括利用 GPU 来实现矢量、矩阵等代数运算，

一些相对复杂的应用问题的求解。现如今 GPU 已经进入通用计算的主流，其自身所持有的特性已经愈发的适合于通用计算

1.2 神经网络发展及历史现状

1) **第一阶段——启蒙时期**

这是神经网络的提出阶段，1943 年由生物学家 McCulloch W.S.和数学家 Pitts W.A 合作，Rosenblatt 证明 只要有两层的感知器就可以进行分类功能，他还提出了以隐层处理元件组合形成的三层感知器的科研目标[3]。

2) **第二阶段——低潮时期**

由于 Minsky 和 Papert 的深入研究，认为不能解决分类问题和不能实现“亦或”逻辑关系，从此神经网络陷入低迷

3) **第三阶段——复兴时期**

Hopfield 神经网络由一组非线性微分方程组成的，如下：

$$C_i dU_i/dt=\sum_{i=1}^{N} T_{ij} f_j[U_j]-u_i/R_i+I_i,(i=1,2...N) \quad (1-1)$$

这种模型用电子电路实现，Hopfield 的模型进行非线性描述，还对其表述出动力方程和学习方程。

1990 年之后，LeCun 确立了 CNN 的现代结构，之后改进模型。他们设计了一种多层的人工神经网络，被称为 LeNet-5(如下图)。能够把手写数字区别出来。和其他 NN 相同，LeNet-5 也能使用 backpropagation 算法训练。LeNet-5 在手写字符识别等模式识别领域的成功应用引起了学术界对于卷积神经网络的关注。与此同时，卷积神经网络在语义识别碰撞检测、人脸识别等方面的研究也慢慢开始。

4) **第四阶段——全面研究时期**

2012 年，Kirizhevsky 提出的 AlexNet 在大型数据库 ImageNet 的图像分类竞赛以巨大优势的夺冠了，使得了卷积神经网络成为了重点研究对象，此后，不断提出新的卷积神经网络模型，像 Visual Geometry Group，还有 Google

的 GoogLeNet 这些神经网络刷新了 AlexNet 在 ImageNet 上创造的记录，CNN 和经典的算法结合，使得了在大多领域都有了应用，例如面向视频的行为识别模型-3D 卷积神经网络。还有通过迁移学习的 CNN 在 sample 容量较小的图像识别数据库上准确度获得提高。

1.3 识别手写数字意义

1) 阿拉伯数字是世界通用的数字符号，有着极其广泛的使用人群，所以各国此研究有着很大的使用人口基数。所以具有普适性
2) 手写体数字识别在有些的场景下应用特别多，比如邮政编码自动识别系统，税表以及银行支票自动处理系统，自动邮件分类和财务报表等领域，科学家门提出了大量的方法。随着机器的硬件和理论的发展，人们面对这些问题的探索采用了新的方式方法。
3) 字符识别系统可以做出巨大贡献以推进自动化进程并能改善人与人之间的互动，机器在许多方面都有很大的应用，包括办公自动化，核查和各种各样的银行业务，业务和数据输入应用程序。因此，如果对人工神经网络的改进加以重视，将会在脱机手写数字识别领域得到广泛的应用。
4) 可以在对手写数字识别技术实践运用的基础上运用于其领域，比如在研究英文字母上，阿拉伯字母[5][6]。

第2章　神经网络的基本概念

2.1 神经网络概述

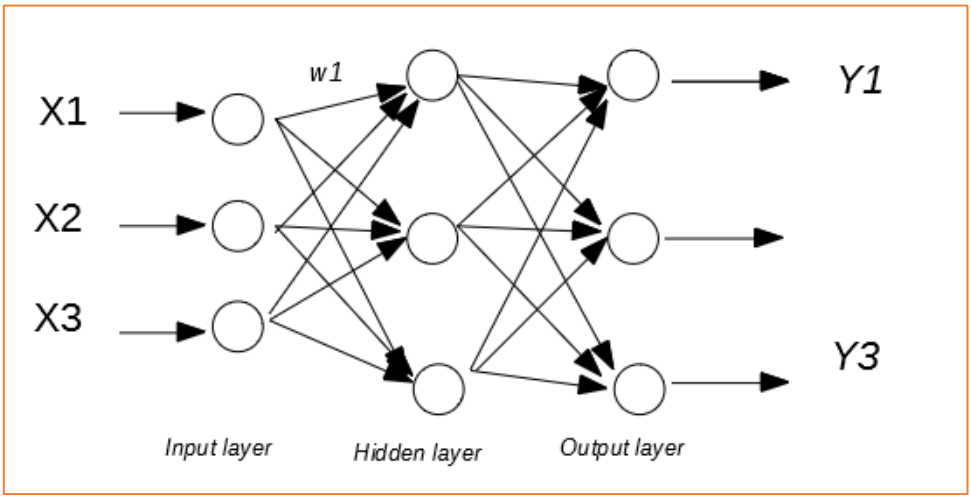

图 2-1. Structure of BP Ann

人工神经网络(简称 ANN)是一种信息处理范式，其灵感来自于大脑等生物神经系统处理信息的方式。它是由大量高度互联的处理元素(神经元)组成的，它们协同工作以解决特定的问题。

2.2 神经网络的种类

2.2.1.　卷积神经网络 CNN

1980 年,提出了新的认知机器,并首次介绍了 CNN 的概念。它成为了第一个深度学习的模型。2003 年,总结出了 CNN 模型。如下图所示,CNN 是一种多层次的前反馈网络。每一层由多个二维平面组成，由多个神经元组成。在下图中,C1 和 C3 是卷积,S2 和 S4 被采样。卷积层和子采样层可以有多个层。一般来说,CNN 的第一层是卷积和次采样的交替层,是 CNN 深度的反映[8][9]。

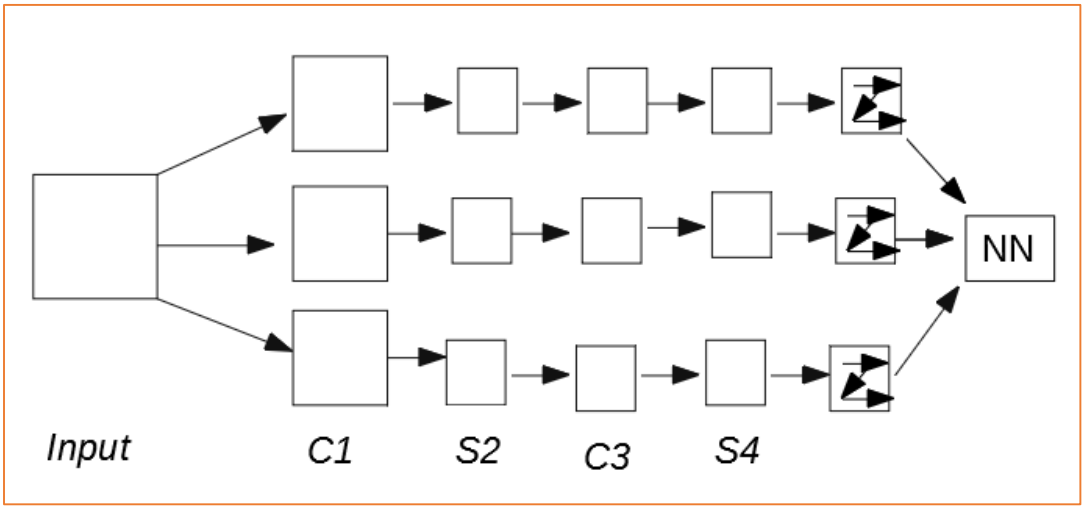

图 2-2 The structure of convolution nerve

卷积是图像中常用的一种算法识别。它指的是输出图像中的每个像素的小区域的像素加权输入图像的位置。这个小区域叫做局部经验场中，区域权重称为卷积内核。偏置项是在输入图像之后添加的，通过激活，得到特征图函数。下式子给出卷积层的形式:

$$X_j^l = f\left(\sum_{i \in M_j} X^{l-1} * K_{ij}^l + b_j^l\right) \tag{2-1}$$

其中 l 是层数，Xlj 是卷积层 l 的第 j 个特征图，K 是卷积核的数，b 是偏差，f 激活函数用于激活神经元。池化过程是进行降维处理的过程，与卷积过程类似，设定池化窗口，采用激活函数进行输入值的池化，此时，需要注意由于函数存在饱和死区，可能会导致梯度消失。卷积神经网络有三个核心，局部感知野——表示卷积过程中存在卷积和输入值重合部分，权值共享——卷积窗口可以维持不变，下采样——即池化操作。

2.2.2. 循环神经网络 RNN

循环神经网络(RNN)由于其建模序列数据的优势而成为一种流行的研究趋势。最近，RNN 在各种各样的任务上取得了很多令人印象深刻的应用，在语义识别等自然语言处理的问题上，包括语言建模，语音识别，情感分析。但是长时间的学习依赖于循环神经网络是十分困难的任务，因为梯度的消失和爆炸式的增长。结果，RNN 能够短时间内学习而不能长时间的独立，人们花费的了很长的时间攻克这个问题，其中最著名和成功的案例就是被称为 LSTM(Long Short-Term Meory)架构。LSTM 使用随机梯度下降代替了 tanh、logistics 函数在隐藏单元以至于能够存储之前的值，所以被称之为存储单元[10][11][12]。这样的信息循环存储入下图

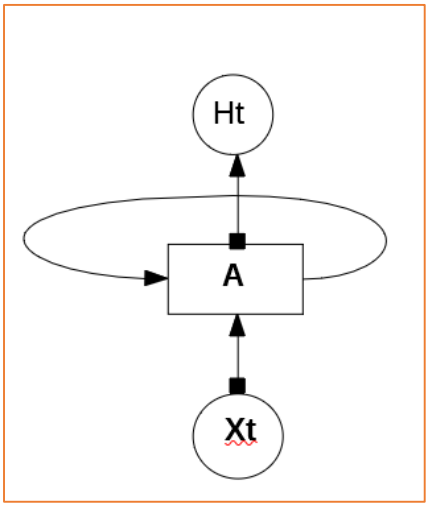

图 2-3 循环卷积神经单元

上图中，Xt 是一些输入，Ht 是输出值，这个循环过程有效的帮助信息进一步传通。其实 RNN 和一般的神经网络是有共通之处的。只要把 RNN 分解开来看，其实他是多个网格的组合，每一个网格都会将信息传递到后者。下图为循环展开后的图。

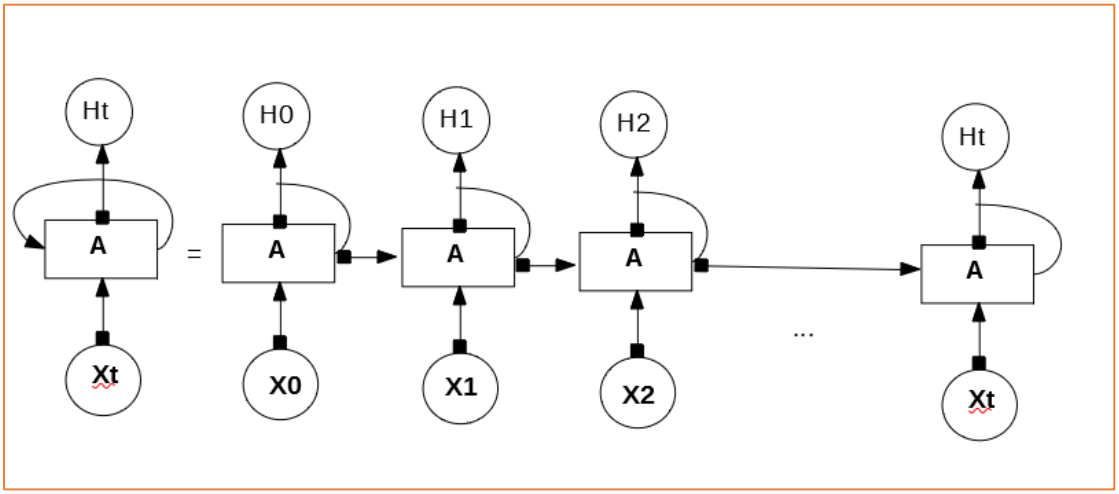

图 2-4 循环卷积网络处理过程

链式性质揭示了 RNN 与序列列表的密切相关，其可以很自然的被应用于这样的数据。过去几年 RNN 已经被成功的应用于许多问题. 演讲识别，语言模式，文本翻译等。这些存在的问题，都使用了一种特殊的网格，它归属于 RNN 网格，被称之为 LTSTMS 网格。当存在这些问题时，使用这个特殊网格较普通的网格有很好的适用性，几乎所有好的结果都与之相关。

循环的优点在于能够将前序任务做联系，例如在一个语言面 0 型中，根据前期出现的词语对后期可能出现的情况做预测，所以循环模式具有很好的学习性及可预测性。

2.3 可运用在神经网络中算法

2.3.1. BP 算法

神经网络学习算法，因其良好的非线性映射能力和柔软的网络结构，现被广泛应用于模式识别、数据预测、系统辨认图像处理以及函数拟合等各个领域。最近，研究员如何对传统 BP 网络优化，使其能够大大提高收敛的速度，从而避免陷入局部最优解作了大量工作。为此生成了许多的方案。

BP 神经网络通常具有学习和识别两个过程，在 BP 的学习处理中，分为正传递处理和误差反向传递处理两个阶段，输入信号是从外部传播到输出层并且通过输入层和隐含层神经元层的处理层给出结果。若不是预期的输出结果，则转换为反向扩展处理，网络输出的真值和误差沿原耦合访问返回。在识别处理中，除了输入层和输出层的数据外，权值没有变化。BP 神经网络的结构如 Figure1 所示[11][12]。

BP 是一种多层前馈网络所使用的监控式学习算法，主要的原理是使用线性规划中的最快下降法，使得权值沿着误差函数的负梯度方法改变，进而让实际输出与期望输出的方达到最小。

其过程如下：

i. 初始权值设为随机数

ii. 提供标签，也就是输入向量 Xi 和其输出 Yi，其中 i∈1, 2, 3…N

iii. 计算实际的输出，使用激活函数和公式 1，逐级计算输出值，得到最后的输出值 Oi

其中激活函数以为 tanh 或 logistics，表达式如下

$$tanhx=\frac{sinhx}{coshx}=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2-2}$$

$$logistics=\frac{1}{1+e^{T-\theta}} \tag{2-3}$$

激活函数为

$$y_i=f\left(\sum_{i=0}^{N-1} W_{ij}x_i-\theta_i\right) \tag{2-4}$$

iv. 变换权值，递归调用函数输出，每次都调整权值

$$W_{ij}(t+1)=W_{ij}(t)+\Delta w_{ij} \tag{2-5}$$

$$\Delta W_{ij}=-\_\frac{\partial\frac{1}{2}\sum(y_j-o_j)^2}{\partial W_{ij}} \tag{2-6}$$

式中 $W_{ij}(t)$表示 t 时刻时 k-1 层的第 j 个神经元到 k 层的第 i 个神经元的连接权值，y_j表示节点 j 的输出期望，o_j表示节点 j 的实际输出，下划线“_”表示大于零的增益即学习率。

v. 返回第 ii 步重复，是输出层单元误差变得足够小至满意为止。

2.3.2. 随机梯度下降算法

SGD 算法是梯度下降算法的一种大幅简化，有人提出(如 Rumelhart et al.(1986))使用梯度下降(GD)来最小化经验风险 $En(f_w)$，每次迭代升级权重 w 在 $En（f_w）$梯度的偏置值。其中 E（f）与 En(f)函数如下。

$$E(f)=\int l(f(x),y)dP(z) \tag{2-7}$$

$$E_n(f)=\frac{1}{n}\sum_{i=1}^{n}\int l(f(x_i),y_i) \tag{2-8}$$

简化后的梯度算法如下：

$$\omega_{t+1}=\omega_t-\gamma\frac{1}{n}\sum_{i=1}^{n}\nabla_\omega Q(z_i,\omega_t) \tag{2-9}$$

其中 γ 是一个充分的选择中获得的，在足够的规则下的一种假设，当初始化的ω_0接近最佳的情况下，且当γ足够的小，这个算法的所取得的线性收敛是$-\log\rho\sim t$，其中ρ表示剩余的偏差。

通过正定矩阵Γ_t在最优时，t 接近成本的黑森数的倒数，从而可以得到更加优化的算法，同时也可以由从γ被替换为更小值而获得[13][15][16]。其公式如下:

$$\omega_{t+1}=\omega_t-\Gamma_t\frac{1}{T}\sum_{i=1}^{n}\nabla_\omega Q(z_i,\omega_t) \tag{2-10}$$

还有一种二阶梯度下降算法(2GD，second order gradient descent)，这是一种众所周知的变体牛顿算法，在足够准确的规律性假设下，当 w0 与最优值足够接近时，二阶梯度下降达到二次收敛。当成本是二次和缩放矩阵 Γ 精确的情形下，一个迭代后的算法达到最优。如果足够平滑可以得到$-\log\log\rho\sim t$。

随机梯度下降(SGD)算法是一种高度简化的算法。不是精确地计算 En(fw)的梯度，而是每一次迭代都基于一个随机选取的例子 zt 来估计梯度:

$$\omega_{t+1}=\omega_t-\gamma\nabla_\omega Q(z_t,\omega_t) \tag{2-11}$$

随机过程{wt, t= 1，…}取决于在每个迭代中随机选取的示例。希望上面式子的行为与预期一致，尽管这个简化过程会带来噪声。因为随机算法不需要记住在之前的迭代中访问了哪些实例，所以它可以在部署的系统中动态地处理实例。在这种

情况下，随机梯度下降直接优化了预期风险，因为样本是从地面真实分布中随机抽取的。

随机梯度下降的收敛性在随机逼近的文献中得到了广泛的研究。收敛结果通常要求在满足条件的条件下增益递减。Robbins-Siegmund 定理(Robbins and Siegmund(1971))提供了在温和条件下建立几乎确定收敛的方法(Bottou) (1998))，包括损失函数并非处处可微的情况。

2.4 神经网络中的基本概念和参数

神经网络由大量的神经元相互连接而成。每个神经元接受线性组合的输入后，最开始只是简单的线性加权，后来给每个神经元加上了非线性的激活函数，从而进行非线性变换后输出。每两个神经元之间的连接代表加权值，称之为权重（weight）。不同的权重和激活函数，则会导致神经网络不同的输出。

神经网络使用的反向传播算法（BP）可以被描述如下, 结合上图 Figure 1 进行讲解：

1) Input X_i:设置对于 a1 的相对应的激活函数值

2) 向前传播：计算相对应的权重输入和激活值 a1 l=2，3，4，5…

$$Z^l=W^l d^{l-1}+b^l$$

$$a^l = \sigma(Z^l)$$

其中，$\sigma(z)$表示激活函数[15][16]。

3) 计算输出层的误差

$$\delta^L=\nabla_a\cdot\sigma'(z^L) \tag{2-13}$$

反向传播误差：计算 δ^l，其中 l=L-1，L-2,…2.

$$\delta^l=(w^{l+1})^T\delta^{l+1}\cdot\sigma'(Z^L) \tag{2-14}$$

4) 输出：cost 函数的梯度被设定为如下

$$\frac{(\partial C)}{(\partial W^J_{jk})}=a_k^{(l-1)}\delta_j \tag{2-15}$$

$$\frac{\partial C}{\partial b^l_j}=\delta^l_j \tag{2-16}$$

5) 激活函数

人工神经网络由同样也是由神经元组成，不过是人工神经元。每个人工神经元都有一个激活函数来决定被激活神经元的激活率。激活功能的概念是由中枢神经系统中的生物神经元所启发的。如下图生物神经网络和 ANN。

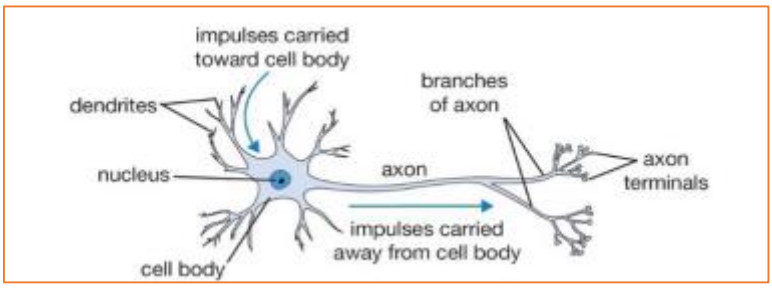

图 2-5 神经元示意图

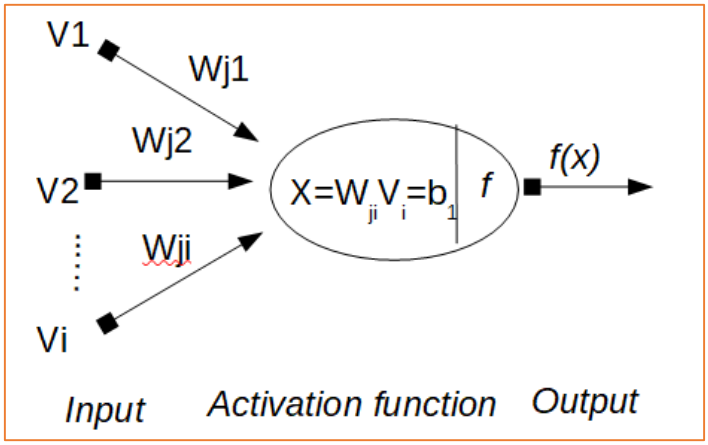

图 2-6 神经元处理输入和输出

输入数据 v 与权值 w 相乘，一个激活函数 f 被用来决定那个特定的人工神经元是否被激活。激活函数大致有如下几种，其函数表达式和函数图像用 python 画出。

1) **Logistics function**

logistics 函数是神经网络中常用的经典激活函数，它与生物神经元的激活速率有相似的表达。逻辑函数有两种类型，即 S 状弯曲函数和双曲正切函数(tanh)。这个 S 形函数是一个增长函数，它在线性和非线性行为之间保持平衡。Sigmoid 函数的取值范围为 0 到 1。图像如下:

$$f_1(x)=\frac{1}{1+\exp(-ax)} \tag{2-17}$$

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-10, 10, 0.1)
y = 1. / (1. + np.exp(-x))

plt.plot(x, y)
```

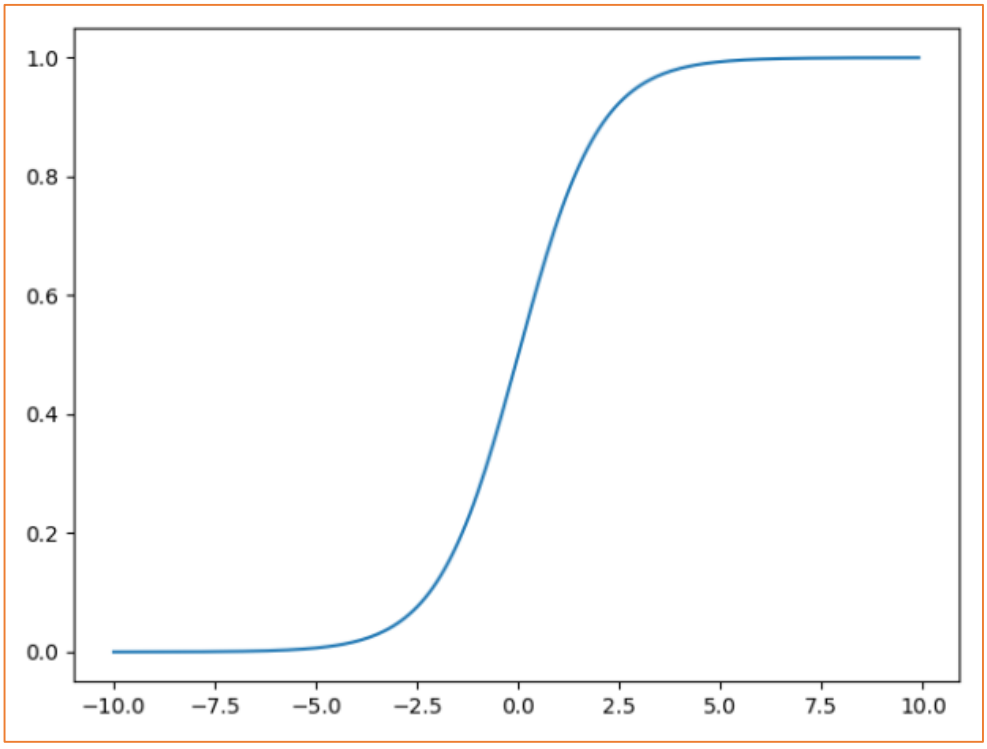

图 2-7 激活函数

落在 0 或 1 区域的数据的梯度几乎为零。因此，在反向传播训练过程中，数据不会通过神经元来更新权值参数。因此，这个消失梯度问题导致网络性能下降。但是，可以通过使用更好的初始化权值参数和预训练算法来克服这个问题[19][20]。

2) **Multistate activation function (MSAF)多状态激活函数**

$$f_2(x)=\frac{\exp(2x)-1}{\exp(2x)+1} \tag{2-18}$$

多状态激活函数具有多个额外的输出状态，MSAF 具有两个版本，N-阶对称 MSAF 函数如下：

$$f_3(x)=\frac{1}{1+\exp(-x-r_1)}+\frac{1}{1+\exp(-x)} \tag{2-19}$$

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-10, 10, 0.1)
r1 = 0
y = (1. / (1. + np.exp(-x))) + (1. / (1. + np.exp(-x -
r1)))

plt.plot(x, y)
plt.show()
```

图像如下：

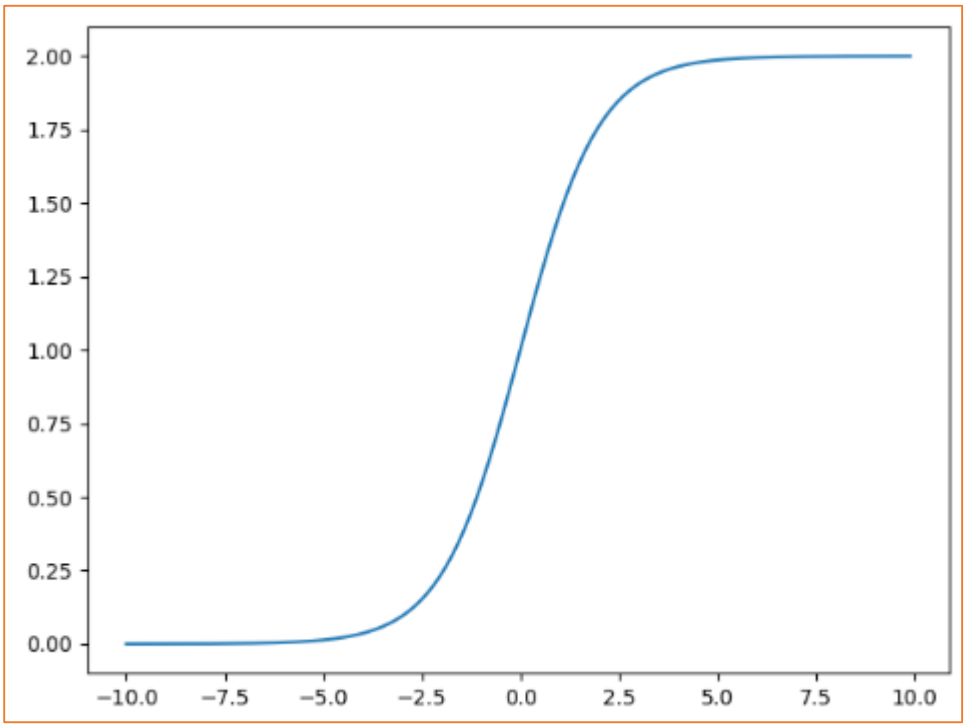

图 2-8 激活函数

3) Rectified linear unit and leaky rectified linear unit

Rectified linear unit (ReLU)，由于其简单性，缩短了训练计算时间，已成为 DNN 的默认激活函数。但是，在 DNN 训练过程中，ReLU 可能会超出预定的边界，因此，需要仔细调整学习速率。这个问题会导致神经元在负的激

活功能下，在整个训练过程中不再被激活。

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(-10, 10, 0.1)
r1 = 0
y = np.maximum(0, x)

plt.plot(x, y)
plt.show()
```

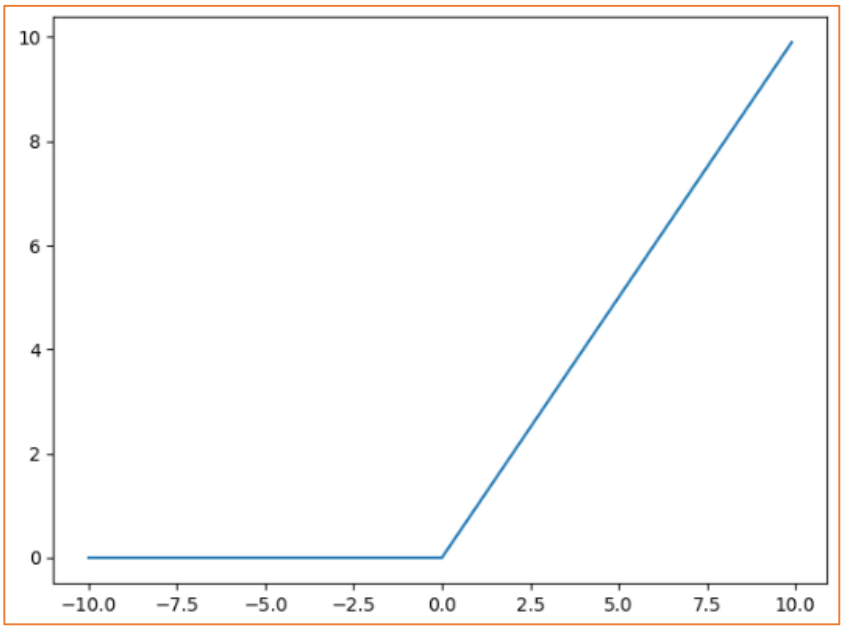

图 2-9 ReLU 激活函数

对网络性能的另一个重要影响是在训练开始时初始化权值。由于复杂的方法通常不能提供稳定的性能，所以常常要使用随机的方法或者是简单的方法。作为一个近似的过程与分段线性前馈神经网络中引入，在对称线性饱和激活函数用于隐藏层，ReLU 激活函数被认为是十分有效的。因为它可以用于非线性系统以一种十分相似的方式运用在其他相似的系统上。此外，与对称线性饱和激活函数相比，它的评估可能更有效。

本文中只需要用到 ReLU 函数，不介绍过多其他

6) 学习率 Learning Rate

学习率(learning rate)是深度学习模型训练过程中非常重要的一个超参

数。学习率是梯度下降法在搜索过程中的步长，步长过大将导致模型震荡，步长过小又使得模型找到最优值的速度很慢并且可能陷入局部最优解。在训练中，模型最终的收敛效果很大程度地受到学习率的影响。通常情况下会根据经验设置学习率，例如使用不同大小的学习率进行训练模型，观察比较模型收敛情况，选取最合 适的值。选择合适的学习率非常重要。在模型的训练中，当模型收敛时，学习率与代价函数的值慢慢减小。

在上面的分析中提到当学习率偏大时，准确率是会下降的，因此除了通过准确率大小判断模型当前的训练情况，还可以通过观察准确率的增量大小更具体地获取模型离最优值的距离。在经过多为科学家的研究之下，选择使用 0.01 作为初始学习率，在训练的前几个世代增大学习率，之后再根据准确率的大小及其增量动态减小学习率。首先通过一次不使用优化算法的预训练得到大致的模型测试准确率收敛值 A。

7) **动量(Momentum)**

在修正网络权值时，利用动量的方法不仅考虑了误差对梯度的影响，也对曲面的变化趋势进行了分析。它的作用可以描述为一个低通滤波器，可以忽略小的变化[4]。当一个值大于实际最小值时，这种方法可以防止选择最小值，否则它将比真实最小值更大。其原因是该方法将之前的权值的变化作为调整权值的重要因素。公式可以表示如下：

$$\Delta \omega_{ij}(K+1)=\eta \delta_i P_j+m*\delta \Delta_{ij}(K) \tag{2-20}$$

其中 m 是动量， η 是学习率， δ 是变化量，P 是一些层的输入。

第3章 基于 BP 神经网络的手写数字识别

3.1 识别的流程

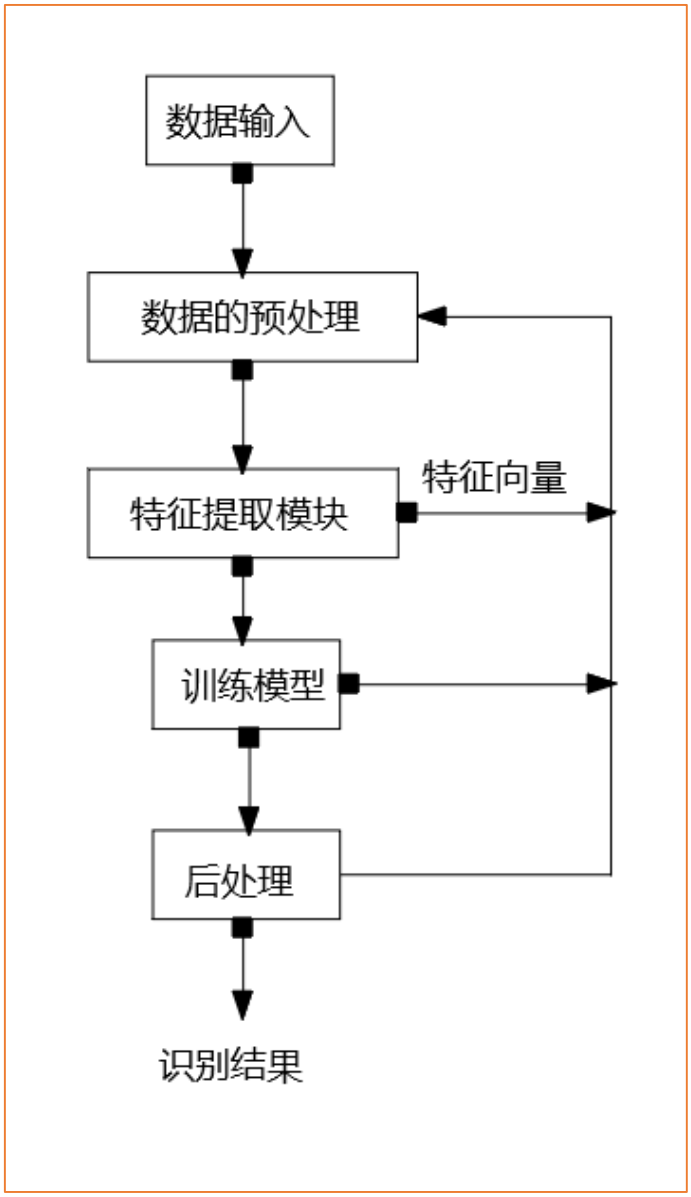

图 3-1. Recognition Procedures

预处理是对采集到的图像进行识别前的必要处理工作，包括几何校正，去噪等操作。特征提取出对所用方法有效的特征。识别阶段利用上一步抽取的特征对待识别的字符进行识别处理[19][20]。

3.2 识别方法

3.2.1. 基于结构特征方法

对于一个复杂的模式，常常采用分解的方法将其划分成为若干较简单的子模式乃至基元，通过对基元和子模式识别的综合集成来完成复杂模式的整体识别。结构模式识别方法是以同类模式具有相似结构为基础的识别方法，从字符的描边或整体架构上提取基本特征。结构指的是组成一个模式的基本单元（简称基元）之间的关系。例如，拼音的基元是拉丁字母，几个字母组成单词，在识别一个单词时，如果能够识别组成这个单词的各个字母以及它们的结构关系，即可识别出这个字[16][18]。

1) **建立统计特征的方法**

建立在统计数学，主要理论为贝叶斯决策理论，由模式紧致性，距离和相似性度量等概念和假定等组成的统计决策。该方法是对字符整体的分析。

2) **逻辑特征法**

这类问题的特征选择一直是一个问题。这种方法确立了关于知识表示及组织，所以此方有不错的结果，但是当样品有缺陷，背景不清晰，规则不明确甚至有歧义时，效果不好。

3) **模糊模式法**

能表现出全体的、重要的特性，模糊模式有相当不匀称的抗干扰与畸变能力，所以有一定概率让 sample 受到一定的干扰与畸变，但准确合理的隶属函数关系却往往难以建立。近年来的研究将其引入神经网络方法形成模糊神经网络识别方法。

3.3 MNIST 数据集

MNIST(Modified National Institute of Standards and Technology)手写体数字数据库有一个包含 60,000 个示例的训练集和一个包含 10,000 个示例的测试集。NIST 首先以 SD-3 训练集，把 SD-1 作为测试集。因为，相比于 SD-1，SD-3 的

表现更清晰，更容易识别。原因在于，SD-3 是在人口普查局工作人员中收集的，而 SD-1 是在高中生中收集的。

MNIST 它是 NIST 提供的一个较大集合的子集。数字的大小已标准化并集中在固定大小的图像中。这些文件不是任何一种的标准图像格式[13][17]。只有通过自己编写的程序才能阅读它们。通过 python 代码在其部分显示结果如下：

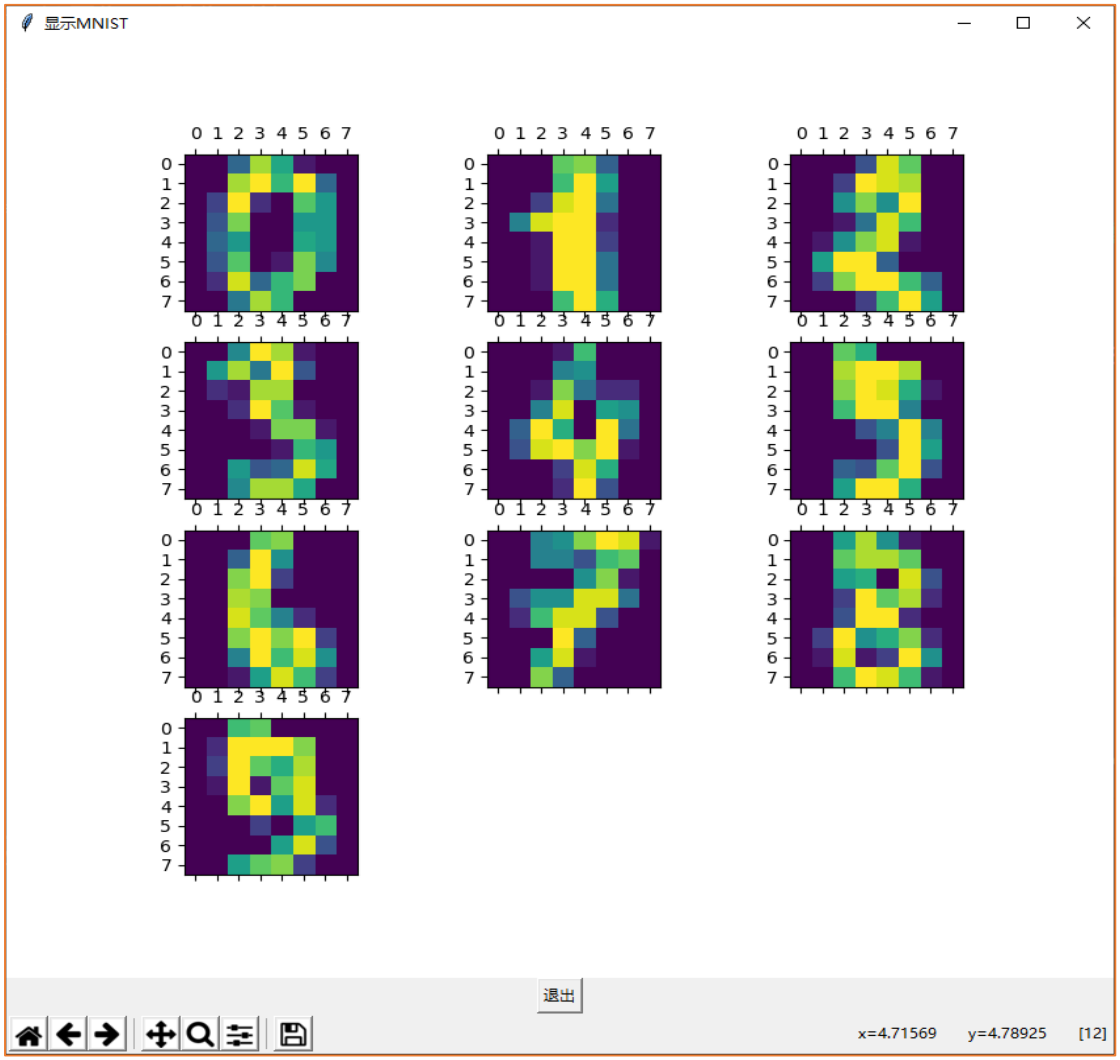

图 3-2 python 显示出数字

Python 代码在附件代码一。

此数据来自 NIST 的原始黑白(两层)图像被归一化，以适应一个 20x20 像素的盒子，同时保持它们的长宽比。由于归一化算法使用了反锯齿技术，生成的图像包含灰度级。通过计算像素的质心，将图像居中到 28x28 图像中，并对图像进行平移，从而将该点定位到 28x28 区域的中心。数据以非常简单

的文件格式存储，用于存储向量和多维矩阵[20]。

3.4 神经网络的初次运行

在做足一切的理论工作之后，在通过 Python 的 numpy 进行神经网络代码的编写，从而自己真正的实现神经模型。Python 代码在附件中。

Backpropagation 被使用在多层向前神经网络上，多层向前神经网络由以下部分组成：输入层(input layer), 隐藏层 (hidden layers), 输入层 (output layers)每层由单元(units)组成，输入层(input layer)是由训练集的实例特征向量传入。经过连接结点的权重(weight)传入下一层，一层的输出是下一层的输入，隐藏层的个数可以是任意的，输入层有一层，输出层有一层，每个单元(unit)也可以被称作神经结点，根据生物学来源定义，以上成为 2 层的神经网络（输入层不算），一层中加权的求和，然后根据非线性方程转化输出作为多层向前神经网络，理论上，如果有足够多的隐藏层(hidden layers) 和足够大的训练集, 可以模拟出任何方程。

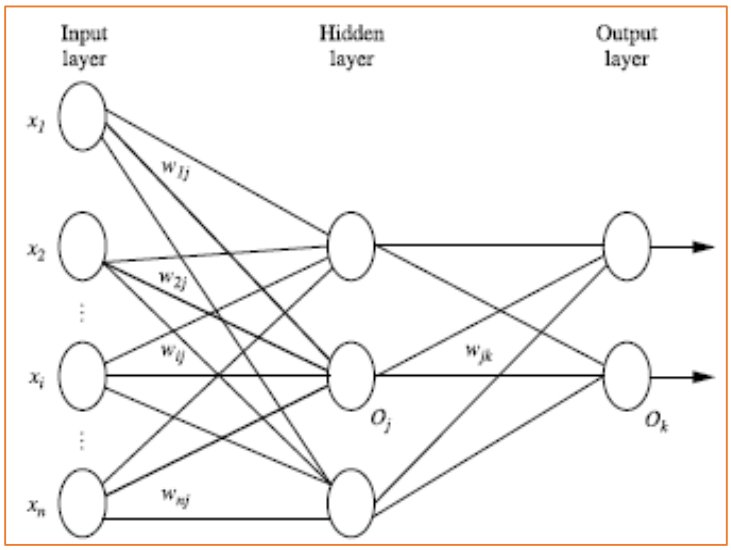

图 3-3 神经网络示意图

运行神经网络得到结果如下：

```
[0, 0] [0.00116574]
[0, 1] [0.99836377]
[1, 0] [0.99821145]
[1, 1] [-0.01229332]

Process finished with exit code 0
```

图 3-4 运行“亦或”结果

通过这个神经网络实现“亦或”操作，相同为‘0’，不同为‘1’。

调用测试主函数，首先通过 X=[[0, 0], [0, 1], [1, 0], [1, 1]]输入训练集合，在通过 y=[0, 1, 1, 0]作为标签，调用神经网络的 fit(X,y)进行训练。

可以看到[0,0]输出的结果为 0，[0,1]输出结果为 1，[1,0]输出的结果为 1，[1,1]输出结果为 0。神经网络就不像之前数学家 Minsky 和 Papert 证明的那样不能实现“亦或”操作。

3.5 使用 BP 神经网络的识别情况

识别结果如下：

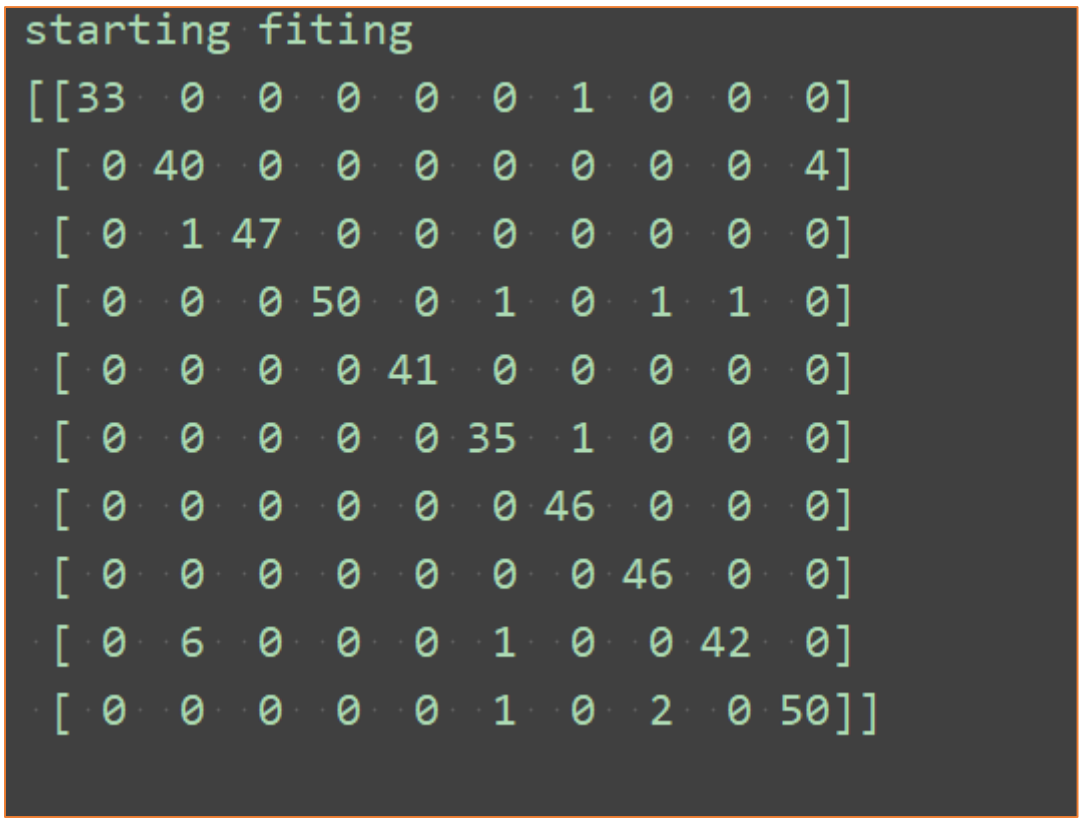

图 3-6 识别数字的结果

从结果来看(对角线)，BP 神经网络的表现已经很不错了。

第4章　卷积神经网络

卷积神经网络(CNN)是一种特殊类型的神经网络，特别适合于计算机视觉应用，因为它具有对局部可以进行操作以及有分层抽象表示的能力。在计算机视觉中，有两个关键的设计思想推动了卷积架构的成功。

卷积神经网络是一个具有多层网络结构的网络。它包括正向和反向传输，其中正向传输是指输入特征经过多层操作，最后在输出层输出特征。在每一层中，都需要添加一个激活函数。反向传输是指先将正向传输的结果与给定的样本标签进行误差计算。之后误差函数返回到每一层。最后，利用梯度下降算法更新神经网络权值和偏置参数。误差函数定义如下的数学表达式[20]：

$$E^n=\frac{1}{2}\sum_{k=1}^{C}\left(t_k^n-y_k^n\right)=\frac{1}{2}\left\|t^n-y^n\right\|_2^2 \tag{4-1}$$

其中 tk 表示第 k 个标签所对应的第 n 个实例，yk 表示第 n 个实例所对应的第 k 个网络的输出。输出单元在第 l 层的残差如下表示：

$$\delta^l=\frac{\partial E^n}{\partial u^l}=f'\left(u^l\right)\left(y^n-t^n\right) \tag{4-2}$$

对于误差函数和 W 的偏导数的表示如下：

$$\frac{\partial E^n}{\partial W^l}=\frac{\partial E^n}{\partial u^l}\frac{\partial u^l}{\partial W^l}=\left(\delta^l\right)^T x^{l-1} \tag{4-3}$$

$$\frac{\partial E^n}{\partial W^l}=\frac{\partial E^n}{\partial u^l}\frac{\partial u^l}{\partial b^l}=\delta^l \tag{4-4}$$

第 l 层的迭代表达式如下：

$$W^l=W^l-\alpha\frac{\partial E^n}{\partial W^l} \tag{4-5}$$

$$b^l = b^l - \alpha \frac{\partial E^n}{\partial b^l} \tag{4-6}$$

其中 α 表示学习率，W 和 b 表示权重矩阵和各自层的偏差。

4.1 卷积神经网络原理和过程及参数

卷积神经网络利用了图像的二维结构和相邻像素高度，因此，ConvNets 避免使用所有像素单元之间的一对一连接(即大多数神经网络的情况)，而倾向于使用分组的本地连接。此外，ConvNet 架构依赖于特征共享，因此每个通道(或输出特征图)都是通过在所有位置与相同的滤波器进行卷积而生成的。与标准的神经网络相比，ConvNets 的这一重要特性导致其架构依赖的参数要少得多。此外 ConvNets 还引入了一个池化步骤，该步骤提供了一定程度的转换不变性，使体系结构较少受到位置上的小变化的影响。值得注意的是，由于网络接受域的大小增加，池化层也允许网络逐渐看到更大的输入部分。随着网络深度的增加，多层架构会发送更多抽象的输入特征。例如，在物体识别的任务中，主张先将焦点集中在物体部分的边缘上，最终将整个物体覆盖在层次结构的较高层次上。以下的示意图概括了整个过程。

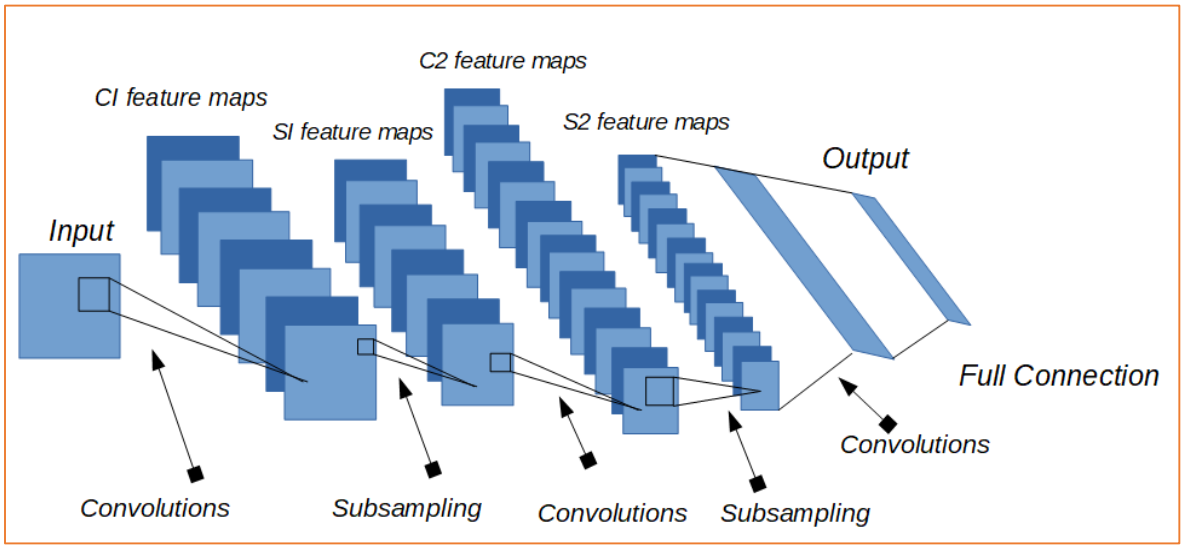

图 4-1. Illustration of the structure of a standard Convolutional Network

1) 卷积

假设神经网络的输入它可以是一个图像(如彩色图像，宽度和高度为32×32 的数据像素，深度为 3 的 RGB 通道)或一个视频(灰度视频，其高度和宽度为分辨率，深度为帧数量)，有宽度和高度(L×L)的传感器值，和深度与不同的时间帧相关联的实验视频。以下用更具体的例子来说明卷积

假设有 7*7=49 个输入特征，3*3=9 个输出特征。如果这是一个标准的全连接层，那么将拥有一个 49×9＝441 个参数的一个矩阵，每个输出特性都是每个输入特性的加权和。卷积允许我们只使用 9 个参数进行转换对于每个输出特性，不是查看每个输入特性，而是“查看”来自大致相同位置的输入特性。如下图所示，绿色部分为卷积核，一直滑动计算。

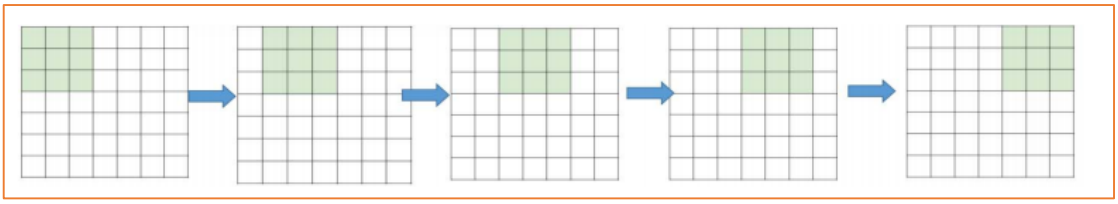

图 4-2 卷积运算过程

以下图都是卷积计算的模拟

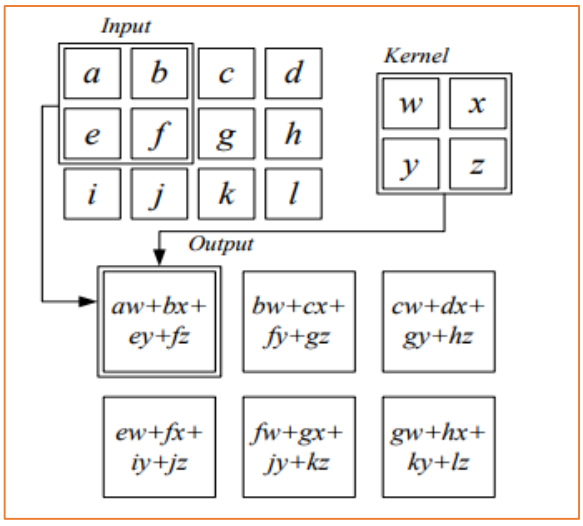

图 4-3 卷积模拟过程

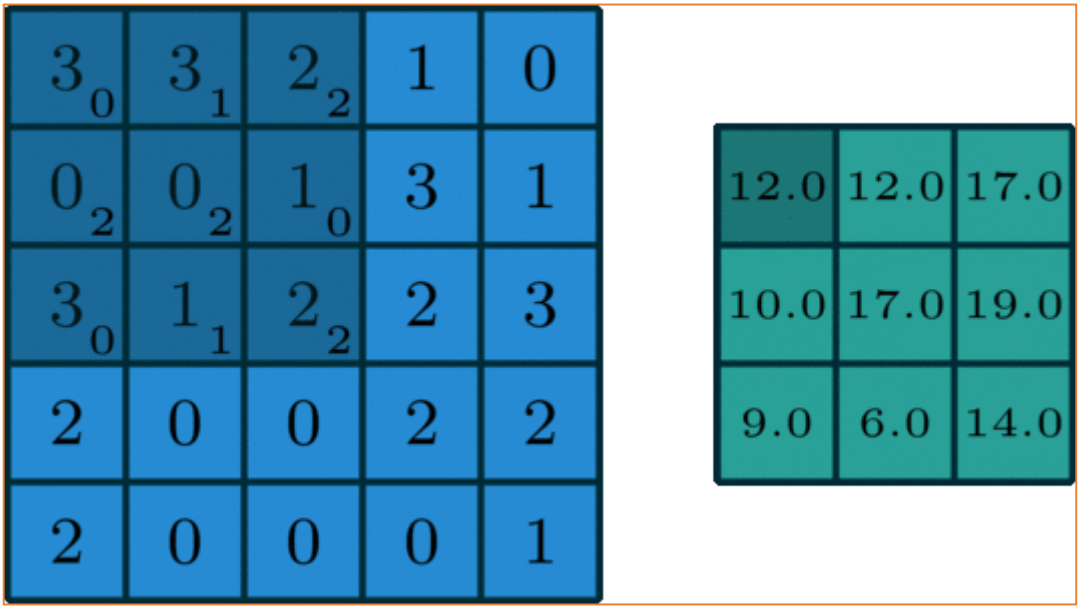

图 4-4 卷积过程

同时卷积核的不同选取，得出的结果可能会大相径庭。如下图：

图 4-5 中值滤波处理图片的结果

这个计算采用的的是中值滤波计算，即每次计算过程都取中位数。

中值滤波其卷积核为：

$$kernel=\begin{bmatrix}1 & 2 & 1\\ 2 & 3 & 5\\ 2 & 5 & 4\end{bmatrix} \tag{4-7}$$

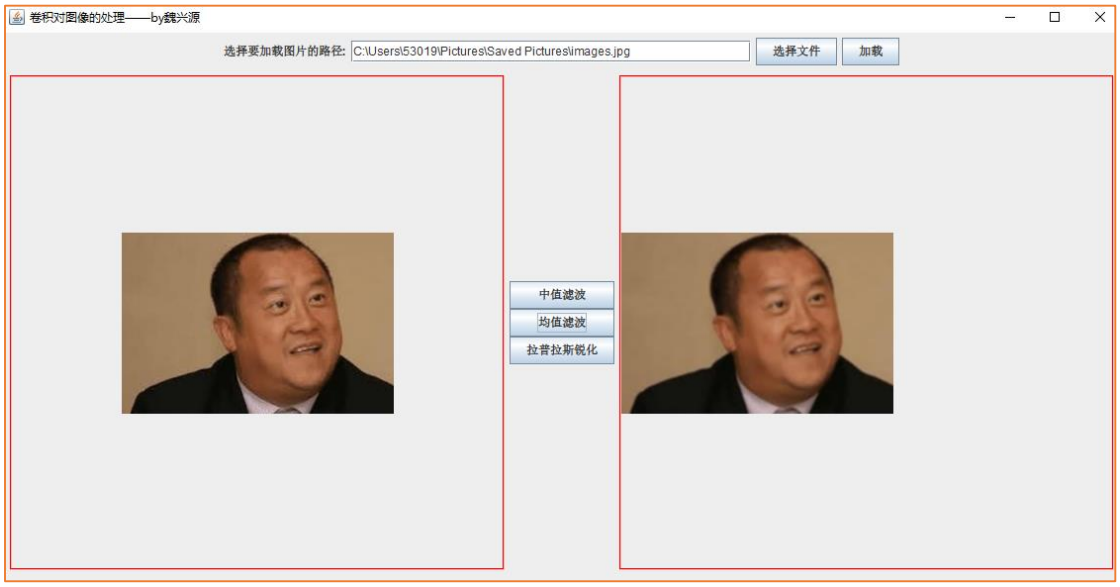

图 4-6 均值滤波处理结果

均值滤波其卷积核如下：

$$kernel=\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix} \tag{4-8}$$

拉普拉斯锐化过滤结果如下：

图 4-7 拉普拉斯锐化处理结果

其核心如下：

$$kernel=\begin{bmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{bmatrix} \tag{4-9}$$

2) 填充

我们观察到在滑动过程中，边缘基本上被“裁剪掉”了，将一个 5×5 的特征矩阵变成了一个 3×3 的特征矩阵。填充做了进行了一项精明的操作来解决边缘消失的问题。:用额外的空的像素值填充边缘(通常值为 0，零填入)。这样，内核在滑动时可以让原始的边缘像素位于其中心，同时扩展到边缘之外的伪像素，生成与输入相同大小的输出[24][25]。如下图所示：

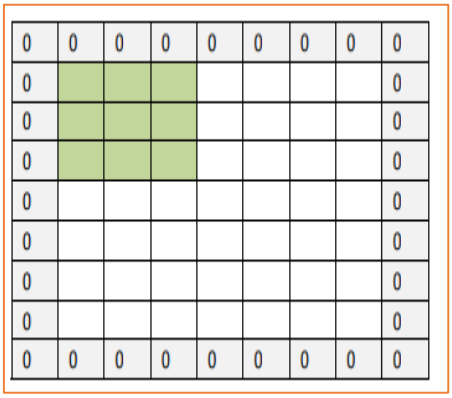

图 4-8 填充结果

3) 池化

下行采样层也称为池化层。将图像分割成小块的小区域，并为每个区域计算一个值。然后将计算值按顺序排列，输出新图像。该过程相当于模糊滤波，可以提高图像特征提取的鲁棒性。本文采用的池化方法是最大池化

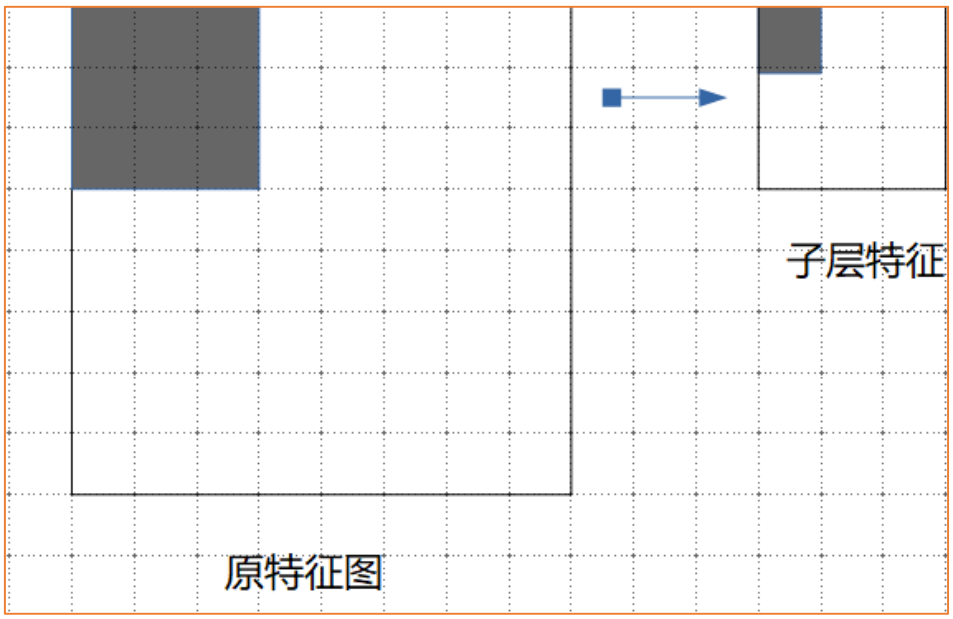

图 4-9 特征处理结果

距离说明，池化即取得区域中平均值或最大值，如下图所示：

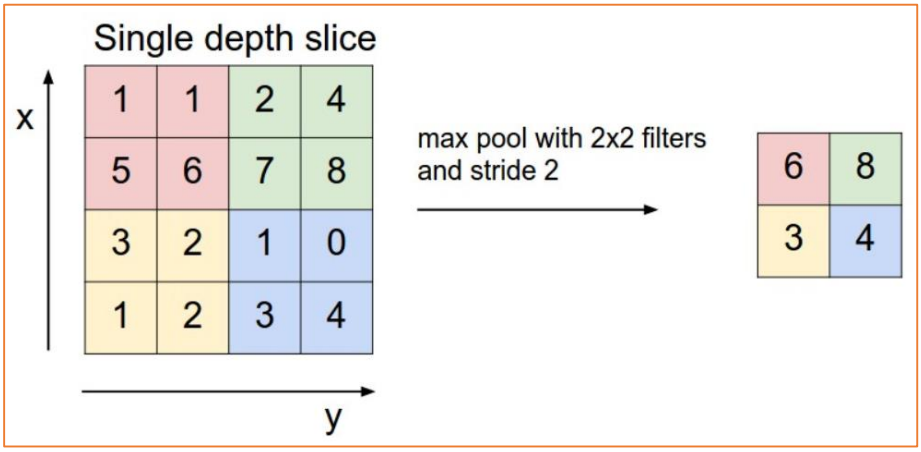

图 4-10 池化结果

上图所展示的结果是取某一个区域中最大的一个数，即上图左上角边粉色部分 2x2 的矩阵中 6 的值为最大值，右上角绿色部分 2x2 矩阵中 8 最大，左下角黄色部分 2x2 的矩阵中 3 最大，右下角蓝色部分 2x2 的矩阵中 4 最大，所以得到上图右边部分的池化结果：6 8 3 4。

4) 非线性

同样的 CNN 也需要激活函数使其非线性化，否则无论多少层隐藏层都只是线性叠加，所以同 NN 一样需要激活函数使其非线性化。

5) 全连接

全连接层与传统神经网络中神经元的排列方式相似。因此，全连接层中的每个节点都直接连接到上一层和下一层中的每个节点。全连通层的主要缺点是在训练实例中包含了大量需要复杂计算的参数。因此，我们试图消除节点和连接的数量。通过使用 dropout 技术，可以满足删除的节点和连接[26][27][28]。

4.2 具有判别信息的卷积神经网络

对于分类问题，为了更好地分类，定义了两种度量函数 E1 和 E2，分别表示同一类别的距离和不同类别之间的距离。

$$E_2=\frac{1}{2}\sum_{c'=1}^{C}\left\|M_c-Mc'\right\|_2^2 \tag{4-9}$$

$$M_c=\frac{\sum_{n=1}^{N_c} y_c^n}{N_c} \tag{4-10}$$

其中 yc 表示类别 c 的第 n 个实例的输出值，Mc 和 Mc’分别表示类别 c 和类别 c’实例的平均输出。所以新定义的误差函数表达式如下：

$$E=E^n+\lambda E_1-\eta E_2 \tag{4-11}$$

其中 λ 和 η 表示权重参数，新定义第 l 层输出单元的残差表达式如下

$$\delta'(l)=\frac{\partial E}{\partial u^l}=f'(u^l)(y^n-t^n)+\lambda\left(1-\frac{1}{N_c}\right)f'(u')(y_c^n-Mc)-\eta\frac{1}{N_c}f'(u^l)\sum_{c'=1}^{C}(M_c-M_c') \tag{4-11}$$

4.3 GPU

GPU，它已经彻底改变了深度学习的世界。图形处理单元最初用于游戏

目的，即为了获得更好的游戏分辨率而每秒显示更多的屏幕。深度学习网络使用大量的矩阵乘法，特别是卷积，用于前向传递和后向传播。gpu 擅长矩阵间的乘法运算。因此，数千个 GPU 核被用来并行处理数据。这加快了深度学习训练的速度。因此，成千上万的 GPU 核心用于并行处理数据，这个速度加速了深度学习的训练。以下为市场中常见的 GPU。

1) NVIDIA GTX TITAN XGE
2) NVIDIA GTX TITAN X
3) NVIDIA GeForce GTX 1080
4) NVIDIA GeForce GTX 1070

4.4 CUDA

CUDA 是一个并行计算平台和编程模型开发的 NVIDIA 通用计算图形处理单元(gpu)[29]。有了 CUDA，开发人员可以利用 gpu 的强大功能极大地提高计算应用程序的速度。在 GPU 加速的应用程序中，工作负载的连续部分在 CPU 上运行——这是为单线程性能优化的——而应用程序的计算密集型部分则在数千个 GPU 核心上并行运行。当使用 CUDA 时，开发人员使用流行的语言如 C、C++、Fortran、Python 和 MATLAB 进行编程，并通过一些基本关键字的扩展形式来表达并行性。来自 NVIDIA 的 CUDA 工具包提供了开发 gpu 加速应用程序所需的一切[30]。CUDA 工具包包括 gpu 加速库、编译器、开发工具和 CUDA 运行时环境。

GPU 最初被设计成图形加速器，在 90 年代变得更加可编程，在 1999 年 NVIDIA 的第一个 GPU 达到顶峰。研究人员和科学家们很快开始将该 GPU 出色的浮点性能应用于通用计算。2003 年，由 Ian Buck 领导的一组研究人员公布了 Brook，它是第一个被广泛采用的用数据并行结构扩展 C 的编程模型。Ian Buck 后来加入了 NVIDIA，并在 2006 年领导了 CUDA 的发布，这是世界上第一个在 gpu 上进行通用计算的解决方案。自成立以来，CUDA 生态系统迅速发展，包括软件开发工具、服务和基于合作伙伴的解决方案。CUDA Toolkit 包括库、编译器和用于部署应用程序的运行时库。

第5章　移动端设计

5.1 概述

智能移动设备已经改变了人类的生活，深度学习模型和机器学习模型的融合不断增加了目标设备。计算机技术的简化和移动电话的适应开辟了一条新的可能性之路。目标检测正从单目标识别向多目标识别发展。移动设备功能的增加使得我们从一个互联网社会向一个移动的[2]社会过渡。这种演变的一个例子就是TensorFlow，因为早期的 Android、Raspberry 和 iOS 版本已经发布了。这个机器学习的框架提供了一些语音和对象模式识别的样本，用自己的数据集模型进行测试，允许创建知识转移来训练自己的数据[27][28]。

5.2 Android 平台

Android 的发展始于 2003 年的 Android,Inc .。2005 年由谷歌收购。在测试版发布之前,谷歌和 OHA 中至少有两种内部版本的版本。测试版于 2007 年 11 月 5 日发布，软件开发工具包(SDK)于 2007 年 11 月 12 日发布。发布了几个 SDK 的公测试版。

2007 年 11 月 5 日，谷歌公司正式向外界展示了名为 Android 的操作系统，并且在这天谷歌宣布建立一个全球性的联盟组织“开放手持设备联盟”（OpenHandsetAlliance）来共同研发改良 Android 系统，这一联盟将支持谷歌发布的手机操作系统以及应用软件，Google 以 Apache 免费开源许可证的授权方式，发布了 Android 的源代码。

5.3 Tesorflow

来自谷歌 TensorFlow 是一个开源库，用于深度学习。它使用计算数据流图来表示复杂的神经网络结构。图中的节点表示数学计算，也称为 ops(操作)，而边表示在它们之间传输的数据张量（tensor）。此外，相关的梯度被存

储在计算图的每个节点上，在反向传播时，这些梯度被组合起来，得到相权值的梯度。张量是张量流，使用的多维数据数组来表示。

TensorFlow 2 的优势是它的简单和易用，它的更新包括即时执行、直观的高级 api 以及在任何平台上构建灵活的模型。

TensorFlow 具有以下优点：

1) **高度的灵活性**

可以使用 Tensorflow 进行图的构建，自己编写驱动计算程序的内部循环。TensorFlow 提供了丰富的工具来帮助开发者组装“子图”（常用于神经网络），当然用户也可以自己在 Tensorflow 基础上写自己的“上层库”。也不用担心性能不足。如果发现找不到想要的底层数据操作，我们也可以自己写底层代码来完成底层的操作。

2) **多语言支持**

Tensorflow 支持 c++，也有一个易用的 python 工具使得开发者构建 graph。开发者可以直接写 python/c++程序[23]。

5.4 TensorFlow Lite

TensorFlow Lite 是一套帮助开发人员在移动、嵌入式和物联网设备上运行 TensorFlow 模型的工具。它支持设备上的机器学习推理。

TensorFlow Lite 由两个主要组件组成：

1) TensorFlow Lite 解释器，它在许多不同的硬件类型上运行特别优化的模型，包括移动电话、嵌入式 Linux 设备和微控制器。
2) TensorFlow Lite 转换器，它将 TensorFlow 模型转换成一种高效的形式供解释器使用，可以使用优化方法来改进大小和性能。。

TensorFlow Lite 对于开发人员而言，在设备上执行机器学习可以帮助改善。运行 python 程序，将会生成一个名为“mnist_model_graph.pb”的文件，将在文件运用到 Android 项目中。

5.5 将训练好的模型移植到 Android 系统

1) 使用 Android Studio 来开发 Android 程序，通过以下步骤创建构建 Android 项目：

2) 生成 pb 文件：使用 TensorFlow Python API 构建并训练网络，最后将训练后的网络的拓扑结构和参数保存为 pb 文件。

构建 jar 包和 so 库：TensorFlow Android Inference Interface 提供了名为：org.tensorflow.contrib.android.TensorFlowInferenceInterface 的 Java 类，使得开发者可以在 Android 平台上加载 TensorFlow graphs，完成本地识别。

3) 将 pb 文件、jar 包以及 so 库引入 Android 工程中，并基于 TensorFlowInferenceInterface 类完成识别[28]。

生成的项目文件结构如下图：

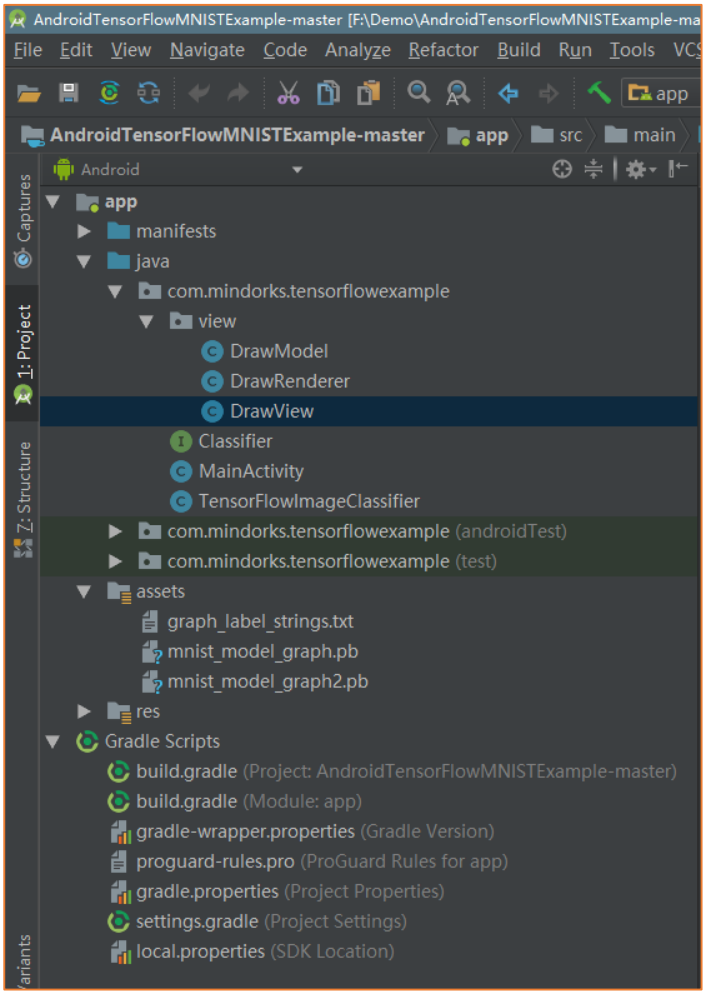

图 5-1 Android 项目结构

将 python 训练出的模型放在了 assets 文件夹目录下，这里训练了两份模型，所以有两个“.pb”文件，使用的时候只需要用一个即可。

5.6 软件安装

首先在名为“Application”的文件夹下打开名为“安卓人工智能识别”文件夹，其中有个文件名为 recognize_hand 后缀为.apk 的文件，如下图：

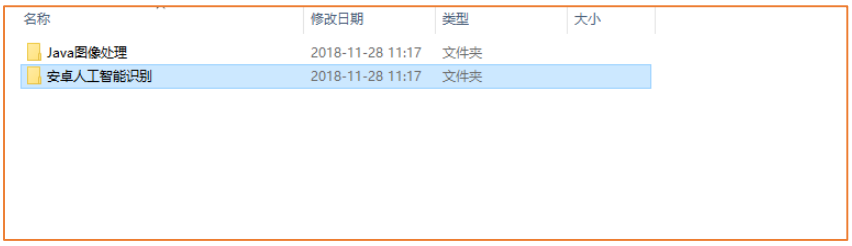

图 5-2 文件示意

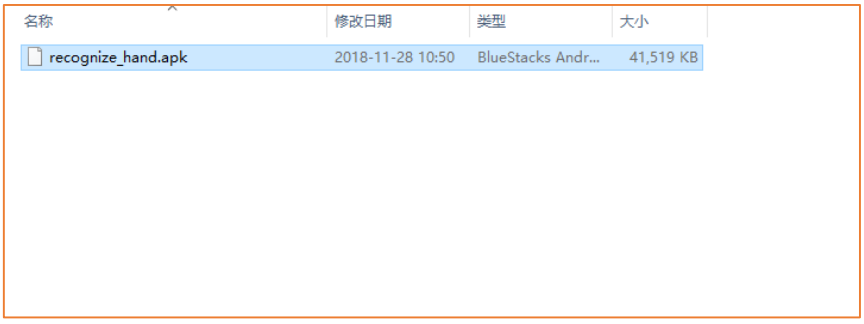

图 5-3 Android 程序文件

连接 Android 手机把该.apk 文件保存到手机存储卡中进行安装。安装成功后在手机桌面图标中有个 Android 图标的 app 出现，app 名称为 Recognize_HandWriting。（一定要在 Android 手机下才能运行，不能苹果手机，Android 的版本最好要高于或等于 7.0，Android6.0 运行可能会有错误）如下图：

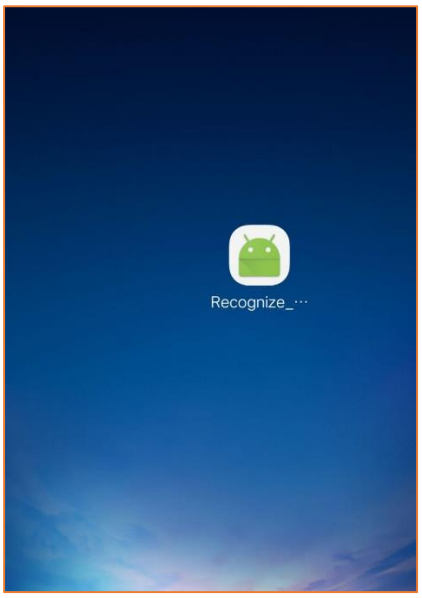

图 5-4 Android 桌面程序

进入 app 后出现如下界面：

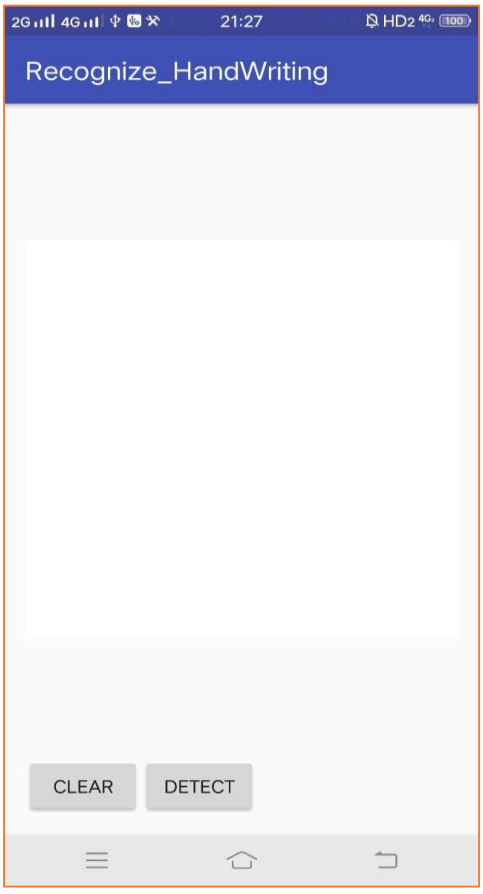

图 5-5 程序主界面

程序主界面的介绍如下图：

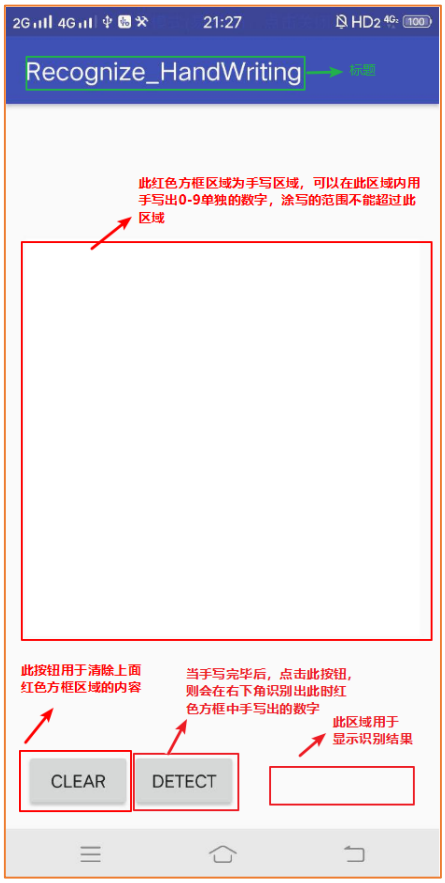

图 5-6 程序功能按钮介绍

以下为一些识别结果，该模型识别手写数字的正确率大致为 97.98%。

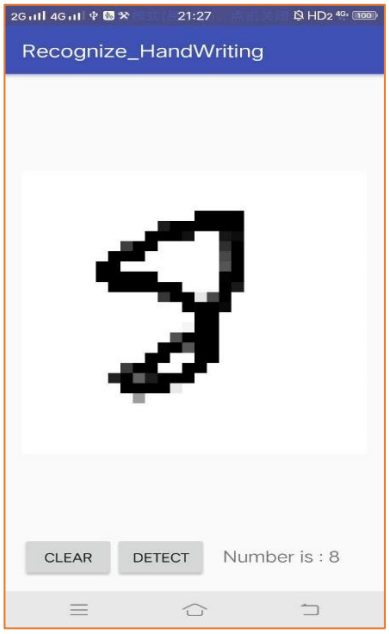

图 5-7 识别手写数字 8

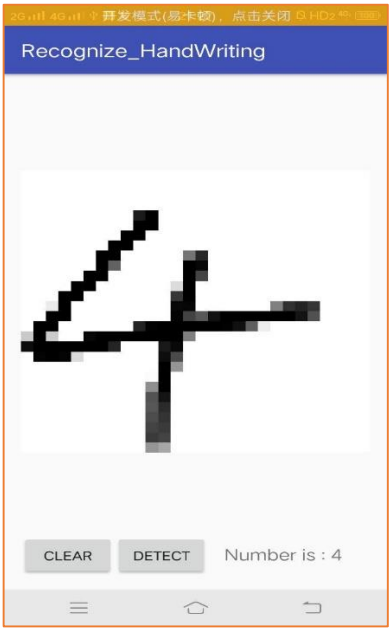

图 5-8 识别手写数字 4

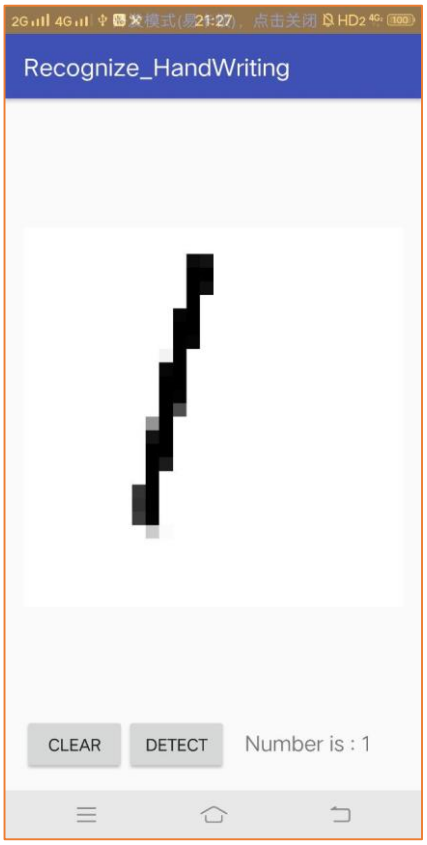

图 5-9 识别手写数字 1

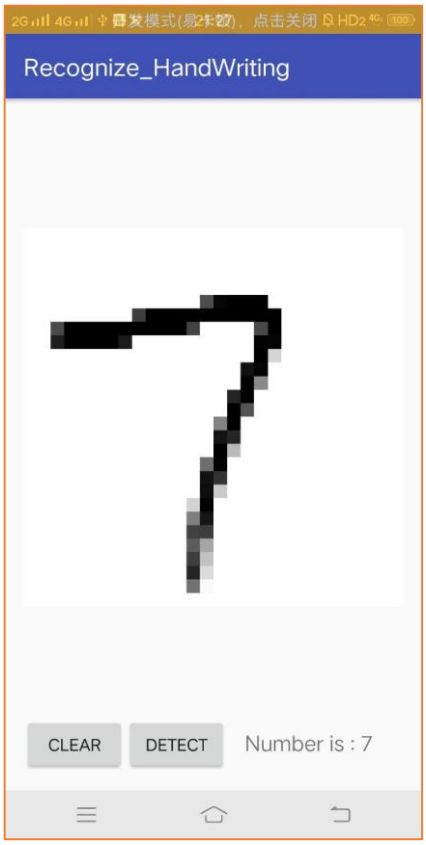

图 5-10 识别手写数字 7

第6章　总结与展望

自动化书写语言识别的研究可以追溯到几个世纪以前。今天打印体的文档可以在 OCR 软件的帮助下基本实现识别。正如我们在本文中所看到的，手写识别也取得了一些成功，尤其是孤立的手写数字和单词在这篇论文中，我们列举了在世界各国的科学家前辈正在的研究成果，希望这些项目和研究能够成功地保持全球人类多样性和丰富性，促使科学的进步[31]。

随着技术领域的不断创新，手写数字识别技术已经同深度学习、机器学习、人工智能等各种学科形成了交相辉映的关系，其本身也逐步发展成一门单独的全面综合的技术。但与此同时，手写技术同时也存在诸多不足，例如：实践中，不同的人手写的方式均有差异，要准确识别并非易事。另外手写数字自身所蕴含的信息量也较小，无法进而分析出内在更多含义。因此，其时间、空间的复杂度也是亟待解决的问题[32]。

首先，本文对所研究的手写数字识别系统的意义做了阐述，也对前人研究和发展历史进行了介绍。也用实践证明了神经网络的可行性。

其次，本文对各种算法和网络模型进行了介绍，也结合具体的代码示例进行演示，成功的实践说明了理论的正确性，也使用了通过 GPU 进行加速。

最后，结合了移动端的示例和机器学习框架 TensorFlow Lite 的成功的将模型移植到 Android 手机上。并取得了较高的识别率。

第7章　附件代码

代码一：显示手写数字

```
import tkinter
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg, NavigationToolbar2Tk
from matplotlib.backend_bases import key_press_handler
from matplotlib.figure import Figure
from sklearn.datasets import load_digits
# 从 sklearn 中获取数据
digits = load_digits()
root = tkinter.Tk()  # 创建 tkinter 的主窗口
root.title("显示 MNIST")
"""
    子图的编号 如下:
    # a = f.add_subplot(331)  添加子图:i 行 j 列第 k 个
    imageIndex 表示图片索引编号
"""
subplot = [431, 432, 433, 434, 435, 436, 437, 438, 439]
imageIndex = [0, 1, 2, 3, 4, 5, 6, 7, 8]
f = Figure(figsize=(7, 7), dpi=100)

"""
   # 在前面得到的子图上绘图
    a.matshow(digits.images[0])
"""
```

```
for i in range(0, 9):
    a = f.add_subplot(subplot[i])
    a.matshow(digits.images[imageIndex[i]])

"""
    4 行 3 列共 12 个格子，放在第 10 幅图的位置上
    由于 4310 不合法，所以单独放在写出
"""
a = f.add_subplot(4, 3, 10)
a.matshow(digits.images[9])
# 将绘制的图形显示到 tkinter:创建属于 root 的 canvas 画布,并将图 f
置于画布上
canvas = FigureCanvasTkAgg(f, master=root)
canvas.draw()  # 注意 show 方法已经过时了,这里改用 draw
canvas.get_tk_widget().pack(side=tkinter.TOP,  # 上对齐
                            fill=tkinter.BOTH,  # 填充方式
                            expand=tkinter.YES)  # 随窗口大小调整
而调整
# matplotlib 的导航工具栏显示上来(默认是不会显示它的)
toolbar = NavigationToolbar2Tk(canvas, root)
toolbar.update()
canvas.tkcanvas.pack(side=tkinter.TOP,  # get_tk_widget()得到
的就是_tkcanvas
                     fill=tkinter.BOTH,
```

```
                    expand=tkinter.YES)
def on_key_event(event):
    """
        键盘事件处理
    """
    print("你按了%s" % event.key)
    key_press_handler(event, canvas, toolbar)
# 绑定上面定义的键盘事件处理函数
canvas.mpl_connect('key_press_event', on_key_event)
def _quit():
    """点击退出按钮时调用这个函数"""
    root.quit()  # 结束主循环
    root.destroy()  # 销毁窗口



# 创建一个按钮,并把上面那个函数绑定过来
button = tkinter.Button(master=root, text="退出",
command=_quit)
# 按钮放在下边
button.pack(side=tkinter.BOTTOM)
# 主循环
root.mainloop()
```

代码二：神经网络实现

```
import numpy as np

# 双曲正切 （e^x - e^(-x)）/ (e^x + e^(-x))

def tanh(x):
    return np.tanh(x)

# 双曲正切的导函数

def tanh_deriv(x):
    return 1.0 - np.tanh(x) * np.tanh(x)

# logistic 函数

def logistic(x):
    return 1 / (1 + np.exp(-x))

# logistic 导函数

def logisticDeriv(x):
    return logistic(x) * (1 - logistic(x))
7.
    class Neural_Network:

        # 构造函数

        def __init__(self, layers, activation='tanh'):
            """
            :param layers: a list containing the number of
    units in each layer
                should be at least two values,
            :param activation: the activation function to be
    used,can be logistic or
                tanh,the default value is tanh,this is python
    special grammar
            """
            if activation == 'logistic':
                self.activation = logistic
                self.activation_deriv = logistic_deriv
            elif activation == 'tanh':
                self.activation = tanh
                self.activation_deriv = tanh_deriv

            # 权重，以一维数组形式存在

            self.weights = []

            # 以下部分用于初始化权
```

```
        for i in range(1, len(layers) - 1):  # 有多少层神经网络就要给多少层权重赋初值
            """
                假设神经网络为 2 层(第一列的神经元不是第一层，从第二列开始算第一层) 如下图...

                两个神经元一个连线为一个 weight，for 循环从 1 开始，也就是从第二层神经元为基准
                第一句为第二层与第一层的权重赋予初值“(layers[i - 1] + 1, layers[i] + 1)) - 1) * 0.25”
                其值在-0.25 与 0.25 之间。
                后一句是第二层与第三层之间权重的赋值
            """
            self.weights.append((2 * np.random.random((layers[i - 1] + 1, layers[i] + 1)) - 1) * 0.25)
            self.weights.append((2 * np.random.random((layers[i] + 1, layers[i + 1])) - 1) * 0.25)

    def fit(self, X, y, learning_rate=0.2, epochs=10000):  # 训练函数
        """
        :param X: Training Set 训练集，用二维矩阵来描述,每一行代表一个实例，含有一定数量的特征值
                每一列是一个维度的特征值，比如下图的 age 就是一个维度，第一行所有维度就组成一个实例
                如下图(是否买电脑预测图): ...
```

```
        :param y:class label 函数预测最终目标
        :param learning_rate:学习率，因为我们使用 gradient
descend 算法，每次都要乘以这个 rate
                    如果这个 rate 太大了，那么可能将会越过这个点。
        :param epochs:每更新一次神经网络都要用实例数据，如果每个
实例都使用的话，这样运算量
                    太过庞大，所以可以抽取抽样的方法去更新，而不是将
所有数据都更新。而这个 epoch 表示
                    要迭代多少次这个神经网络。（神经网络停止三个条件
满足之一即可 ）
        """
        # 将 X 参数转换成 2 维 实例是二维的
        X = np.atleast_2d(X)
        # 创建全 1 的矩阵，其行数列数与 X 矩阵的行数列数一样
        # X.shape[0]是行数  X.shape[1]是列数
        # 这里列数多 1，用于给 bias 即偏向赋值
        temp = np.ones([X.shape[0], X.shape[1] + 1])
        """
             取所有的行，从第一列到除最后一列所有值

        """
        temp[:, 0:-1] = X
        X = temp
        y = np.array(y)
        for k in range(epochs):  # 循环固定次数结束
            rand = np.random.randint(X.shape[0])  # 从 0 到行数
```

```
个数之间取一个数，即任取一行
            a = [X[rand]]
            for j in range(len(self.weights)):  # 权重数量有多
少，循环多少次，即更新多少次
                a.append(self.activation(np.dot(a[j],
self.weights[j])))  # 该属性值与权重值相乘
            """
                对于输出层的误差，Errj = Oj(1-Oj)(Tj-0j)
                最后一层输出值 *（1 - 最后一层输出值）*（等于标签
真实值 - 最后一层输出值
            """
            error = y[rand] - a[-1]
            deltas = [error * self.activation_deriv(a[-1])]

            # 反向传播开始
            for length in range(len(a) - 2, 0, -1):  # 从最后一
层循环倒退循环到第零层
                """
                    deltej = (1 - Oj)*sigma(errj * weightj)
                    [].T 表示对矩阵的转置
                """
                deltas.append(deltas[-
1].dot(self.weights[length].T) *
self.activation_deriv(a[length]))
            # 不要写在 for 循环里面
            deltas.reverse()  # 颠倒顺序
            # 对权重进行更新
            for i in range(len(self.weights)):
```

```
                layer = np.atleast_2d(a[i])
                # delta表示当前层的误差，deltas表示每一层的误差
                delta = np.atleast_2d(deltas[i])
                """
                    new_weight = old_weight + learning_rate *
error * delta
                """
                self.weights[i] += learning_rate *
layer.T.dot(delta)

    def predict(self, x):  # 预测函数
        x = np.array(x)
        temp = np.ones(x.shape[0] + 1)
        temp[0:-1] = x
        a = temp
        for length in range(0, len(self.weights)):
            a = self.activation(np.dot(a,
self.weights[length]))
        return a

nn = NeuralNetwork([2, 2, 1], 'tanh')
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])
nn.fit(X, y)
for i in [[0, 0], [0, 1], [1, 0], [1, 1]]:
    print(i, nn.predict(i))
```

代码三：预测手写数字

```
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelBinarizer
from second.Demo.NeuralNetwork import NeuralNetwork
from sklearn.model_selection import train_test_split
from sklearn.datasets import *
import numpy as np

# 加载数据

digits = load_digits()
X = digits.data
y = digits.target
```

```
X -= X.min()
X /= X.max()

nn = NeuralNetwork([64, 100, 10], 'logistic')
X_train, X_test, y_train, y_test = train_test_split(X, y)
labels_train = LabelBinarizer().fit_transform(y_train)
lable_test = LabelBinarizer().fit_transform(y_test)
print("starting fiting")

nn.fit(X_train, labels_train, epochs=3000)
predictions = []
for i in range(X_test.shape[0]):
    o = nn.predict(X_test[i])
    predictions.append(np.argmax(o))

print(confusion_matrix(y_test, predictions))
```

代码四：Android 软件手绘制区域主要代码：

```
package com.mindorks.tensorflowexample.view;


import android.content.Context;
import android.graphics.*;
import android.util.AttributeSet;
import android.view.View;



public class DrawView extends View {
    private Paint mPaint = new Paint();
    private DrawModel mModel;
    // 28x28 pixel Bitmap
    private Bitmap mOffscreenBitmap;
    private Canvas mOffscreenCanvas;

    private Matrix mMatrix = new Matrix();
    private Matrix mInvMatrix = new Matrix();
    private int mDrawnLineSize = 0;
    private boolean mSetuped = false;

    private float mTmpPoints[] = new float[2];

    public DrawView(Context context, AttributeSet attrs) {
```

```
    super(context, attrs);
}

public void setModel(DrawModel model) {
    this.mModel = model;
}

public void reset() {
    mDrawnLineSize = 0;
    if (mOffscreenBitmap != null) {
        mPaint.setColor(Color.WHITE);
        int width = mOffscreenBitmap.getWidth();
        int height = mOffscreenBitmap.getHeight();
        mOffscreenCanvas.drawRect(new Rect(0, 0, width, height), mPaint);
    }
}
private void setup() {
    mSetuped = true;

    // View size
    float width = getWidth();
    float height = getHeight();
    // Model (bitmap) size
    float modelWidth = mModel.getWidth();
    float modelHeight = mModel.getHeight();

    float scaleW = width / modelWidth;
    float scaleH = height / modelHeight;
    float scale = scaleW;
    if (scale > scaleH) {
        scale = scaleH;
    }

    float newCx = modelWidth * scale / 2;
    float newCy = modelHeight * scale / 2;
    float dx = width / 2 - newCx;
    float dy = height / 2 - newCy;

    mMatrix.setScale(scale, scale);
    mMatrix.postTranslate(dx, dy);
    mMatrix.invert(mInvMatrix);
    mSetuped = true;
}
```

```
@Override
public void onDraw(Canvas canvas) {
    if (mModel == null) {
        return;
    }
    if (!mSetuped) {
        setup();
    }
    if (mOffscreenBitmap == null) {
        return;
    }

    int startIndex = mDrawnLineSize - 1;
    if (startIndex < 0) {
        startIndex = 0;
    }

    DrawRenderer.renderModel(mOffscreenCanvas, mModel, mPaint, startIndex);
    canvas.drawBitmap(mOffscreenBitmap, mMatrix, mPaint);

    mDrawnLineSize = mModel.getLineSize();
}

/**
 * Convert screen position to local pos (pos in bitmap)
 */
public void calcPos(float x, float y, PointF out) {
    mTmpPoints[0] = x;
    mTmpPoints[1] = y;
    mInvMatrix.mapPoints(mTmpPoints);
    out.x = mTmpPoints[0];
    out.y = mTmpPoints[1];
}

public void onResume() {
    createBitmap();
}

public void onPause() {
    releaseBitmap();
}
```

```
    private void createBitmap() {
        if (mOffscreenBitmap != null) {
            mOffscreenBitmap.recycle();
        }
        mOffscreenBitmap = Bitmap.createBitmap(mModel.getWidth(), mModel.getHeight(),
Bitmap.Config.ARGB_8888);
        mOffscreenCanvas = new Canvas(mOffscreenBitmap);
        reset();
    }

    private void releaseBitmap() {
        if (mOffscreenBitmap != null) {
            mOffscreenBitmap.recycle();
            mOffscreenBitmap = null;
            mOffscreenCanvas = null;
        }
        reset();
    }

    /**
     * Get 28x28 pixel data for tensorflow input.
     */
    public float[] getPixelData() {
        if (mOffscreenBitmap == null) {
            return null;
        }

        int width = mOffscreenBitmap.getWidth();
        int height = mOffscreenBitmap.getHeight();

        // Get 28x28 pixel data from bitmap
        int[] pixels = new int[width * height];
        mOffscreenBitmap.getPixels(pixels, 0, width, 0, 0, width, height);

        float[] retPixels = new float[pixels.length];
        for (int i = 0; i < pixels.length; ++i) {
            // Set 0 for white and 255 for black pixel
            int pix = pixels[i];
            int b = pix & 0xff;
            retPixels[i] = 0xff - b;
        }
        return retPixels;
```

```
    }
}
```

代码五：调用模型识别过程代码：

```
package com.mindorks.tensorflowexample;

import android.graphics.PointF;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;
import android.widget.TextView;

import com.mindorks.tensorflowexample.view.DrawModel;
import com.mindorks.tensorflowexample.view.DrawView;
import java.util.List;
import java.util.concurrent.Executor;
import java.util.concurrent.Executors;
public class MainActivity extends AppCompatActivity implements View.OnTouchListener {
    private static final String TAG = "MainActivity";
    private static final int PIXEL_WIDTH = 28;
    private TextView mResultText;
    private float mLastX;
    private float mLastY;

    private DrawModel mModel;
    private DrawView mDrawView;
    private View detectButton;
    private PointF mTmpPoint = new PointF();
    private static final int INPUT_SIZE = 28;
    private static final String INPUT_NAME = "input";
    private static final String OUTPUT_NAME = "output";
    private static final String MODEL_FILE = "file:///android_asset/mnist_model_graph.pb";
    private static final String LABEL_FILE =
            "file:///android_asset/graph_label_strings.txt";
    private Classifier classifier;
    private Executor executor = Executors.newSingleThreadExecutor();
    @SuppressWarnings("SuspiciousNameCombination")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
    mModel = new DrawModel(PIXEL_WIDTH, PIXEL_WIDTH);
    mDrawView = (DrawView) findViewById(R.id.view_draw);
    mDrawView.setModel(mModel);
    mDrawView.setOnTouchListener(this);
    detectButton = findViewById(R.id.buttonDetect);
    detectButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            onDetectClicked();
        }
    });

    View clearButton = findViewById(R.id.buttonClear);
    clearButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            onClearClicked();
        }
    });

    mResultText = (TextView) findViewById(R.id.textResult);
    initTensorFlowAndLoadModel();
}

private void initTensorFlowAndLoadModel() {
    executor.execute(new Runnable() {
        @Override
        public void run() {
            try {
                classifier = TensorFlowImageClassifier.create(
                        getAssets(),
                        MODEL_FILE,
                        LABEL_FILE,
                        INPUT_SIZE,
                        INPUT_NAME,
                        OUTPUT_NAME);
                makeButtonVisible();
                Log.d(TAG, "Load Success");
            } catch (final Exception e) {
                throw new RuntimeException("Error initializing TensorFlow!", e);
            }
        }
    });
```

```
}

private void makeButtonVisible() {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            detectButton.setVisibility(View.VISIBLE);
        }
    });
}

@Override
protected void onResume() {
    mDrawView.onResume();
    super.onResume();
}

@Override
protected void onPause() {
    mDrawView.onPause();
    super.onPause();
}

@Override
public boolean onTouch(View v, MotionEvent event) {
    int action = event.getAction() & MotionEvent.ACTION_MASK;

    if (action == MotionEvent.ACTION_DOWN) {
        processTouchDown(event);
        return true;

    } else if (action == MotionEvent.ACTION_MOVE) {
        processTouchMove(event);
        return true;

    } else if (action == MotionEvent.ACTION_UP) {
        processTouchUp();
        return true;
    }
    return false;
}

private void processTouchDown(MotionEvent event) {
```

```
        mLastX = event.getX();
        mLastY = event.getY();
        mDrawView.calcPos(mLastX, mLastY, mTmpPoint);
        float lastConvX = mTmpPoint.x;
        float lastConvY = mTmpPoint.y;
        mModel.startLine(lastConvX, lastConvY);
    }

    private void processTouchMove(MotionEvent event) {
        float x = event.getX();
        float y = event.getY();

        mDrawView.calcPos(x, y, mTmpPoint);
        float newConvX = mTmpPoint.x;
        float newConvY = mTmpPoint.y;
        mModel.addLineElem(newConvX, newConvY);

        mLastX = x;
        mLastY = y;
        mDrawView.invalidate();
    }

    private void processTouchUp() {
        mModel.endLine();
    }

    private void onDetectClicked() {
        float pixels[] = mDrawView.getPixelData();
        final List<Classifier.Recognition> results = classifier.recognizeImage(pixels);

        if (results.size() > 0) {
            String value = " Number is : " +results.get(0).getTitle();
            mResultText.setText(value);
        }

    }
    private void onClearClicked() {
        mModel.clear();
        mDrawView.reset();
        mDrawView.invalidate();

        mResultText.setText("");
    }
```

```
    @Override
    protected void onDestroy() {
        super.onDestroy();
        executor.execute(new Runnable() {
            @Override
            public void run() {
                classifier.close();
            }
        });
    }
}
```

代码六：识别手写数字代码实现

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf

# load test data and reading tools,load test handwritten data,the
directory  MNIST_data is data container that load from the network.
import tensorflow.examples.tutorials.mnist.input_data as input_data

mnist = input_data.read_data_sets("MNIST_data", one_hot=True)

# create interactive session
sess = tf.InteractiveSession()

# create two placeholders,the data type is float.the shape of x
placeholder is [None, 784].it is variable that store image data.
# how many images is not important.the dimension of images is 784。
# where is from? Because the image hava 28*28 pixel,
# the shape of y_ is similar to x,but it has only 10 demension,
# the output has only 0-9,so it has only 10 structures
x = tf.placeholder("float", shape=[None, 784])
y_ = tf.placeholder("float", shape=[None, 10])


# define weight variable by function form.the initial value is from
Gaussion Distribution
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)
```

```
# define bias value by function form, the initial value is 0.1 at
all, the shape of bias depend on shape
def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)


# define convolutional function,the x is input data,W is
weight,strides is step size or movement speed,it contain length and
width and direction.
# padding stand for polishing data,it has two method,the one is
SAME ,which fill zero in image around,the anthor one is VALID .
# it will lose data when convolution from right or bottom
def conv2d(x, w):
    return tf.nn.conv2d(x, w, strides=[1, 1, 1, 1], padding='SAME')


# this step define pooling operation,in convolution operation,it is
a bottom sampling operation,it is the normal method that cluster
classification.
# pooling operation contain max pooling and average pooling,this is
2*2 pooling,pooling operation is not overlap.
# the origin of function: def max_pool(value, ksize, strides,
padding, data_format="NHWC", name=None)。对 ksize 和 strides
# 定义的理解要基于 data_format 进行。默认 NHWC，表示 4 维数据，
[batch,height,width,channels]. 下面函数中的 ksize，
# in strides, dealing one image, corresponding dealing data pipe.
# pooling area is also reflect sampling speed
def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2,
1], padding='SAME')


# define the first layer convolutional kernel.Shape is
correspongding convolutional kernel filter
# the structure of filter: [filter_height, filter_width,
in_channels, out_channels]。Convolution kernel length nad width is
5,
# input pipe is 1,output pipe is 32,it is stand for 32 convolution
kernel involve convolution
W_conv1 = weight_variable([5, 5, 1, 32])
```

```
# the dimension of bias is 32
b_conv1 = bias_variable([32])

# the input tensor reshaping to a 28*28 image,because x is
[None,784] when inputting is [-1,28,28,1].
#,  28,28,1 multiple result = 784, so the value of -1 is stand for
noe
# the batch is quantity of input data
x_image = tf.reshape(x, [-1, 28, 28, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
# the activation function result sampling pooling operation。
h_pool1 = max_pool_2x2(h_conv1)

# the second convolution, the size of convolutional kernel is 5*5,
it has 32 input pipe,64 output pipe,the convolutional unit havs 64
 W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

# 第二层卷积：激活和池化（类似第一层卷积操作的激活和池化）
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# the size of image decrese 7 * 7
# vector multiple weight matrix,and add bias then call ReLU
activation function
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder("float")
# this is ReLU result,is base on tensor value remain or dispose
data in relative data.it is prevent data from overfitting .
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

# in the end, add a softmax layer, as like one layer that softmax
regression.softmax is more select and classification function.
# the classification output result is 10.
```

```
y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

# 实际值 y_与预测值 y_conv 的自然对数求乘积，在对应的维度上上求和，
该值作为梯度下降法的输入
cross_entropy = -tf.reduce_sum(y_ * tf.log(y_conv))

train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)

# 首先分别在训练值 y_conv 以及实际标签值 y_的第一个轴向取最大值，比
较是否相等
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_,
1))

accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))

sess.run(tf.global_variables_initializer())
for i in range(20000):
# 从 mnist 的 train 数据集中取出 50 批数据，返回的 batch 其实是一个列
表，元素 0 表示图像数据，元素 1 表示标签值
    batch = mnist.train.next_batch(50)
    if i % 100 == 0:
train_accuracy = accuracy.eval(feed_dict={x: batch[0], y_:
batch[1], keep_prob: 1.0})
        print("step %d, training accuracy %g" % (i,
train_accuracy))
train_step.run(feed_dict={x: batch[0], y_: batch[1],
keep_prob:0.5})

# 对测试图片和测试标签值以及给定的 keep_prob 进行 feed 操作，进行计算
求出识别率。就相当于前面训练好的 W 和 bias 作为已知参数。
print("cf accuracy %g" % accuracy.eval(feed_dict={x:
mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))

# 经过数据集合训练之后，将训练后的网络的拓扑结构和参数保存为“.pb”
文件。然后运用到 Android 程序中，保存拓扑结构的代码如下：
# Create new graph for exporting
g = tf.Graph()
with g.as_default():
    x_2 = tf.placeholder("float", shape=[None, 784], name="input")

    WC1 = tf.constant(WC1, name="WC1")
    BC1 = tf.constant(BC1, name="BC1")
```

```
    x_image = tf.reshape(x_2, [-1, 28, 28, 1])
    CONV1 = conv2d(x_image, WC1, BC1)
    MAXPOOL1 = maxpool2d(CONV1, k=2)

    WC2 = tf.constant(WC2, name="WC2")
    BC2 = tf.constant(BC2, name="BC2")
    CONV2 = conv2d(MAXPOOL1, WC2, BC2)
    MAXPOOL2 = maxpool2d(CONV2, k=2)

    WD1 = tf.constant(WD1, name="WD1")
    BD1 = tf.constant(BD1, name="BD1")

    FC1 = tf.reshape(MAXPOOL2, [-1, WD1.get_shape().as_list()[0]])
    FC1 = tf.add(tf.matmul(FC1, WD1), BD1)
    FC1 = tf.nn.relu(FC1)

    W_OUT = tf.constant(W_OUT, name="W_OUT")
    B_OUT = tf.constant(B_OUT, name="B_OUT")

    # skipped dropout for exported graph as there
    # is no need for already calculated weights

    OUTPUT = tf.nn.softmax(tf.matmul(FC1, W_OUT) + B_OUT,
name="output")
    sess = tf.Session()
    init = tf.initialize_all_variables()
    sess.run(init)
    graph_def = g.as_graph_def()
    tf.train.write_graph(graph_def, EXPORT_DIR,
'mnist_model_graph.pb', as_text=False)
```

参考文献

[1] Chengde Zhang and Xiangnian Huang, "Off-line handwritten digit

recognition based on improved BP artificial neural network," 2008 IEEE International Conference on Service Operations and Logistics, and Informatics, Beijing, 2008, pp. 626-629.

[2] Y. Hou and H. Zhao, "Handwritten digit recognition based on depth neural network," 2017 International Conference on Intelligent Informatics and Biomedical Sciences (ICIIBMS), Okinawa, 2017, pp. 35-38.

[3] 人工神经网络原理及应用/朱大奇，史慧编著. —北京：科学出版社，2006 ISBN 7-03-016570-5

[4] 韩晓雪. 基于人工神经网络和GPU加速的手写数字识别并行算法[D].大连理工大学,2009.

[5] Ashiquzzaman and A. K. Tushar, "Handwritten Arabic numeral recognition using deep learning neural networks," 2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR), Dhaka, 2017, pp. 1-4.

[6] Amin, A. (1998). Off-line Arabic character recognition. Pattern Recognition, 31(5), 517－530. doi:10.1016/s0031-3203(97)00084-8

[7] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[8] 孙泽禹. 基于tensorflow的手写数字识别[D].长春理工大学,2019.

[9] 李彦冬， 郝宗波， 雷航. 卷积神经网络研究综述[J]， Journal of Computer Applications ， 2016 ， 36(9):2508-2515,2565,doi:10.11772/j.issn.1001-9081.2016.09.2508

[10] X. Xu, H. Ge and S. Li, "An improvement on recurrent neural network by combining convolution neural network and a simple initialization of the weights," 2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS),

Chongqing, 2016, pp. 150-154.

[11] Bottou, L. (2010). Large-Scale Machine Learning with Stochastic Gradient Descent. Proceedings of COMPSTAT'2010, 177－186. doi:10.1007/978-3-7908-2604-3_16

[12] THE MNIST DATABASE of handwritten digits http://yann.lecun.com/exdb/mnist/

[13] TkDocs https://tkdocs.com/tutorial/intro.html

[14] M. M. Lau and K. Hann Lim, "Review of Adaptive Activation Function in Deep Neural Network," 2018 IEEE-EMBS Conference on Biomedical Engineering and Sciences (IECBES), Sarawak, Malaysia, 2018, pp. 686-690.

[15] D. Stursa and P. Dolezel, "Comparison of ReLU and linear saturated activation functions in neural network for universal approximation," 2019 22nd International Conference on Process Control (PC19), Strbske Pleso, Slovakia, 2019, pp. 146-151.

[16] 蒋文斌,彭晶,叶阁焰.深度学习自适应学习率算法研究[J].华中科技大学学报(自然科学版),2019,47(05):79-83.

[17] Matplotlib New matplotlib English official website. https://www.matplotlib.org.cn/en/

[18] Ari Ernesto Ortiz Castellanos. Mobile Object Detection Using 2D and 3D Basic Geometric Figures in Colour and Grayscale[C]. Asia Pacific Institute of Science and Engineering.Proceedings of 2019 3rd International Conference on Machine Vision and Information Technology (CMVIT 2019).Asia Pacific Institute of Science and Engineering:成都夏洛克教育咨询有限公司,2019:347-354.

[19] Campoverde A., Barros G. (2020) Detection and Classification of Urban Actors Through TensorFlow with an Android Device. In: Fosenca C E., Rodríguez Morales G., Orellana Cordero M., Botto-

Tobar M., Crespo Martínez E., Patiño León A. (eds) Information and Communication Technologies of Ecuador (TIC.EC). TICEC 2019. Advances in Intelligent Systems and Computing, vol 1099. Springer, Cham doi:10.1007/978-3-030-35740-5

[20] Pattanayak, S. (2017). Introduction to Deep-Learning Concepts and TensorFlow. Pro Deep Learning with TensorFlow, 89–152. doi:10.1007/978-1-4842-3096-1_2

[21] What is Android——The platform changing what mobile can do. https://www.android.com/what-is-android/

[22] Google: Android TensorFlow lite (2019). https://www.tensorflow.org/lite/models/object_detection/overview

[23] TensorFlow is an end-to-end open source platform for machine learning https://www.tensorflow.org/guide

[24] M. Han, J. Chen, L. Li and Y. Chang, "Visual hand gesture recognition with convolution neural network," 2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Shanghai, 2016, pp. 287-291.

[25] G. Yue and L. Lu, "Face Recognition Based on Histogram Equalization and Convolution Neural Network," 2018 10th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), Hangzhou, 2018, pp. 336-339.

[26] What Do We Understand About Convolutional Networks?

Isma Hadji and Richard P. WildesDepartment of Electrical Engineering and Computer Science, York University, Toronto, Ontario Canada, arXiv:1803.08834v1 [cs.CV] 23 Mar 2018

[27] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), Antalya, 2017, pp. 1-6.

[28] Intuitively Understanding Convolutions for Deep Learning https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1

[29] NVIDIA https://developer.nvidia.com/cuda-zone

[30] T. Fukagai et al., "Speed-Up of Object Detection Neural Network with GPU," 2018 25th IEEE International Conference on Image Processing (ICIP), Athens, 2018, pp. 301-305.

[31] Plamondon, R., & Srihari, S. N. (2000). Online and off-line handwriting recognition: a comprehensive survey. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(1), 63–84. doi:10.1109/34.824821

[32] W. T. Illingworth, "Beginner's guide to neural networks," in IEEE Aerospace and Electronic Systems Magazine, vol. 4, no. 9, pp. 44-49, Sept. 1989, doi: 10.1109/62.35668.

谢 辞

本文的研究工作是在我的导师黄晓红教授的悉心指导和严格要求下完成的。黄老师在学习方法、工作方法和研究思路等方面给予了许多有益的启迪；同时，她对我的研究工作提出了宝贵的建议和意见，使我在研究工作中不断取得新的进展。黄老师深厚的专业知识、严谨的治学精神和求实创新的工作作风深深的影响着我。在此，谨向黄老师致以我最崇高的敬意和真挚的感谢！

感谢我的家人和朋友对我生活上的关心，学习和工作的支持，这些使得我能够安心的完成我的研究工作。

最后，对在我的学习和成长道路上给予帮助的所有老师和朋友们表示深深地感谢，对评阅该论文的所有专家表示最崇高的敬意和真挚的感谢！