



上海大学

毕业设计（论文）

装
订
线

题目：基于证据学习的深度学习图像分类算法的可视化实现

学 院：中欧工程技术学院

专 业：信息工程

学 号：18124722

学生姓名：钟乐幸

指导教师：杜水淼

起讫日期：2022.3.1-2022.6.8

目录

摘要	III
ABSTRACT.....	IV
第 1 章 绪论	1
§1.1 深度学习的发展背景	1
§1.2 图像分类的发展背景	1
§1.3 证据深度学习的发展背景	2
§1.4 可视化深度学习算法的应用前景	3
§1.5 本文研究内容及目标	3
§1.5.1 研究内容	3
§1.5.2 研究目标	3
§1.6 本文组织结构	4
第 2 章 深度学习与证据深度学习	5
§2.1 传统深度学习概述	5
§2.1.1 传统深度学习简介	5
§2.1.2 传统深度学习运行过程	5
§2.1.3 传统深度学习常用术语	6
§2.1.4 经典的卷积网络模型	9
§2.2 传统深度学习的问题	10
§2.2.1 过于自信的预测	11
§2.2.2 严格做出一个预测	11
§2.3 证据深度学习	12
§2.3.1 证据深度学习原理	12
§2.3.2 证据深度学习中的损失函数	13
§2.4 性能比较	15
§2.4.1 Mnist 数据集	16
§2.4.2 模型网络结构	16
§2.4.3 模型对比	18
第 3 章 算法的可视化实现	20
§3.1 可视化技术简介	20
§3.2 数学可视化工具 Manim.....	20
§3.2.1 Manim 简介	20
§3.2.2 Manim 库架构	21
§3.2.3 Manim 库常用组件	23
§3.3 基于 Manim 库实现证据深度学习的可视化	25
§3.3.1 可视化动画的主体框架	25
§3.3.2 封装函数说明	25

§3.3.3 可视化实现深度学习运行过程	28
§3.3.4 可视化实现传统深度学习存在的问题	29
§3.3.5 可视化实现证据深度学习运行过程	30
§3.3.6 可视化实现比较两种模型的性能	32
第 4 章 证据深度学习网络应用程序	33
§4.1 需求分析	33
§4.1.1 程序的核心功能	33
§4.1.2 前后端分离架构	34
§4.2 使用 streamlit 开发网络应用程序	34
§4.2.1 Streamlit 概述	34
§4.2.2 Streamlit 功能组件	35
§4.2.3 使用 streamlit 搭建证据深度学习网络应用程序	37
§4.3 使用 fastapi 开发简易 API	39
§4.3.1 Fastapi 概述	39
§4.3.2 Fastapi 开发简易接口	40
§4.3.3 Fastapi 开发证据深度学习预测 API	41
§4.4 使用 Docker 进行程序部署	42
§4.4.1 Docker 简介	42
§4.4.2 Docker 中的容器和镜像	43
§4.4.3 Dockerfile 创建容器镜像	44
§4.4.4 使用 Docker 部署前后端程序	45
§4.4.5 使用 docker-compose 一键部署应用程序	46
第 5 章 总结与展望	48
§5.1 本文总结	48
§5.1.1 本文的主要工作	48
§5.1.2 本文的主要创新点	49
§5.2 展望	49
§5.2.1 本文的应用场景与市场前景	49
§5.2.2 本文的不足及改进措施	50
致谢	51
参考文献	52

基于证据学习的深度学习图像分类算法的可视化实现

摘要

2016 年,随着人工智能 AlphaGo 大胜李世石,机器学习和深度学习迎来一阵关注热潮。近年来,随着人工智能、机器学习等相关技术和算法的发展,机器学习在我们的日常生活中正扮演着越来越重要的角色,正逐渐深刻地影响着我们的生活。其中,图像分类作为机器学习中一大重要应用,深度学习凭借着其复杂的神经网络和卷积层的加持,正日益成为机器学习图像分类领域中的“香饽饽”。

然而,深度学习图像分类算法也有它的一些短板和缺陷。例如,一个训练好的模型对某一分类的预测准确率可能并不高,但仍然做出该类的预测结果。这样的“自负”表现往往会导致结果的误判,也体现了模型的训练不够到位。本文将介绍一种利用证据理论原理实现的证据深度学习图像分类算法,通过计算每次的预测结果的不确定度来衡量模型的预测表现。

此外,机器学习渗透着各个领域,各个专业,几乎成为相关专业大学生的必备的知识储备。然而,在高校的机器学习或深度学习课程中,抽象的概念在静态的课件中往往不能够让学生直观地理解知识内容。这样的教学往往需要过高的基础知识,以及良好的理解能力。这样的高要求可能会让很多学生望而却步,从而错失了一次进入这个奇妙世界的机会。因此本文将以介绍证据深度学习算法为契机,使用数学可视化工具 Manim 来演示该算法的运行过程,通过动态的、有趣的动画来让学生们对深度学习和证据深度学习有一个更为直观的认识。

最后,本文还通过介绍一个简易网络应用程序搭建工具 streamlit,来搭建一个深度学习与证据深度学习的网络应用程序。使用者可以通过手写数字或上传图片等方式来直观感受两种算法的差别和魅力。

关键词: 深度学习, 证据理论, 图像分类, 可视化, 网络应用程序

Visualization of evidence-based image classification algorithms in a deep learning framework

ABSTRACT

In 2016, as the artificial intelligence AlphaGo beat Lee Sedol, machine learning and deep learning ushered in a wave of attention. In recent years, with the development of artificial intelligence, machine learning and other related technologies and algorithms, machine learning is playing an increasingly important role in our daily life and is gradually and profoundly affecting our lives. And with the blessing of its complex neural network and convolutional layers, deep learning is increasingly becoming a "sweet pastry" in the field of machine learning image classification, which is one of the most important fields in machine learning industry.

However, deep learning image classification algorithms also have its shortcomings and flaws. For instance, a trained model may not predict a class with high accuracy, but still make predictions for that class. Such "conceited" performance often leads to misjudgment of the results, and also reflects the fact that the training of the model is not adequate. This paper will introduce an evidential deep learning image classification algorithm which base on the principle of evidence theory. It measures the prediction performance of the model by calculating the uncertainty of each prediction.

In addition, machine learning permeates various fields and majors, and has almost become a necessary knowledge reserve for college students in related majors. However, in the machine learning or deep learning courses in colleges and universities, the abstract concepts in the static courseware often cannot allow students to intuitively understand the knowledge content. Such teaching often requires a high level of basic knowledge, as well as good comprehension skills. Such high demands may discourage many students, thus missing an opportunity to enter this wonderful world. Therefore, this article will use the mathematical visualization tool, Manim, to demonstrate the running process of the algorithm. Through dynamic and interesting animations, students will have a more intuitive understanding of deep learning and evidential deep learning algorithm knowledge.

Finally, this paper also introduces a simple network application building tool,

streamlit, to build a deep learning and evidential deep learning web application. Users can intuitively feel the difference and charm of the two algorithms by handwriting numbers or uploading pictures.

Keywords: Neural networks, evidence theory, Image Classification, Visualization, Web application

第1章 绪论

本章主要介绍了深度学习和证据深度学习的发展背景，以及可视化深度学习相关算法的意义与应用前景，提出用可视化的方式来比较两种机器学习方法的差异，进而让学习者对证据深度学习有一个更直观，更深刻的认识，最后提出本文的研究内容及目标。

§ 1.1 深度学习的发展背景

1943 年，神经科学家麦卡洛克(W.S.McCulloch)和数学家皮兹(W.Pitts)首次建立了神经网络和数学模型^[1]，该模型按照生物神经元的结构和工作原理构造出了所谓的模拟大脑，开启了人工神经网络的大门。

1958 年，计算机科学家罗森布拉特提出了两层神经网络，其称之为感知机(Perceptrons)，第一次将 MCP 模型用于机器学习分类任务^[2]。1986 年，由 Geoffrey Hinton 发明了适用多层感知机的反向传播算法(Backpropagation)，并提出了 sigmoid 函数进行非线性映射^[3]，引发了神经网络的一次热潮。1998 年，LeCun 首次提出了卷积神经网络(Convolutional Neural Networks, CNN)^[4]，并通过 LeNet 介绍了卷积网络的基本结构：卷积层+池化层+全连接层。2012 年，由 Hinton 课题组提出的 AlexNet 网络^[5]在 ImageNet Large-Scale Visual Recognition Challenge 中以绝对优势打败第二名的 SVM 一举夺冠。

经过近十年的发展，随着 GoogleNet, ResNet 等 CNN 网络相继提出，深度学习也凭借卷积网络与高效的性能表现，逐渐在机器学习领域中占据越来越重要的比重。

§ 1.2 图像分类的发展背景

分类任务(Classification task)是机器学习领域中的一大重要应用，其中图像的分类与识别则是分类任务中的重要任务，在传统的机器学习中，图像分类任务需要经过对图像的特征提取，特征筛选，最后将筛选后的特征输入合适的分类器完成基于特征的图像分类任务。

毫无疑问，传统机器学习的图像分类方法是繁琐的、复杂的。直到 1998 年，LeCun 提出的 LeNet 网络^[4]让图像分类的发展进入了全新的历史篇章。LeNet 中的卷积层(Convolutional Layers)使得传统图像分类任务的前期处理变成过去式。通过卷积层、池化层和全连接层的配合，不同的卷积核使得神经网络能够从不同方向、不同维度对图像的特征进行提取。例如将一部分卷积用于提取图像的边缘特征，一部分卷积用于提取特定数据集的特定分布特征等等。

2012 年, 随着 Hinton 课题组提出的 AlexNet^[5]在 ImageNet Large-Scale Visual Recognition Challenge 中夺冠, 卷积网络在图像分类任务中占据着越来越重要的地位。随后随着 VGGNet, GoogleNet 等网络的相继提出, 其在卷积网络层与成本函数、梯度下降等算法的不断更新与发展, 深度学习图像分类算法得到了进一步的提升。

然而, 尽管卷积网络和深度学习等在机器学习图像分类算法中熠熠生辉, 近年来, 人们也逐渐发现了这些传统深度学习算法在分类任务中存在的问题, 如过于自负的预测表现, 以及在面对不相关样本中表现出的严格预测问题。如何解决这些问题成为了优化深度学习在分类任务中的一大重要方向。这些问题如若能够得以解决, 深度学习在图像分类任务中将会得到进一步的提升和发展。

§ 1.3 证据深度学习的发展背景

所谓证据, 是指该算法引入证据理论的思想。证据理论又称 Dempster-Shafer 理论或 DS 理论^[6], 证据理论基于两种思想: (1)从相关问题的主观概率中获得一个问题的置信度(degree of belief), (2)使用 Dempster 规则对基于独立证据因素的问题的置信度进行融合。该理论是对贝叶斯主观概率理论的一种推广, 置信度用信任函数(belief function)表示而不是使用传统的贝叶斯概率分布。一个概率被认为是一个子集(包括全集, 该情况下则认为任何一个事件都有可能, 也即"我不知道")的概率而不是单个特定事件/类的概率。

而证据深度学习, 是在传统深度学习的架构下, 引入证据理论的思想, 在传统深度学习模型的预测输出后加入一个证据调节模块来优化预测, 并在此过程中计算此预测的不确定度(uncertainty)来评估模型的预测的确定性。该方法最早由 Murat Sensoy, Lance Kaplan, Melih Kandemir^[7]在 2018 年共同提出, 该方法不同于经典的证据理论, 而是基于其思想, 将每个样本的输出作为模型生成的“证据”, 引入迪利克雷分布来对模型的输出进行证据调节。该方法能够计算模型输出的不确定度, 并能进一步评估模型输出的性能以及面对负样本攻击(adversarial attack)的鲁棒性。而袁冰, Thierry Deneoux^[8]等人则在 Sensoy^[7]等人的基础上对该算法中在不同类的决策成本不平衡问题上进行了优化。

经过实验证明, 证据深度学习在解决传统深度学习分类算法中的过于自负判决表现以及面对负样本的无能为力等问题中表现出了优异的性能, 然而作为较为新兴发展的理论和算法, 鼓励更多人认识相关知识和加入该领域研究成为证据理论持续发展的长远之计。

§ 1.4 可视化深度学习算法的应用前景

近年来,随着互联网技术的发展和越来越多开发者投入到机器学习领域的研究中,许多机器学习相关工具和开发框架的产生,让机器学习的实现变得相对简单。初学者能够通过 Tensorflow, Pytorch, scikit-learn 等机器学习开发框架方便地搭建神经网络模型并进行模型训练。

并且,机器学习同时也日益成为更多领域的万精油,许多高校也逐渐将机器学习或深度学习列入大学生的必备知识储备清单中,如何让深奥数学理论知识,复杂繁琐的计算过程和抽象的模型概念以一种直观的、动态的、易于接受的形式呈现给学生或初学者成为了目前推广、吸引更多人了解相关技术的方向。

可视化在近年来随着大数据与机器学习的发展逐渐成为另一个热门词汇。可视化以图表、图形、动画等方式将数据呈现给使用者,让分析和处理变得更加直接便捷。如果能通过动画的方式展示深度学习算法中的核心过程,会让初学者更容易接受相关知识,并且动画的短小、内容密度高等特点能方便学习者反复回看而不需要在冗长的文献中寻找问题答案。

因此,可视化深度学习算法成为了最为直接、相对方便简单、成本较低的教学方式之一。

§ 1.5 本文研究内容及目标

§ 1.5.1 研究内容

本文将以介绍证据深度学习为主线,有以下具体内容:

- (1) 研究证据深度学习算法的核心过程,比较其与传统算法的差异。
- (2) 使用合适的可视化工具制作相关动画过程,并开发可视化代码接口,使其具有较好的可拓展性。
- (3) 使用合适的工具搭建一个简易的网络应用程序,用于给学习者在线观看介绍动画并以实时可交互的方式体验两者算法模型的效果。

§ 1.5.2 研究目标

针对本文的研究内容,制定了以下几项目标:

- (1) 可视化工具选取需要满足简单易用、符合直觉、不需要深入学习、修改简单等要求,让动画的生成和修改变得简单直接。
- (2) 可视化代码具有良好的接口特性,能够便于后续生成不同网络模型的动画展示以及证据网络模型的调节过程。
- (3) 可视化动画尽量简单直观,将复杂的概念和计算过程以简单的动画展示。

(4) 网络应用程序的功能应该包含以下几个模块:

(1) 具有视频播放功能, 用于可视化工具生成的动画。

(2) 具有用户可交互模块, 便于输入图像。

(3) 具有结果展示模块, 结果展示的可读性高, 且提供下载接口。

(5) 程序的部署简单, 不需要繁琐的操作步骤。

(6) 网络应用程序结构清晰, 做到前后端分离, 程序代码可读性高, 便于后续增加功能模块及优化操作。

§ 1.6 本文组织结构

通篇论文共分为五章。

第一章介绍了目前深度学习和证据深度学习研究背景以及可视化算法的研究意义, 并且提出了本文的研究内容以及研究目标。

第二章主要介绍传统深度学习和证据深度学习的基本概述, 分析了传统深度学习存在的问题, 并对证据深度学习的运行过程以及对深度学习存在的问题的优化方法进行了介绍, 最后比较了两者的相关性能。

第三章主要介绍算法的可视化实现工具—Manim 的基本情况, 介绍核心方法以及给出案例, 并通过它生成了一个简单的介绍证据深度学习的动画视频。

第四章主要介绍了网络应用程序的搭建, 使用 streamlit 搭建网络应用程序, 用 fastapi 搭建模型后端响应接口, 并介绍使用 docker 以及 docker-compose 部署网络应用程序。

第五章总结全文, 陈述了本文的主要工作, 介绍了本文的创新点, 并且展望未来, 提出下一步的工作目标和任务。

第2章 深度学习与证据深度学习

本章重点介绍了传统深度学习的基本概念和运行过程，并介绍了几个经典的深度学习卷积网络，接着提出了传统深度学习目前存在的问题，引出了证据深度学习，介绍了证据深度学习的原理与损失函数，最后对两种算法的性能做了比较。

§ 2.1 传统深度学习概述

本节重点介绍传统深度学习机器的运行过程，并介绍深度学习在图像分类中的核心网络层，随后简单介绍了几种经典的深度学习图像分类网络。

§ 2.1.1 传统深度学习简介

深度学习(deep learning)是机器学习的一大重要分支，其利用多层神经网络(Neural Networks)和各种算法对图像以及文本等问题进行分析处理，深度学习在大类上归于神经网络，但和神经网络在具体实施上有许多不同。学习的核心任务是通过分层网络对输入进行分层次的特征提取，以此解决传统的机器学习任务中前期特征提取与筛选等繁琐的工作。

深度学习的最核心的算法之一同时也是最为热门的词汇之一卷积神经网络(Convolutional Neural Networks, CNN)最早由 LeCun^[4]提出，其提出的第一个卷积神经网络 LeNet 为后续的神经网络层级提供了一个经典的卷积网络模式。LeNet 网络内部结构如图 2-1 所示。

深度学习凭借着卷积网络的功能特性，去除了传统图像处理的特征工程(feature engineering)等前期操作，成为了计算机视觉和图像分析处理的一大重要法宝。不仅如此，深度学习在搜索技术，数据挖掘，机器翻译，自然语言处理(NLP)，推荐算法等领域也逐渐发挥着越来越重要的作用。

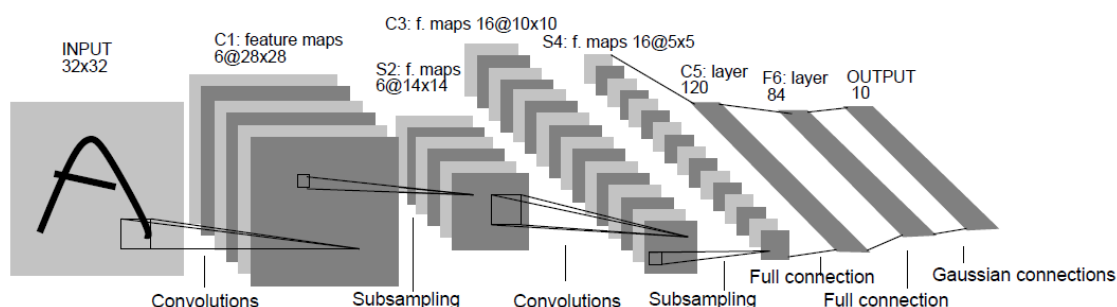


图 2-1 LeNet 网络内部结构^[4]

§ 2.1.2 传统深度学习运行过程

深度学习或神经网络从外部上都可以视其为一个黑匣子，通过输入图像或文

字等待分析处理的数据，经过各个网络层的相互配合从而学习输入数据的特征，并根据任务需要做出相应的预测输出。

在一个经典的图像分类深度学习网络中，输入的图片一般会先经过多层卷积层进行特征的提取，其中一个卷积层包括卷积层与池化层，这也是 LeCun 在 1998 年提出的 LeNet 卷积神经网络^[4]中经典的卷积层模式。在经过多层卷积层的特征提取后，深度学习会通过多个全连接层进行对多维特征的展平，此过程类似于传统神经网络的神经元，通过对提取特征的“深度学习”，最终得到模型的输出结果。

图像分类任务的模型训练过程中，每一个样本的输入经过深度学习网络层的处理和学习最终得到模型的预测结果，此过程是为前向传播(Forward Propagation)。接着，模型的输出结果与该样本的分类真实值(ground truth)一起，输入到预先设定好的损失函数(loss function)中，其中损失函数一般为均方值或最大似然估计，随后通过梯度下降(gradient decent)等最优化方法进行反向传播(BackPropagation, BP)，以期对网络中所有参数计算损失函数的梯度，以进一步更新参数来最小化损失函数，因此模型的训练过程则为得到最小化损失函数的模型各参数的过程。若模型各参数用 Θ 表示，损失函数用 L 表示，则模型的训练过程可用式(1)表示。

$$\min_{\Theta} L(\theta) \quad (1)$$

§ 2.1.3 传统深度学习常用术语

在深度学习中，沿袭了神经网络一些基本术语如权重，偏置等，另外加入了一些深度学习特有的卷积、池化等概念，总体来说，一个经典的深度学习图像分类网络主要包含以下几个组成部分：

(1) 卷积层 Convolutional layer: 深度学习中的卷积层不同于信号处理的卷积处理，而是通过一个卷积核对前一层的图像进行处理。我们将一张图像所有的像素构成一个 $m \times m$ 的矩阵，而卷积核可以看成是一个 $n \times n$ 的矩阵，其中 $m > n$ ， n 一般取奇数。所谓卷积，可以看成卷积核矩阵与图像部分位置矩阵的 Hadamard 乘积即点乘。将卷积核按规律的逐行扫描前一层的矩阵(输入图像或前一层卷积后的结果)，即可得到对应卷积核特征提取图像。例如，我们选取一个 3×3 的用于检测垂直边缘的卷积核，输入的图像如图 2-2 左侧所示时，可以发现输入的图像有明显的黑白垂直边界，输入图像与卷积核的乘积结果如图 2-2 右侧所示，可以发现该图像经过卷积核的卷积后，其黑白分界线被明显提取。若输入的像素为 $m \times m$ 维，卷积核大小为 $f \times f$ ，则卷积后的图像维度为 $(m-f+1) \times (m-f+1)$ ，另外，卷积层的运行还受以下两个参数影响：

(1) 填充 padding: 填充是为了解决在一幅图像中边框像素点参与卷积运

算次数较少,从而无法有效获得边框像素信息的问题所做的处理。假设 padding 值为 p (一般取 1), 则会在图像四周增加 p 层 0 像素点作为填充, 然后再进行卷积处理。若取 padding 为 p , 输入的图像为 $n \times n$, 卷积核为 $f \times f$, 则填充了的卷积后的图像维度为 $(n+2p-f+1) \times (n+2p-f+1)$ 。

(2) 步长 stride: 步长指的是每次卷积核移动的步数, 默认为 1, 步长参数会影响卷积核提取特征的密度, 步长越大, 卷积的次数越低, 得到的信息会随之减少。若填充参数为 p , 步长为 s , 输入图像为 $n \times n$, 卷积核为 $f \times f$, 则输出图像维度为 $((n+2p-f)/s+1) \times ((n+2p-f)/s+1)$ 向下取整, 则最终卷积后的维度图像如图 2-2 所示。

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

图 2-2 卷积提取边缘特征示例

(2) 池化层 pooling layer, 池化层一般跟在卷积层之后, 其目的是为了减少图像像素的维度, 加快计算。所谓池化, 即将一组数据放入一个“池子”中, 设定一种标准来代表这个“池子”, 如 MaxPooling 则为选取该“池子”的最大值, AvgPooling 则为选取该“池子”的平均值。MaxPooling 示例如图 2-3 所示。池子的参数与卷积核相似, 同样为 $((n+2p-f)/s+1) \times ((n+2p-f)/s+1)$ 向下取整, 池化与卷积的差别仅在于对核中选中的元素所处理的方式不同。池化层也因为仅对卷积层提取的特征做数学上的简单压缩, 因此人们通常将卷积层与池化层的组合称为一个卷积层(Conv layer)。

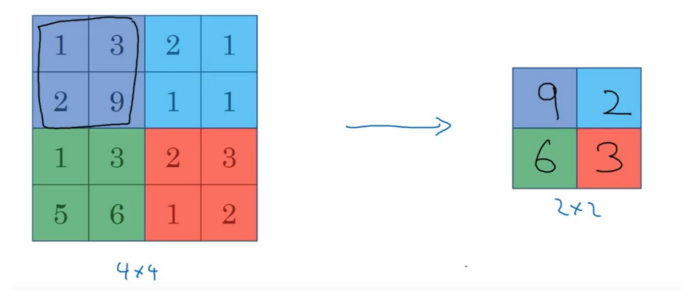


图 2-3 Maxpooling 池化层示例

(3) 全连接层 full-connected layer: 全连接层上的每一个结点都与上一层的所有节点相连, 其一般是用来整合前面卷积层所提取的特征参数。全连接层的思想类似于传统的神经网络, 在传统神经网络中, 经过特征工程后提取到的特征输

入到仅由多个、多层神经元构成的神经网络中进行学习。由于全连接层全相联的特性，其参数一般也是最多的。

(4) **Dropout**: 是一种防止过拟合的模型训练技巧。其核心思想是在一个神经网络中，一层中的神经元以一定的概率 p 停止工作，即该神经元输出为 0，这样为了防止多个隐藏层单元的相互作用导致过拟合现象。**Dropout** 在防止模型训练产生过拟合的表现上非常有效，已经成为了常用的模型训练技巧之一。

(5) **激活函数 activation**: 激活函数是对每一层网络的输出引入非线性变化的方式。如果不加入激活函数，那么模型的正向训练过程就仅为简单的矩阵线性变化，在此情况下，多层、深层的神经网络并不会比简单的单层线性变化的网络效果好。一般情况下，激活函数接在每层网络后，激活函数的输出作为下一层的输入，常见的激活函数有 sigmoid 函数，ReLU 函数等，其中 ReLU 函数是目前最为常用的激活函数。sigmoid 激活函数如图 2-4 左侧所示，ReLU 激活函数如图 2-4 右侧所示。

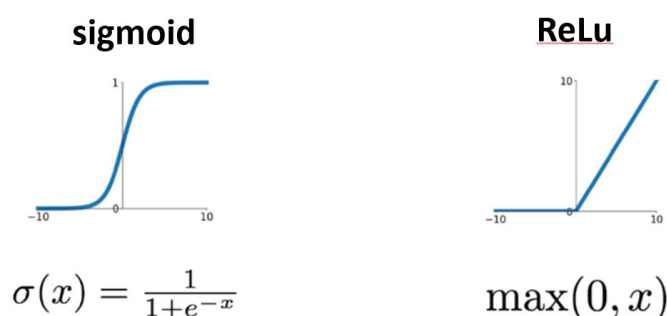


图 2-4 常见激活函数图像

(6) **损失函数 loss function**: 损失函数又称代价函数(cost function)，在优化问题中，损失函数常作为学习准则，通过最小化损失函数来评估学习的效果。在机器学习分类问题中，常用交叉熵损失函数(cross-entropy loss function)作为学习准则。交叉熵损失函数如式(2)所示，其中 y 为样本真实值， \hat{y} 为模型预测值。交叉熵损失函数本质是信息理论中的交叉熵在分类问题中的应用，最小化交叉熵等价于最小化观测值与估计值的相对熵即两者概率分布的 KL 散度，KL 散度是一种提供无偏估计的代理损失，KL 散度如式(3)所示，其中 X 为样本数据， ω 为模型参数， $f(X, \omega)$ 为输入样本 X 后模型对各个类的预测值， $p(y | X)$ 为输入样本 X 的真实的类向量，KL 散度的计算公式如式(12)所示。

$$L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \quad (2)$$

$$KL[f(X, \omega) || p(y | X)] \quad (3)$$

通过以上介绍,我们可以发现,相较于传统的神经网络,多个隐藏层以及每层的神经元个数一旦稍微增加,总体参数量是非常巨大的,而在深度学习卷积网络中,若某层的卷积核的维度为 $f \times f$,一共有 n 个不同的卷积核,那么该卷积层的参数仅有 $n \times f \times f$ 个,与传统神经网络相比,这样的参数是非常可观的,而最终一个复杂的卷积网络的参数最终大部分由将参数整合起来的全连接层产生。所以,可见卷积网络的优势与方便性都要优于传统复杂神经网络,这也是为什么卷积神经网络如此受欢迎的原因。

§ 2.1.4 经典的卷积网络模型

本小节将对流行的几种经典的卷积神经网络做一个简单的介绍,分别如下:

(1) LeNet-5: LeNet-5 是由 LeCun^[4]在 1998 年提出的第一代经典的卷积神经网络模型。其提出的卷积层与池化层结合的两层卷积网络与三层全连接层的结构为后续卷积神经网络提供了较为经典的模型范式。其内部结构如图 2-1 所示。其第一层卷积网络由 6 个 5×5 的卷积核组成,若一个样本的输入维度为 32×32 ,经过第一层卷积后得到 6 个不同的特征提取卷积核处理的 28×28 卷积图像。接着通过一个 2×2 的 maxpool 层缩小图片维度,增加信息密度。随后又经过一个卷积层进行更深层次的特征提取。

(2) AlexNet: 由 Alex Krizhevsky, Geoffrey Hinton 等人共同提出的一个较为大型的卷积网络^[5]。该网络与 LeNet-5 网络有很多相似之处,但其规模更加庞大,并且使用 ReLu 激活函数作为每层网络的输出。其内部结构如图 2-5 所示。

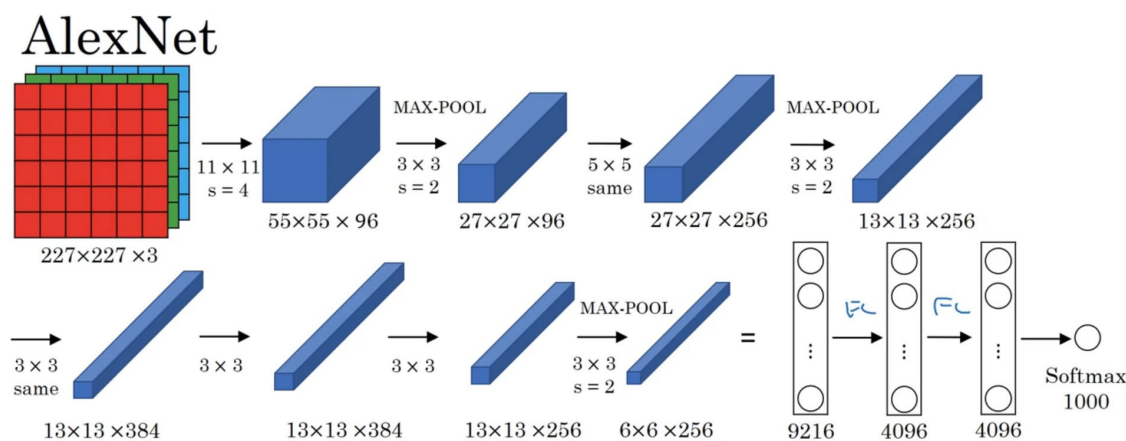


图 2-5 AlexNet 结构图^[5]

(3) VGG-16: VGG-16 是由 Oxford 的 Visual Geometry Group^[9]在 2014 年提出的更为庞大的网络结构。其与 AlexNet 相比,最大的改进之处在于采用连续几个 3×3 的卷积核来代替 AlexNet 中卷积层较大的卷积核。采用这种连续的卷积核的效果优于单个大卷积核,且卷积核的参数更小。所谓 16 是指其有 16 个有参

数的网络层级，虽然卷积层的参数更小，但是网络总体上的训练参数还是要比 AlexNet 高出一两个数量级。其内部结构如图 2-6 所示。

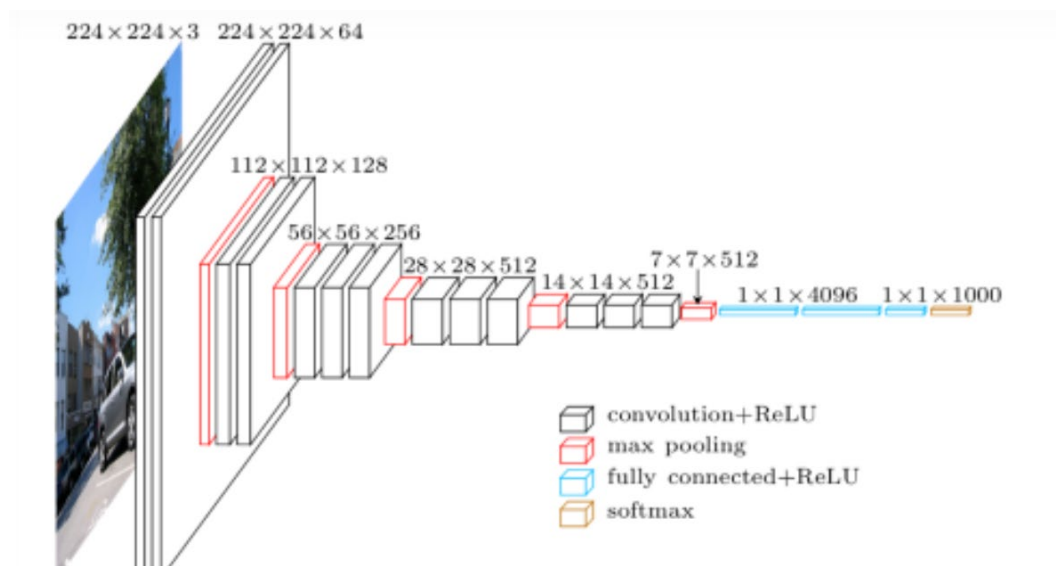


图 2-6 VGG16 模型结构^[9]

不难发现，这些较为经典的卷积神经网络在结构上有一定的相似性，如在多分类任务中，一个卷积网络往往会经过多层的卷积层的特征提取处理，随后将提取后的特征送入全连接层进行整合学习，最后经过 softmax 得出每个分类的占比预测。在接下来小节中，我们将会讨论这些卷积网络会遇到的问题。

§ 2.2 传统深度学习的问题

本节重点提出了目前传统深度学习在图像分类等分类任务中存在的问题。在互联网技术高度发展的今天，使用一个较为经典的网络模型对能够轻易获取到的庞大的数据集进行一个简单的训练，往往能获得较高的性能表现，得到一个测试准确率(test accuracy)较高的模型似乎并不是什么难事。因此，如今深度学习面临的挑战往往不在于测试准确率，而在于模型的可靠性与有效性，表现在两个方面：

- (1) 模型对目标做出预测时的把握程度。
- (2) 模型对不相关样本的“抗性”。

这两个方面的要求同时也说明了目前深度学习存在的两个问题：

- (1) 对目标做出过于自信(overconfident)的预测。
- (2) 对任何一个目标都会严格做出一个预测。

以上两个问题，归根结底就是模型做出预测时的不确定度有多高。在接下来的小节中，我们将会讨论深度学习遇到的这两个问题。

§ 2.2.1 过于自信的预测

首先,所谓过于自信是指,在分类任务中,倘若我们将模型的 softmax 层的输出结果看作是模型判断输入样本为各种类的概率,每个种类的概率不相上下且维持在较低水平,模型仅仅根据其中的最大值做出相应判断,如图 2-7 所示。无论结果正确与否,这样的模型表现明显是过于自信的,同时也反映出了模型的训练不到位等问题。

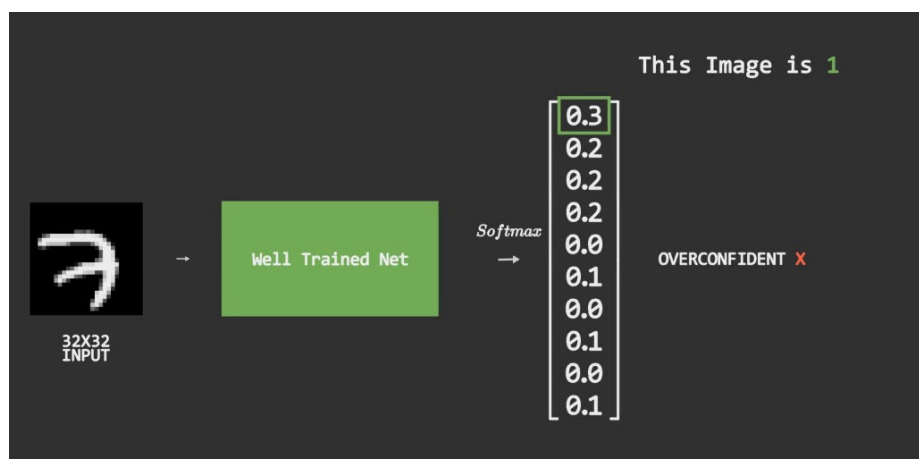


图 2-7 传统深度学习做出过于自信的预测

§ 2.2.2 严格做出一个预测

其次,在面对与分类任务完全不相关的样本(即负样本)时,模型也会严格的做出一个判断,即模型永远不会说“我不知道”(I don't know),如图 2-8 所示。诚然也许输入的负样本中确实有一些特征与分类任务中训练样本不谋而合,但将风景看成香蕉的模型是不能体现人工智能的智能性的。因此在此情况下,模型做出判断的不确定程度往往比最终所作出的预测结果或是模型对每个分类的预测概率更有说服力。



图 2-8 传统深度学习面对负样本时的预测表现

§ 2.3 证据深度学习

本节将重点介绍证据深度学习的运行原理，以及其对于解决深度学习在模型预测中遇到的问题的优化方法，最后再介绍证据深度学习中两个重要的损失函数。

§ 2.3.1 证据深度学习原理

首先，证据深度学习是基于 Dempster-Shafer 证据理论思想^[6]。Dempster-Shafer 证据理论是对贝叶斯理论在主观概率上的推广，它将信任质量(belief mass)或信任函数(belief function)分配给识别框架的子集，用以标志结果可能是该集合中的某个特定状态，所分配到的子集即为可能状态的集合，这种对集合的相对确定与对状态的相对不确定，丰富了推理的逻辑。这样处理的显著好处是，当子集为全集时，其实是对所作出判决的不确定的确定，也即为我不知道(I don't know)。

其次，根据主观逻辑(Subjective Logic)的观点，其用迪利克雷概率密度函数形式化 DS 理论对判决框架的信任分配，并且规定每一个子集都是互斥(mutually exclusive)的(比如每一个单独的数字类)。这样的观点又把 DS 理论框架中的信任质量函数又重新以状态变量的主观意见(Subjective opinion)的形式转换到了传统的主观概率问题上来。并且，主观逻辑将每一个类的信任质量 b_i 与整体的判决不确定度 u (Overall uncertainty) 做了如式(4)所示的规定。

$$u + \sum_i^K b_k = 1 \quad (4)$$

如式(4)所示，其中不确定度 u 与各信任质量 b 都为非负且相加为 1。证据深度学习认为，每个样本输入模型后所得到的对每个类的预测结果都应该作为该样本对每个类所贡献的证据，若单个样本对第 k 个类所贡献的证据为 $e_k (>=0)$ ，则我们可通过式(5)计算该样本对第 k 类的信任质量与整体不确定度 u (overall uncertainty)，其中 S 为迪利克雷长度(Dirichlet strength)，其值为所有证据 e_i 加 1 后的和。

$$\begin{aligned} b_k &= \frac{e_k}{S}, u = \frac{K}{S} \\ S &= \sum_i^K e_i + K = \sum_i^K (e_i + 1) \end{aligned} \quad (5)$$

若令 $\alpha = e + 1$ 则有式(6)：

$$b_k = \frac{(\alpha_k - 1)}{S}, S = \sum_i^K \alpha_i \quad (6)$$

式(6)中 α_i 为迪利克雷分布的参数。在证据深度学习中，我们称模型输出的预

测结果为一个观点 *opinion* 或是一个证据(*evidence*)。另外，我们可以计算，当给定一个观点时，证据深度学习对每个类的预期预测概率如式(7)所示，并且每次预测都会附加一个整体不确定度 u 作为此次预测结果的不确定指标。

$$\hat{p}_k = \frac{\alpha_k}{S}, u = \frac{K}{S} \quad (7)$$

因此，由式(7)的计算公式，证据深度学习依托迪利克雷分布，使传统神经网络能够对分类任务组建预测意见，即提供证据来为后续的预测进行一个更为确定的不确定度推算。并且 $(\alpha_{ij} - 1)$ 为第 i 个样本对第 j 个类提供的证据也即为 e_{ij} ， α_i 为第 i 个样本的迪利克雷分布的参数。

训练过程中，当一个样本图片中的特定模式如一个圈相关于其中一个预测类如数字 0，则相应的迪利克雷分布参数如 α_0 应该会相应的增大，以说明该样本对模型预测数字零所做的证据贡献为图片中圈的识别。接下来，我们将会讨论证据深度学习的损失函数处理每个样本提供的证据的原理。

§ 2.3.2 证据深度学习中的损失函数

首先，证据深度学习通过引入迪利克雷分布，对传统深度学习神经网络输出进行了证据调节。迪利克雷分布是一组连续多变量概率分布，是多变量普遍化的 *beta* 分布，其常常作为贝叶斯统计的先验概率。多维迪利克雷分布在参数 $\alpha_1, \dots, \alpha_k > 0$ 上，其概率密度函数定义为式(8)上部所示，其中 $B(\alpha)$ 为 *Beta* 函数，其可由 *gamma* 函数 $\Gamma(\cdot)$ 定义，*Beta* 函数定义式如式(8)下部所示：

$$f(x_1, \dots, x_k; \alpha_1, \dots, \alpha_k) = \frac{1}{B(\alpha)} \prod_{i=1}^K x_i^{\alpha_i-1} \quad (8)$$

$$B(\alpha) = \frac{\prod_{i=1}^K \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^K \alpha_i)}, \alpha = (\alpha_1, \dots, \alpha_K)$$

不难发现，概率密度分布函数中的 α_i-1 即为样本对第 i 类提供的证据。

证据深度学习中的损失函数分为两部分，第一部分为模型的预测损失 (*prediction error*)，第二部分为证据的调节损失 (*adjustment error*)。

因为迪利克雷是连续多变量分布函数，证据深度学习使用预测值与真实值差的平方的期望作为预测的损失函数。对于每个样本 i 的预测损失函数定义如式(9)所示，其中 y_i 为样本 i 的真实值， p_i 为模型对样本 i 的预测结果， θ 为模型参数。

$$L_i^{pred}(\theta) = \int \|y_i - p_i\|^2 \frac{1}{B(\alpha_i)} \prod_{j=1}^K p_{ij}^{\alpha_{ij}-1} dp_i \quad (9)$$

由式(10)所示的方差公式特性：

$$E[p_{ij}^2] = E[p_{ij}]^2 + \text{Var}(p_{ij}) \quad (10)$$

可得式(11)所示简化公式:

$$\begin{aligned} L_i^{\text{pred}}(\Theta) &= \sum_{j=1}^K (y_{ij} - E[p_{ij}])^2 + \text{Var}(p_{ij}) \\ &= \sum_{j=1}^K (y_{ij} - \hat{p}_{ij})^2 + \frac{p_{ij}^2(1 - \hat{p}_{ij})}{(S_i + 1)} \end{aligned} \quad (11)$$

由式(11), 预测损失函数分为两部分, 前一部分是模型预测的错误损失 L_{ij}^{err} , 后一部分是神经网络对训练集中的每个样本 i 产生的迪利克雷实验方差 L_{ij}^{var} (variance of the Dirichlet experiment)。式(11)具有以下三个性质:

性质一: 任何 $\alpha_{ij} \geq 1$, 总有 $L_{ij}^{\text{var}} < L_{ij}^{\text{err}}$ 。这保证数据拟合优先于方差估计。

性质二: 对于一个其真实值为 j 的训练样本 i , 当发现该样本有新的证据加入到 j 类的预测中时, 即 α_{ij} 增加, 则 L_i^{err} 会减小, 当该样本对 j 的证据被移除, 即 α_{ij} 减小, 则 L_i^{err} 会增加。该性质保证了损失函数优先控制数据的拟合。

性质三: 对于一个其真实值为 j 类的训练样本 i , 当最大 α_{il} 减小且 $l \neq j$ 时, 即不是目标类的证据被移除, L_i^{err} 会减小。此性质根据不确定性建模中的学习损失衰减理论^[11], 保证了模型建模过程中当模型确定地且正确地预测出结果时, 样本提供的正向证据会被发掘更多, 而在预测其他类时, 因为不正确且不确定, 模型应该避免提取目标类的预测证据。

以上三个性质使得神经网络能够通过式(11)的预测损失函数来优化神经网络去为每个类的正确样本生成更多的证据, 并且通过移除不必要的误导证据来避免模型错误分类。预测损失还会仅在证据的生成导致更好的拟合的情况下通过增加证据来衰减训练集上的预测方差。

除此之外, 证据深度学习的损失函数还包含另外一部分——证据调节损失函数, 其主要用到 KL 散度(Kullback-Leiber divergence)来评估样本迪利克雷分布与均匀迪利克雷分布的差异。KL 散度源于信息论, 是一种量化两种概率分布 P 和 Q 之间差异的方式, KL 散度又称为相对熵(relative entropy), KL 散度的计算公式定义为式(12)所示, 其中 P, Q 为两种不同的分布, $D_{KL}(P||Q)$ 的值代表 P 分布与 Q 分布的相似程度。

$$D_{KL}(P || Q) = - \sum_i P(i) \ln \frac{Q(i)}{P(i)} \quad (12)$$

在模型训练过程中, 样本的某些特征或模式(pattern)的存在可能有利于某 i 类的预测, 此时模型因为该模式或特征的存在而为该类生成更多的证据(即 e_i 或 α_i 的增大), 以减小整体的损失, 从而使模型更好的拟合。例如, 样本中一个大圈的存在可能会是预测结果为 0 的证据, 所以模型在发现样本中有某个大圆时, α_0 即

数字 0 的输出或样本对数字 0 提供的证据应该增加。然而因为 6 或 9 的样本中也存在同样的大圈特征，模型应该在训练这类负样本时相应减少该类证据的生成，来减小对于 6 或 9 这类样本的损失，以免夸大大圈存在即为数字 0 的证据，从而减小整体的损失。然而上述的做法在“负”样本数量相对较小时，反而会增加模型对于大圈存在的疑惑，虽然减小了负样本的损失，但是反而增加了整体的损失，在此情况下，模型会对错误的样本生成不必要的证据从而造成误导。这样的误导虽然在模型能够预测出正确类时不会是一个问题，但证据深度学习认为，若一个样本无法正确被分类，模型对其生成的所有证据的应当衰减为 0 即不确定度为 1。要实现这样对无法被正确分类的样本进行惩罚，我们首先引入均匀迪利克雷分布。均匀迪利克雷分布为所有迪利克雷参数 α_i 都为 1，在此情况下 $S=K$, $u=1$ 。因此，证据深度学习的证据调节损失函数定义为：

$$L_i^e(\Theta) = \text{KL}[D(\mathbf{p}_i | \tilde{\alpha}) \| D(\mathbf{p}_i | \mathbf{1})]$$

$$= \log \left(\frac{\Gamma(\sum_{k=1}^K \tilde{\alpha}_{ik})}{\Gamma(K) \prod_{k=1}^K \Gamma(\tilde{\alpha}_{ik})} \right) + \sum_{k=1}^K (\tilde{\alpha}_{ik} - 1) \left[\psi(\tilde{\alpha}_{ik}) - \psi \left(\sum_{j=1}^K \tilde{\alpha}_{ij} \right) \right] \quad (13)$$

其中， $\tilde{\alpha}_i = y_i + (1 - y_i) \odot \alpha_i$ ， $\Gamma(\cdot)$ 为 gamma 函数， $\psi(\cdot)$ 为 digamma 函数。由两部分损失函数可得，证据深度学习总体损失函数为：

$$L(\Theta) = \sum_{i=1}^N L_i^p(\Theta) + \lambda_t L_i^e(\Theta) \quad (14)$$

其中 $\lambda_t = \min(1.0, t/10) \in [0, 1]$ 为退火系数(annealing coefficient)，用于控制证据调节惩罚力度，其中 t 为模型训练迭代次数(epoch)。通过退火系数，逐步增加 KL 散度的证据调节损失函数的影响，从而使模型能够开拓更为广阔参数空间，避免过早地将“负”样本收敛到均匀迪利克雷分布(即证据为 0，不确定度为 1)，因为在训练早期，负样本不仅仅作为其前置样本正确预测出的类的负样本（如 0 与 6 的关系），负样本还需要作为自己目标类的正样本，所以这类错误样本或负样本很可能在以后的训练中被重新正确分类，避免过早将其打入冷宫能够提升模型训练的有效性。

以上我们分析了证据深度学习的运行原理与损失函数，可以发现，所谓证据学习，其实是在传统深度学习的基础上，对传统深度学习的输出结果做一个证据调节，使结果变得更为确定。并且通过证据深度学习的损失函数来促使模型在训练中能够有效处理样本提供的证据。

§ 2.4 性能比较

在本节，我们使用 pytorch 实现上述提及的证据深度学习与传统深度学习，

使用两种模型对 Minist 数据集进行训练，并从对旋转数字图像的识别处理与对完全不相关的负样本的预测能力对两种模型做比较，着重关注证据深度学习中加入不确定度对模型预测的影响。

§ 2.4.1 Mnist 数据集

Mnist 数据集，全称为 Mixed National Institute of Standards and Technology database，是美国国家标准与技术研究院收集的大型手写字符数据库，其中包含 60000 张手写字符图片的训练集与 10000 个测试用例的测试集。数据集图片示例如图 2-9 所示。



图 2-9 mnist 数据集示例

minist 数据集是广泛用于机器学习或深度学习的模型训练测试集，相当于机器学习领域中的"hello world"程序。在本节中，我们使用 mnist 数据集来对比两种不同的深度学习方法的效果。

§ 2.4.2 模型网络结构

本次实验的网络结构我们使用经典的 LeNet 结构，其包含 2 个卷积层，分别使用 20 个和 50 个 5x5 的卷积核对手写字符进行特征提取，本次使用的网络不使用池化层来减小图像维度，为保证卷积层的完整性，在池化层参数上，我们使用 1x1 最大卷积核 (MaxPooling) 进行池化。在两个卷积层后我们加入一个有 500 个节点的全连接层用于特征的整合。本次网络层使用 Pytorch 构建，构建完成后使用 pytorch 的打印函数即可打印模型网络层次结构，打印出的网络结构层次如图 2-10 所示。

Layer (type:depth-idx)	Output Shape	Param #
Conv2d: 1-1	[-1, 20, 24, 24]	520
MaxPool2d: 1-2	[-1, 20, 24, 24]	--
ReLU: 1-3	[-1, 20, 24, 24]	--
Conv2d: 1-4	[-1, 50, 20, 20]	25,050
MaxPool2d: 1-5	[-1, 50, 20, 20]	--
ReLU: 1-6	[-1, 50, 20, 20]	--
Flatten: 1-7	[-1, 20000]	--
Linear: 1-8	[-1, 500]	10,000,500
ReLU: 1-9	[-1, 500]	--
Linear: 1-10	[-1, 10]	5,010
Total params: 10,031,080		
Trainable params: 10,031,080		
Non-trainable params: 0		
Total mult-adds (M): 20.29		

图 2-10 模型结构示意图

使用 `pytorch.nn.module` 构建 LeNet 网络层代码如图 2-11 所示。

```
class LeNet(nn.Module):
    def __init__(self, dropout=False):
        super().__init__()
        self.use_dropout = dropout
        self.conv1 = nn.Conv2d(1, 20, kernel_size=5)
        self.maxpool1 = nn.MaxPool2d(1)
        self.relu1 = nn.ReLU()
        self.conv2 = nn.Conv2d(20, 50, kernel_size=5)
        self.maxpool2 = nn.MaxPool2d(1)
        self.relu2 = nn.ReLU()
        self.flatten = nn.Flatten()
        self.fc1 = nn.Linear(20000, 500)
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = self.relu1(self.maxpool1(self.conv1(x)))
        x = self.relu2(self.maxpool2(self.conv2(x)))
        x = self.flatten(x)
        x = self.relu3(self.fc1(x))
        if self.use_dropout:
            x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return x
```

图 2-11 pytorch 自定义网络模型

由上述证据深度学习的介绍，证据深度学习与传统深度学习在模型层面是完全一致，仅有一些微小的差别。证据深度学习在传统深度学习的输出上引入了新的损失函数对其进行了证据调节，为了保证证据调节中的 α 符合迪利克雷分布参数的特征即大于等于 1，证据深度学习在模型的输出层上使用 Relu 激活函数来确保这项性质。

§ 2.4.3 模型对比

首先, 我们对一个测试用例进行有规律的旋转 180 度, 观察模型在各角度对样本的预测情况。在传统的深度学习方法中, 模型对数字 1 倾斜角度有一定的包容性, 在手写字符“1”逆时针旋转 0-50 度和 150-180 度的情况下, 模型预测其为数字“1”的概率仍然很高(由于对称性), 但在数字“1”旋转 60-110 度时, 模型却逐渐将数字“1”预测为数字 7, 从直观上来看这是因为过大的倾斜角度使得“1”更像数字 7 的上部分如图 2-12 minit 数据集示例, 而当数字 1 逆时针旋转到 110-140 度时, 此时数字 1 则像数字 5 的上部分。传统深度学习模型对各旋转角度的预测结果如图 2-12 左图所示。

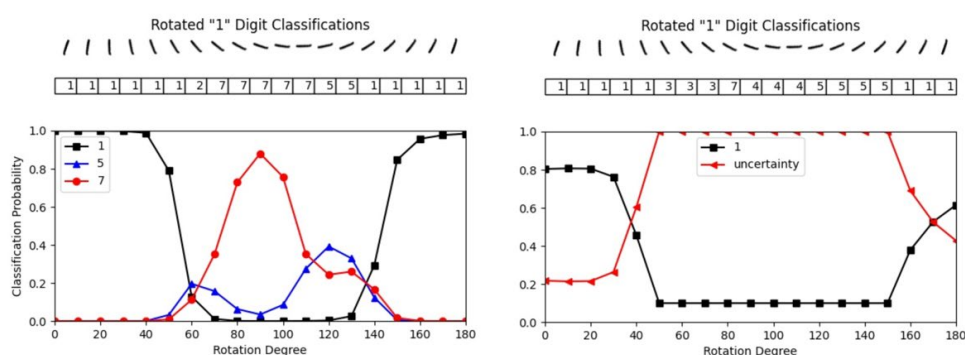


图 2-12 两种模型对旋转数字的预测

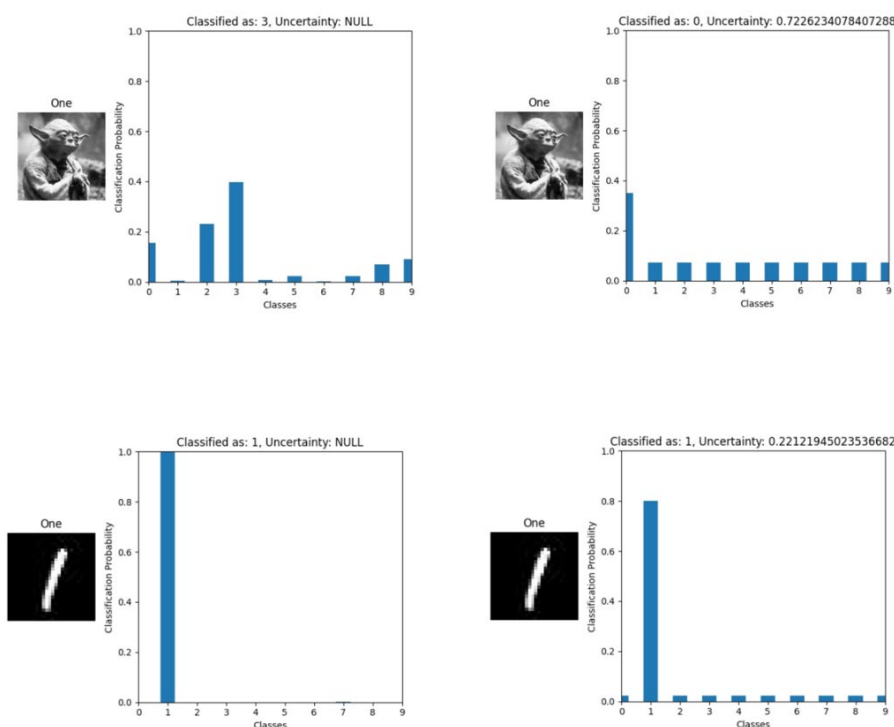


图 2-13 两种模型对同一样本的识别结果

而在证据深度学习中，由于有不确定度的加入，对于传统深度学习中错误判决的结果，其更多地是利用不确定度来作为自己的预测结果，而不是仅仅从各预测概率中选取最大值作为自己的预测结果。证据深度学习对数字“1”不同角度的预测结果如图 2-12 右侧所示，可以发现，证据深度模型在 0-50 和 150-180 度的预测结果与传统模型的预测结果相似，但证据模型在数字“1”旋转到 50-150 间，其预测的不确定度居高不下，以此说明模型对这种情况的疑惑与不确定，在此情况下，模型对各类的预测概率已经不那么重要了。

另外，我们还通过上述介绍的模型比较了两种深度学习对不相关样本的预测结果，结果如图 2-13 所示。其中图 2-13 上部为传统模型（上左）与证据模型（上右）对同一个不相关样本的预测结果。可见，对于一个不相关样本，相较于传统深度学习方法从各预测概率严格选择一个最大值作为预测结果，证据深度学习以较高的不确定度(0.72)来说明模型对于预测该样本的疑惑程度。而图 2-13 下图则为传统模型（下左）与证据模型（下右）对同一个测试样本的预测结果。由式 (4)所知，模型对各类的预测概率之和与不确定度是成互补关系的，当不确定高于一个阈值时，证据深度学习中模型对各类的预测概率就已经不像传统深度学习那样重要，即这种情况更多地说明模型的我不知道(“I don't know”)。我们认为合理设置阈值，充分利用不确定度的表达效果可有效改善模型网络对于不相关样本的处理结果，也能更好地控制模型盲目自信的训练结果的情况。

第3章 算法的可视化实现

本章介绍了什么是可视化技术,并且重点介绍了数学可视化工具 Manim 的主体架构,常用方法与相关示例,最后介绍了如何使用 Manim 来实现传统深度学习与证据深度学习的算法可视化。

§ 3.1 可视化技术简介

可视化技术(visualization)最早源于在科学研究领域中,人脑分析数据的逻辑与多种信息源产生的庞大数据量不匹配,从而导致对大量数据的处理的低效,阻碍科学研究的发展。可视化技术涉及计算机图形学,图像处理,计算机视觉等多个领域,其核心思想是利用图形或图像处理技术,将庞大的数据转换成主观的,动态的图形或图像显示出来,从而让人能够更加宏观地分析处理数据。经过长时间的发展,可视化技术已经成为了数据科学分析处理,决策分析等涉及金融、医疗、科学研究、交通等多个领域的一个重要技术手段。

目前,可视化技术大体可分为:

(1) 科学数据可视化。主要针对物理世界中客观的科学数据的可视化,其数据往往是实验或调查中得到的真实数据。科学数据可视化的核心在于真实地、准确地展示数据的真实状态。

(2) 信息可视化。主要应用于教育等非专业研究领域,其核心目的在于降低对数据或科学过程的理解难度,以图形或动画等具象化方式展示数据的相关关系或复杂难懂的科学过程。

(3) 可视分析学。结合信息可视化,人机交互,数据挖掘等技术,通过交互等方式,根据人的意志和目的,对数据进行二次加工处理。在信息可视化技术上加入了人这一关键要素,主要应用于数据分析,自主推理,智能决策等领域。

在本章中,我们要实现的算法可视化是属于信息可视化的一环,主要是通过图像和动画等具象化的方式,将证据深度学习的核心原理与运行过程做一个简单的介绍,便于理解。在本章接下来的章节中,我们通过一款数学可视化工具 Manim 来实现上述目标。

§ 3.2 数学可视化工具 Manim

§ 3.2.1 Manim 简介

Manim 是由 3blue1brown 开发的一款开源的数学动画渲染引擎,专为解释性数学视频而设计。我们都知道,数学世界是奇妙、缤纷、严谨并且有趣的,然而正是因为这样的奇妙与多样,数学世界同时是复杂的、多维的、深刻的。不同知

识可能要从不同的角度,不同的维度理解。然而,Manim 的出现完美地融合了这两者,其凭借动态、华丽且顺滑的动画和其针对数学领域的渲染引擎,将复杂的“数学公式”通过动画的形式展现了数学世界的奇妙与多彩。3b1b 作者本人也通过 Manim 制作了许多有意思的视频,其中涵盖线性代数,机器学习,数学分析,傅里叶级数等复杂难懂的知识。其制作的部分视频如图 3-1 所示。

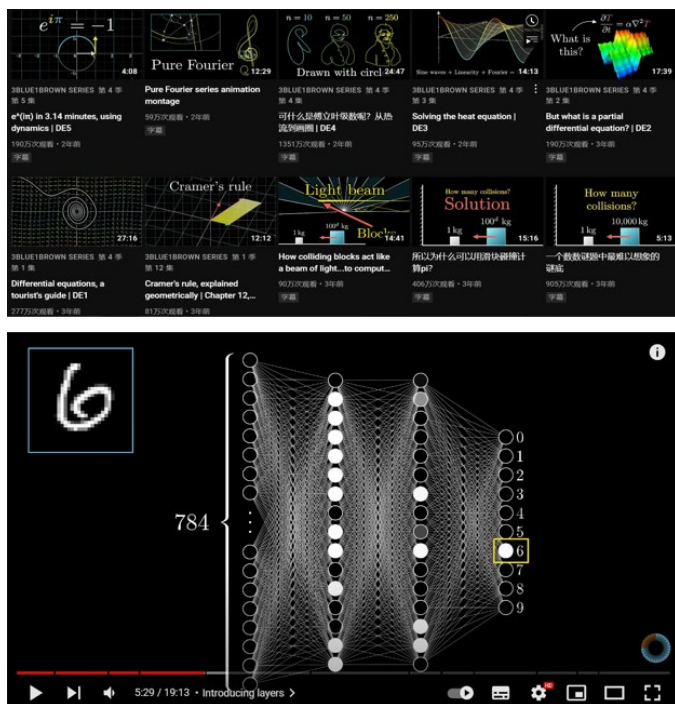


图 3-1 Manim 制作的动画示例

在接下来的小节中,我们将介绍 Manim 的主体架构,常用方法以及相关案例,然后着重介绍如何使用 Manim 来实现证据深度学习的可视化展示。

§ 3.2.2 Manim 库架构

本文使用的 Manim 版本是社区版,使用 python 的 pip 命令可快速安装,如图 3-2 所示。

```
# Install manimgl
pip install manimgl

# Try it out
manimgl
```

图 3-2 安装 Manim

Manim 库主要是 manimlib 下的文件,文件虽多但其结构清晰,主要包括:

(1) Mobject 数学物件: mobject(Mathematical object)数学物件是动画中用到的基础物件,主要包括矩阵,概率,几何图形,参数方程,坐标系统等基础数学

元素，如图 3-3 所示。Mobject 是 Manim 动画运行的基本元素。

```

— mobject/ # 数学物品
  |— mobject.py          # 所有mobject的父类
  |— types/ # 4种子类mobject
    |— dot_cloud.py      # Dot cloud (PMobject 的一个子类)
    |— image_mobject.py  # 插入图片
    |— point_cloud_mobject.py # PMobject (点集构成的 mobject)
    |— surface.py        # ParametricSurface
    |— vectorized_mobject.py # VMobject (向量化的 mobject)
  |— svg/ # 和svg有关的mobject
    |— svg_mobject.py    # SVGMobject
    |— brace.py          # 大括号
    |— drawings.py       # svg 图像的一些特殊 mobject
    |— mtex_mobject.py   # 依赖 LaTeX 实现的文字
    |— tex_mobject.py    # 依赖 LaTeX 实现的文字
    |— text_mobject.py   # 依赖 manimpango 实现的文字
  |— changing.py         # 动态变化的 mobject
  |— coordinate_systems.py # 坐标系统
  |— frame.py            # 和 frame 有关的 mobject
  |— functions.py        # 参数方程
  |— geometry.py         # 几何图形的 mobject
  |— interactive.py      # 可交互的物件
  |— matrix.py           # 矩阵
  |— mobject_update_utils.py # 一些定义的更新程序
  |— number_line.py      # 数轴
  |— numbers.py          # 可以变化的数字
  |— probability.py      # 和概率有关的 mobject
  |— shape_matchers.py   # 适应其它物体大小的 mobject
  |— three_dimensions.py # 三维物体
  |— value_tracker.py    # ValueTracker(存储数的 mobject)
  |— vector_field.py     # 向量场

```

图 3-3 Mobject 元素概览

(2) Animation 动画：Animation 类接受 Mobject 类，来实现 mobject 包括创建，移动，转换等动态效果。其可实现的动画效果如图 3-4 所示。

```

— animation/ # 动画
  |— animation.py      # 动画的基类
  |— composition.py    # 动画组
  |— creation.py       # 和 Create 有关的动画
  |— fading.py         # 和 Fade 有关的动画
  |— growing.py        # 和 Grow 有关的动画
  |— indication.py     # 一些用于强调的动画
  |— movement.py       # 和移动有关的动画
  |— numbers.py        # 实现对 DecimalNumber 数字的变化
  |— rotation.py       # 和旋转有关的动画
  |— specialized.py    # 一些针对特殊项目的不常用动画
  |— transform_matching_parts.py # 自动匹配部分的 Transform
  |— transform.py      # 一些 Transform 变换
  |— update.py         # 从函数实现 update

```

图 3-4 Manim 可实现动画类概览

(3) Scene 场景：Scene 场景是 Manim 中的画布，在画布上使用 Animation 实现不同的 mobject 动起来的动画效果，每个场景的动画通过 Scene 类中的 construct 函数定义。Scene 类包括的场景如图 3-4 Scene 类场景概览所示。在 Scene 类中的 construct 函数中定义好场景的动画过程后，通过命令行命令可直接渲染目

标类，具体过程如图 3-5 创建方形转换成圆的动画效果示例所示。

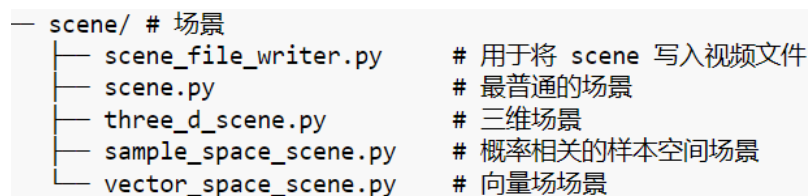


图 3-4 Scene 类场景概览

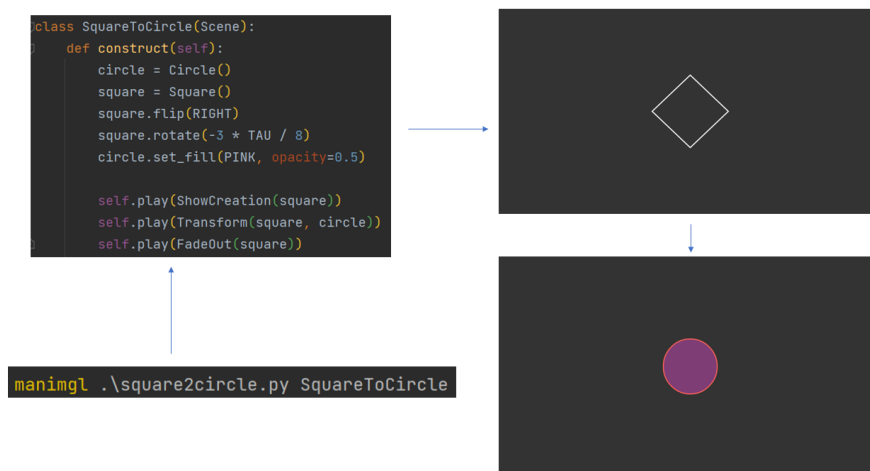


图 3-5 方形转成圆的动画制作过程

(4) 其他配置文件：另外 Manim 库还提供包括相机，Tex 模板，GLSL 渲染脚本等一系列优化动画的配置文件。

总体来说，Manim 的架构清晰，使用简单，是如今数学可视化工具的一大主流，极大简化了数学等复杂但有趣的领域的解释性工作，在接下来的小节中，我们将会陆续介绍 Manim 中用到的常用方法，并且通过相关示例来感受 Manim 的简易操作。

§ 3.2.3 Manim 库常用组件

根据上述 Manim 架构介绍，在制作一个动画时，我们需要创建所需要的数学物件，通过 Animation 实现的动画效果，将物件动起来，最后将动画在场景 Scene 下显示出来。理清了 Manim 创建动画的具体流程，接下来我们将对常用的数学物件和动画效果做一个简单的介绍。

(1) 常用数学物件

(1) 矩阵或向量 Matrix/Vector: Manim 中提供了 Matrix 类供创建矩阵，Matrix 类是属于 VMOBJECT(Vectorized MObject)的子类，其内部提供了许多获取矩阵内元素的方法，如我们可以在一个向量中标记高亮框，实现代码如下如图 3-6 左图所示，显示效果如图 3-6 右图所示。

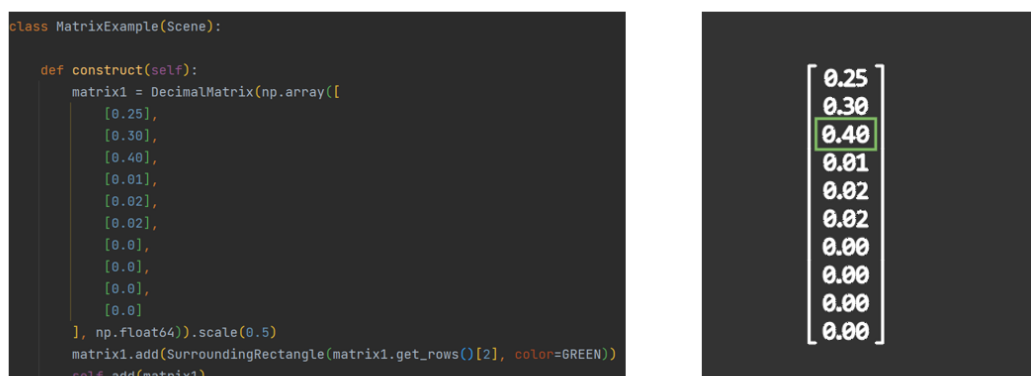


图 3-6 Manim 创建向量

(2) 文本 Text/TeX: 在一个动画视频中, 我们或多或少地需要稍加注释, 或数学公式对图像加以说明, Manim 提供 Text 类与 Tex 类方便使用者往动画中加入文字或公式。其中 Tex 类支持 Latex 文本, 方便生成复杂表格和数学公式。使用 Manim 生成 KL 散度计算公式示例代码如图 3-8 左图所示, 生成结果如图 3-7 右图所示。

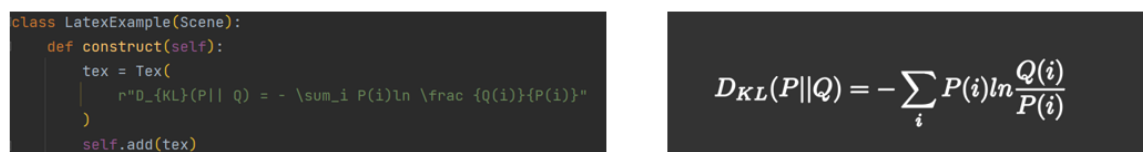


图 3-7 Manim 创建 Latex 公式

(2) 常用动画效果

Manim 的动画效果的相互配合使得图形能够以有逻辑、动态的方式呈现出来。这些常用的动画效果一般属于 animation 的子类, 大体包括对象的出场, 对象的移动, 对象的转换与对象的退场, 各种常用动画效果如下:

(1) 对象的出场: Manim 中对数学物件的出场有多种动画支持, 在 animation 类下有写 Write, 飞入 FadeIn, 展示创建 ShowCreation 等基本出场动画类, 他们都接受一个数学物件, 并以对应的方式对数学物件实现对应的入场效果。另外, 在 Scene 下, 也可以通过其场景自有方法 add 实现物件的加入。

(2) 对象本身的运动: 在 Manim 中, 每一个数学物件都可自行调用函数实现自身的运动, 如 Text().to_edge(), Text().to_corner(), Text().move_to(), Text().next_to()等方法分别将物件移动到边框, 角落或移动到另一个物件的对应位置, 或以紧挨的方式移动到某一物件的对应方位 (上下左右)。

(3) 对象的转换: 平滑的转场一直是动画的一大特色, Manim 使用 Animation

中的 Transform, ReplacementTransform, FadeInTransform 等动画类实现物件之间的过度转场。

(4) 对象的退场: 与出场类似, Manim 提供消除 UnCreate, 飞出 FadeOut 等类实现退场, 另外 Scene 下还有 remove, clear 等方法实现物件的退场。

以上介绍的动画效果除了对象本身的运动外, 都要依托场景本身的方法 play() 才能够在场景中呈现, 如 play(ShowCreation(Text("Hello world"))), play(Transform(Text("hello world"), Text("Manim"))))。

可以发现, Manim 中创建动画的过程, 整体上是比较直观的, 且符合视频动画制作逻辑。在接下来的小节中, 我们将会通过 Manim 实现本文介绍的证据深度学习的可视化展示。

§ 3.3 基于 Manim 库实现证据深度学习的可视化

在本小节中, 我们将通过 Manim 来证据深度学习算法的可视化动画。首先介绍了可视化实现的整体流程, 之后再分节讲解流程中各模块可视化的实现过程。

§ 3.3.1 可视化动画的主体框架

首先, 本次算法的可视化动画的实现是面向有一定基础的学习者的, 旨在帮助其快速了解证据深度学习的运行过程与主体逻辑。其次, 本文制作的动画视频更多地是比较证据深度学习与传统深度学习的差异, 来引出证据深度学习的特点即不确定度与引入迪利克雷分布等让预测更加的“确定”的优化方法。因此本次动画展示的主体流程应该包括:

- (1) 传统深度学习的运行过程(以传统卷积网络为例)
- (2) 传统深度学习存在的问题
- (3) 证据深度学习的运行过程
- (4) 两种算法的性能比较

§ 3.3.2 封装函数说明

纵然 Manim 的代码函数相对完整, 但我们仍然封装了一些函数模块, 以适配我们的开发流程。封装的函数模块如下:

(1) 函数 create_vector: 创建一个向量 vector, 并附上相应的标签 text, 并且将物件定位在 aligned 的右侧, 函数放回一个数学物件组合 group。具体代码如图 3-8 所示。


```
def create_vector(self, matrix, aligned, text, matrix_scaler=0.5, interval=2, font_size=30):
    vector = DecimalMatrix(matrix).next_to(aligned, RIGHT * interval).scale(matrix_scaler)
    text = Tex(
        r"" + text,
        font_size=font_size
    ).next_to(vector, UP)
    group = VGroup(
        vector,
        text
    )
    return group
```

图 3-8 程序清单 3.3.2.1

(2) 函数 `create_arrow`: 创建一个从 `left` 到 `right` 的箭头, 并附上对应的标签 `text`, 函数返回箭头与标签的组合。具体代码如图 3-9 所示。

```
def create_arrow(self, left, right, has_text=True, text="", arrow_scaler=1.2, font_size=25):
    arrow = Arrow(left.get_right(), right.get_left()).scale(arrow_scaler)
    if has_text:
        text = Tex(
            r"" + text,
            font_size=font_size
        ).next_to(arrow, UP)
        group = VGroup(
            arrow,
            text
        )
        return group
    else:
        return arrow
```

图 3-9 程序清单 3.3.2.2

(3) 函数 `load_image`: 导入目录 `path` 对应位置的图片, 并创建对应标签 `img_text`, 函数返回图片与标签。具体代码如图 3-10 所示

```
def load_image(self, path, toedge=LEFT, height=2.0, image_scale=0.7, text="32X32\nINPUT"):
    img = ImageMobject(
        path,
        height=height
    ).scale(image_scale).to_edge(toedge)

    img_text = Text(text).next_to(img, DOWN).scale(0.3)

    return img, img_text
```

图 3-10 程序清单 3.3.2.3

(4) 函数 `create_layer_group`: 为实现卷积网络中多个卷积核形成的多层图像效果, `create_layer_group` 函数根据 `num` 创建对应数量的方形以实现多个被特征提取的图像效果, 并根据 `layer_name` 和 `size_text` 创建对应的卷积层名与层级维度标签, 具体代码如图 3-11 所示, 该函数实现的效果如图 3-12 所示


```
def create_layer_group(self, size=1.0, num=6, size_text="6X28X28", layer_name="convolution", buff=0.3):
    layers = [Square(side_length=size, color=GREY).set_opacity(1)]
    for i in range(1, num):
        cur = layers[i - 1].copy().set_color(GREY_B).move_to(
            layers[i - 1].get_center() + np.array((buff, buff, 0)))
        layers.append(cur)
    layer_bloc = VGroup(
        *layers
    )
    layer = VGroup(
        layer_bloc,
        Text(f"{size_text}").scale(0.3).next_to(layers[0], DOWN),
        Text(f"{layer_name}").scale(0.3).next_to(layers[0], DOWN * 2),
    )
    return layer
```

图 3-11 程序清单 3.3.2.4



图 3-12 卷积层网络可视化效果

(5) 函数 `create_full_connection`: 创建一个长方形来表示卷积网络中的全连接层, 其根据 `width`, `height` 来调节全连接层的大小, 并根据 `size_text`, `layer_name` 来创建对应标签, 函数返回全连接层与函数标签的组合, 具体代码如图 3-13 所示, 具体效果如图 3-14 所示。

```
def create_full_connection(self, width=0.2, height=4, size_text="120X1X1", layer_name="F1"):
    rec = Rectangle(width=width, height=height, color=GREY).set_opacity(1)
    flayer = VGroup(
        rec,
        Text(f"{size_text}").scale(0.3).next_to(rec, DOWN),
        Text(f"{layer_name}").scale(0.3).next_to(rec, DOWN * 2),
    )
    return flayer
```

图 3-13 程序清单 3.3.2.5



图 3-14 全连接层可视化效果

§ 3.3.3 可视化实现深度学习运行过程

由于本次的视频是面向有一定基础的学习者，关于深度学习网络的运行过程展示，我们仅从模型内部传递的总体过程上做出动画演示。

我们以 LeNet-5^[5]为例，使用 3.3.2 小节介绍的 `create_layer_group` 与 `create_full_connection` 函数构建了 LeNet-5 卷积网络的模型，模型构建效果如图 3-15 所示。

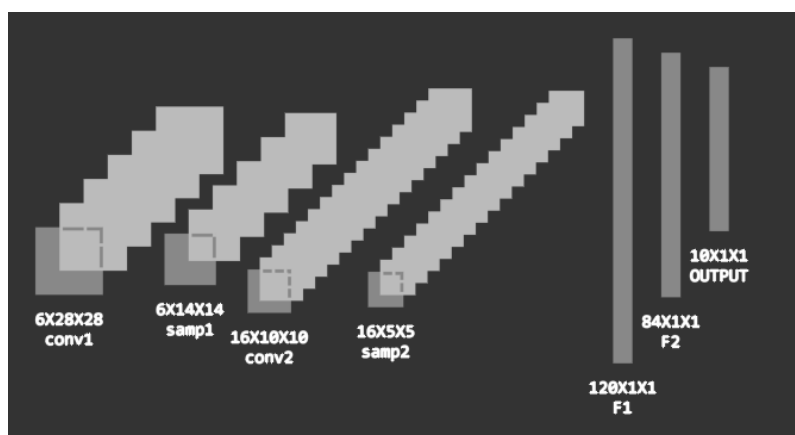


图 3-15 Manim 可视化卷积网络

接下来，我们实现训练图像的正向传播过程展示，按照实际可视化的要求，我们封装了 `connect_net_part` 函数，该函数将两层间的图像，通过一条 `Line` 物件连接，从而实现前一层的图像像素组(5X5/3X3)经过卷积/池化核处理后到后一层对应的像素点的过程，并且通过 `create_arrow` 和 `create_vector` 函数创建了对应的传播途径与输出概率结果的模拟效果具体效果。整体传播细节效果如图 3-16 所示。

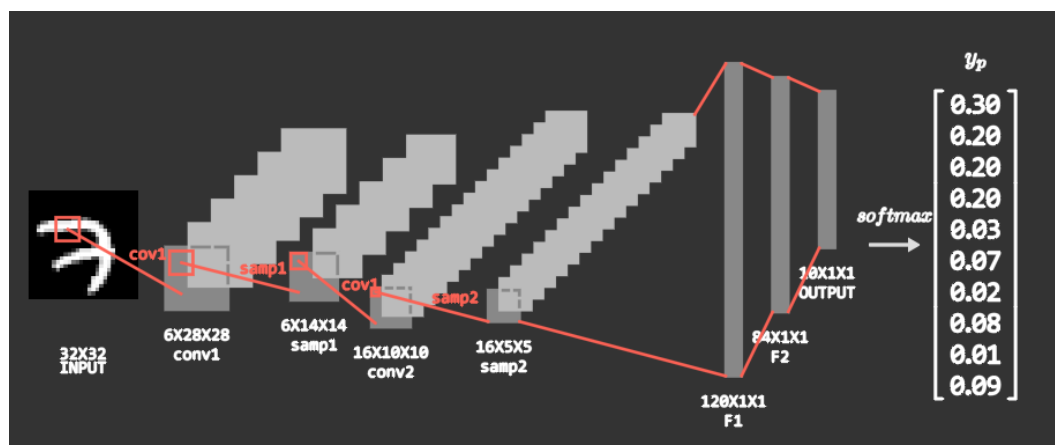


图 3-16 可视化网络传播过程

随后，我们将忽略卷积网络的过程细节，将卷积网络的处理简化为一个抽象的黑箱子，我们使用 `simple_network`(一个简单的矩形，如图 3-17 中蓝色矩形所示)来代替卷积网络，最后列出传统卷积网络的损失函数交叉熵(cross-entropy)的计算公式，完成卷积网络的简单介绍动画。具体效果如图 3-17 所示。

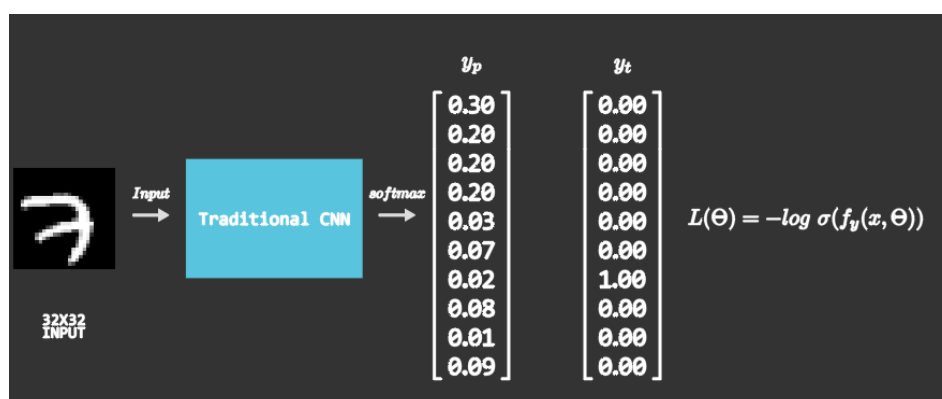


图 3-17 卷积网络简化效果

§ 3.3.4 可视化实现传统深度学习存在的问题

接下来，我们需要指出传统深度学习可能存在的问题：

- (1) 做出过于自信的预测
- (2) 预测负样本时，严格做出一个预测

首先，为了说明模型会做出过于自信的预测问题，我们模拟给模型输入一张数字图像，得到模型的预测结果，且各预测值普遍偏低，模型将其中最高的预测值作为自己的预测，具体演示效果如图 3-18 左侧所示。其次，对于模型在预测负样本时，无法做出有效预测的问题，我们模拟给模型输入一张非数字图像，模型仍然把预测概率的对应类作为自己的预测值，如图 3-18 右侧所示。至此，我们通过以上两个不同图像的模拟输入和输出来说明传统深度学习存在的问题。



图 3-18 可视化传统深度学习存在的问题

§ 3.3.5 可视化实现证据深度学习运行过程

经过前面的介绍，证据深度学习与传统深度学习的主要差别有：

- (1) 证据深度学习的输出层激活函数使用 ReLu 函数而不是 Softmax 函数，使用 ReLu 是为了保证每一个证据 e_i 都符合迪利克雷分布参数的性质即大于 0。
- (2) 证据深度学习将卷积网络的输出视为样本提供的证据，使用新的损失函数对样本进行证据调节。

因此，我们先直观地展示输出层激活函数的异同。这次我们使用动画展示效果 ReplacementTransform 类实现从 softmax 的输出到 ReLu 的输出的转换，具体转换过程如图 3-19 所示。

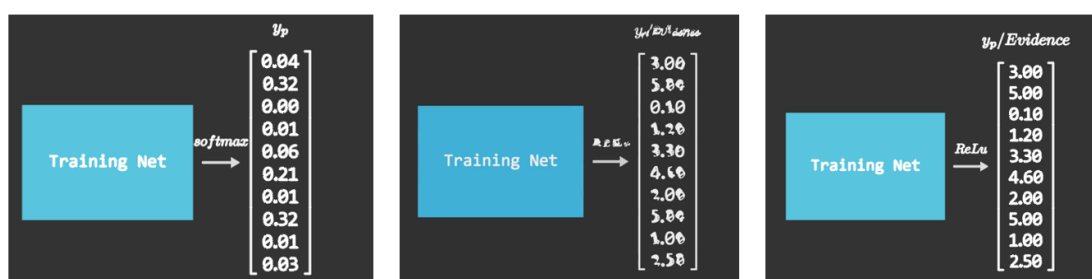


图 3-19 Softmax 输出转成 ReLu 输出过程

我们通过一个模拟的 Relu 输出来计算证据深度学习进行证据调节的各变化过程，具体代码如图 3-20 所示，具体计算公式如式(5)，(6)，(7)所示。

```
evidence_vector = np.array(evidence_matrix)
alpha = evidence_vector + 1
S = alpha.sum()
p_hat = alpha / S
K = 10
uncertainty = round(K / S, 3)
```

图 3-20 证据深度学习处理证据过程实现代码

随后我们通过 `create_vector`, `create_arrow` 等函数方法分别创建 α 向量和证据深度学习的预测概率输出向量, 并创建对应箭头来表明传播过程。接着, 我们通过动画效果将证据深度学习的证据处理环节展示出来, 展示效果如图 3-21 所示。

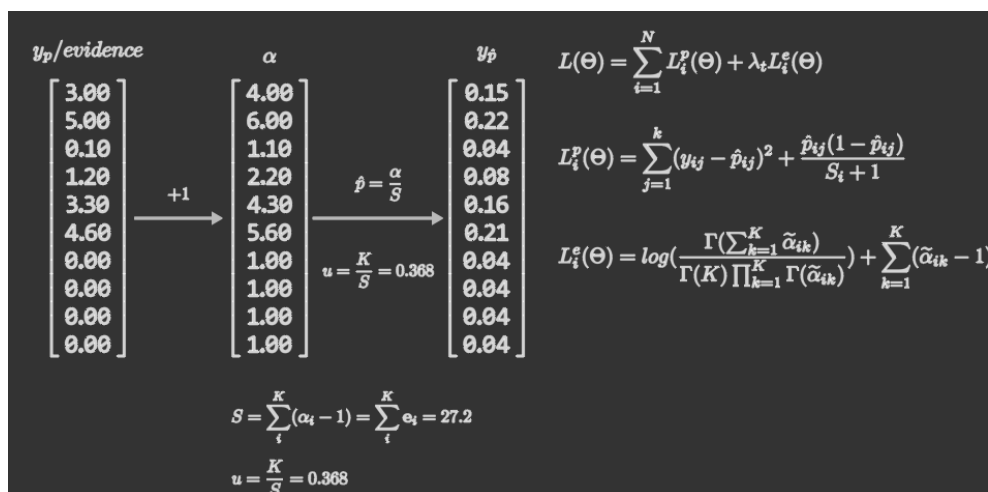


图 3-21 可视化证据处理过程

为了更加直观理解证据深度学习的证据处理模块, 我们把上述证据调节过程转换成一个简单的矩形来表示证据调节模块如图 3-22 左上图所示, 最后将卷积网络与证据调节模块结合如图 3-22 左下图所示, 最终形成我们的证据深度学习的模型框架如图 3-22 右下图所示。

以上, 我们从证据深度学习如何对模型的输出进行证据调节的各过程进行了介绍展示, 最后将证据深度学习的证据调节模块与传统卷积网络结合, 构成了证据深度学习的主体框架, 更直观地诠释了证据深度学习的“证据”二字如何作用。

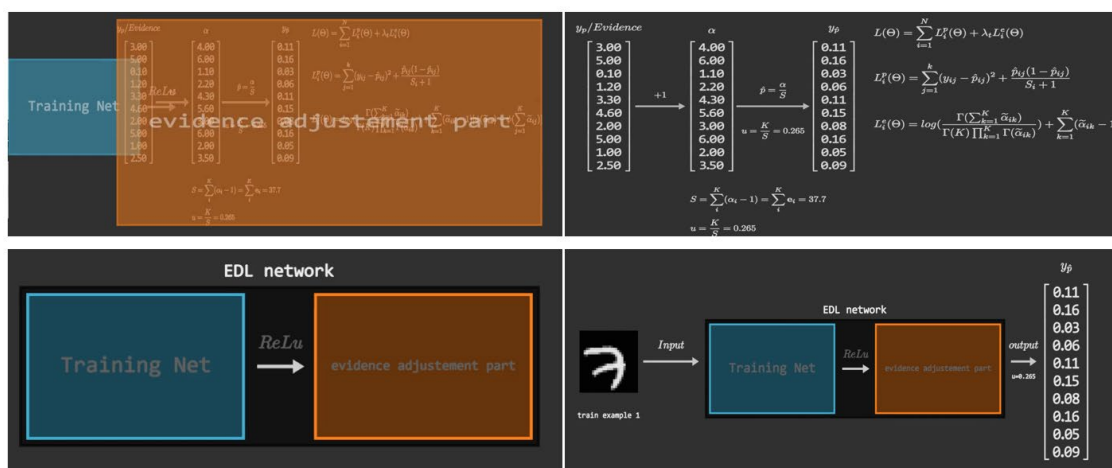


图 3-22 证据调节模块简化过程

§ 3.3.6 可视化实现比较两种模型的性能

最后，我们对证据深度学习与传统神经网络的预测性能做一个比较。

首先，我们输入一张测试集中的图片，比较两者的预测结果。具体可视化过程与上述类似，在此不再赘叙。对于测试集中的图片，两种深度学习方法的预测表现如图 3-23 左图所示。

接着，我们输入一张与训练集无关的图片，比较两者的预测结果，可以发现，证据深度学习对于不相关的图片展现出了较高的不确定。这种高不确定往往说明了模型对于此样本的困惑。具体结果如图 3-23 右图所示。将这两种结果在可视化动画中展示，效果如图 3-24 所示。

以上，我们通过 Manim 制作了一个简单的证据深度学习介绍动画，该动画对有一定基础的深度学习领域学习者快速了解证据深度学习的运行过程有一定的帮助。在接下来的章节中，我们将通过搭建一个网络应用程序，让参与者更加直接地感受证据深度学习的魅力与特点。

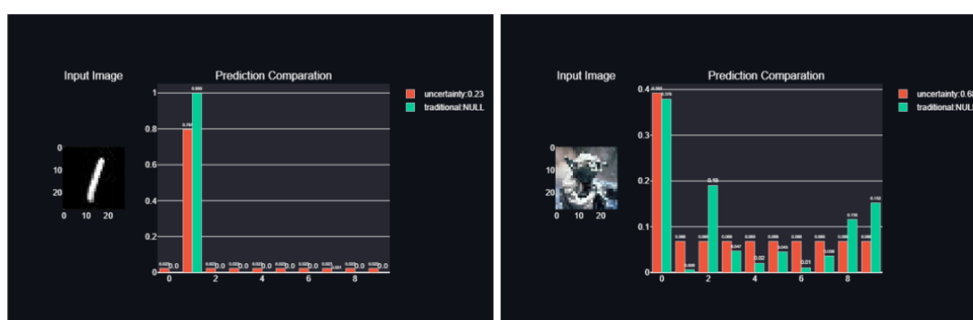


图 3-23 不同模型对不同样本的不同预测结果

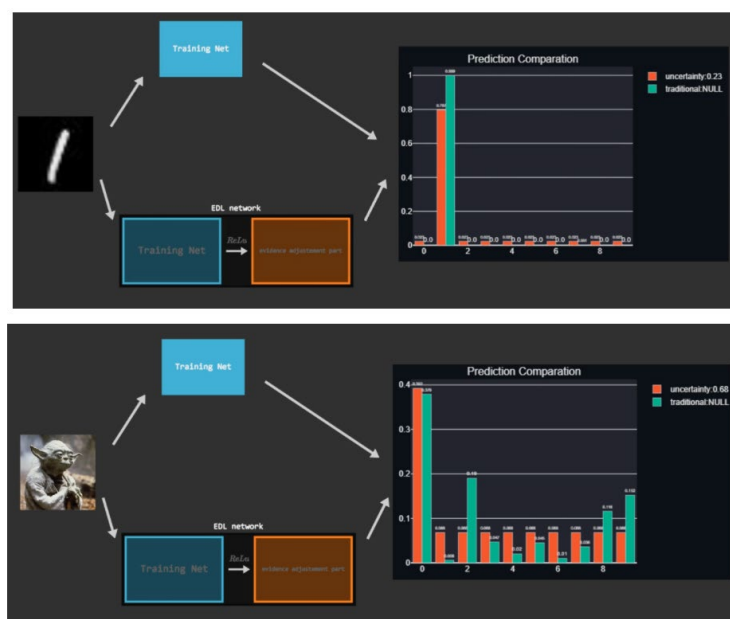


图 3-24 可视化两种模型的性能

第4章 证据深度学习网络应用程序

本章主要介绍了证据深度学习网络应用程序的搭建过程，介绍了如何用 streamlit 搭建简易网络应用程序，如何使用 fastapi 搭建简易的深度学习后端预测接口，最后介绍如何使用 docker 容器技术部署应用程序。

§ 4.1 需求分析

在本节中，我们先对目标网络应用程序(web application)所提供的核心功能模块做一个简单的分析，之后再介绍本次网络应用程序所使用的前后端分离架构。

§ 4.1.1 程序的核心功能

首先，本文所要搭建的网络应用程序目的是想提供一个在线平台让使用者能快速了解证据深度学习的特点与运行流程，所以所要搭建的网络应用程序应该要提供视频播放功能，用以播放第三章所制作的证据深度学习的动画。其次，为了能够让使用者进一步感受证据深度学习的魅力，本应用程序应该提供给使用者自定义参与预测的模块，因为本文的证据深度学习的训练是基于 mnist 手写字符集的，网络应用程序应该提供给使用者输入手写字符的接口，最后为了展示结果，应用程序应该提供相应的结果显示模块供使用者参考。

综上所述，网络应用程序提供的核心功能应该包括：

- (1) 视频播放模块
- (2) 用于预测的图片输入模块
- (3) 用于显示预测结果的结果展示模块

前期预设的各模块分布如图 4-1 所示。

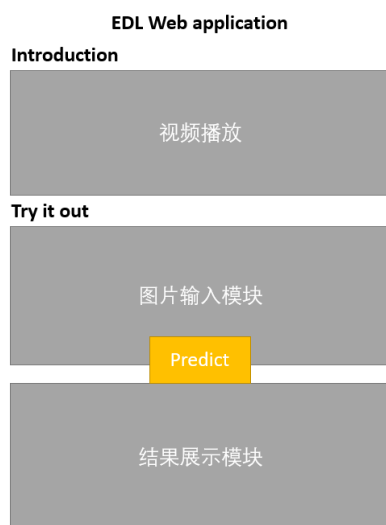


图 4-1 预设网络应用程序各模块分布

§ 4.1.2 前后端分离架构

前后端分离架构是目前互联网项目开发的常规模式，顾名思义，前后端分离架构即前端的 UI 逻辑与后端数据处理逻辑分离，实现两者的解耦。常见的前后端分离架构如图 4-2 所示。这样分离的架构让：

- (1) 前端更关注于与用户的交互，提升用户体验。
- (2) 后端更关注于数据的处理。
- (3) 两者通过 API 接口的形式进行请求通信，更容易拓展功能模块，开发效率更高。
- (4) 在大型的企业项目中，两者的分离可减少同一台服务器同时处理前端 UI 与数据计算的负载压力，让两者的部署更加灵活。
- (5) 某一端无法正常提供服务时，不影响另一端的正常运行。

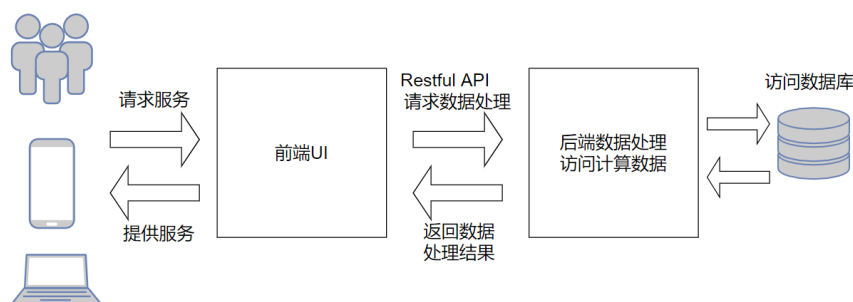


图 4-2 常见前后端分离架构模式

在本文搭建的网络应用程序中，我们将证据深度学习与传统深度学习的预测处理任务交由后端负责，而前端关注于界面的设计与用户的交互体验。两者通过 Restful API 接口的形式来发送请求和接收结果。这样的架构方便我们后续为该应用程序进行功能拓展，如自选参数，自定义模型训练等。在接下来的小节中，本文将会从前端界面开发，后端数据处理响应开发与程序部署等方面逐一对该网络应用程序的开发进行介绍。

§ 4.2 使用 streamlit 开发网络应用程序

本小节将会使用 streamlit 开发网络应用程序的前端模块，首先对 streamlit 做了一个简单的概述，介绍了 streamlit 的功能组件，最后使用 streamlit 搭建我们的目标网络应用程序。

§ 4.2.1 Streamlit 概述

Streamlit 是一个开源的 python 第三方库，是一个专注于构建机器学习和数据科学相关的网络应用程序(web app)的框架。Streamlit 开发简单，其内置的组件适

配于网页前端，让开发者不需要懂得前端开发知识仅凭几行代码就能轻易搭建炫酷的前端界面。图 4-3 是使用 streamlit 搭建的激活函数介绍应用程序，使用者可以选择不同的激活函数来快速了解相关函数的公式与图形。

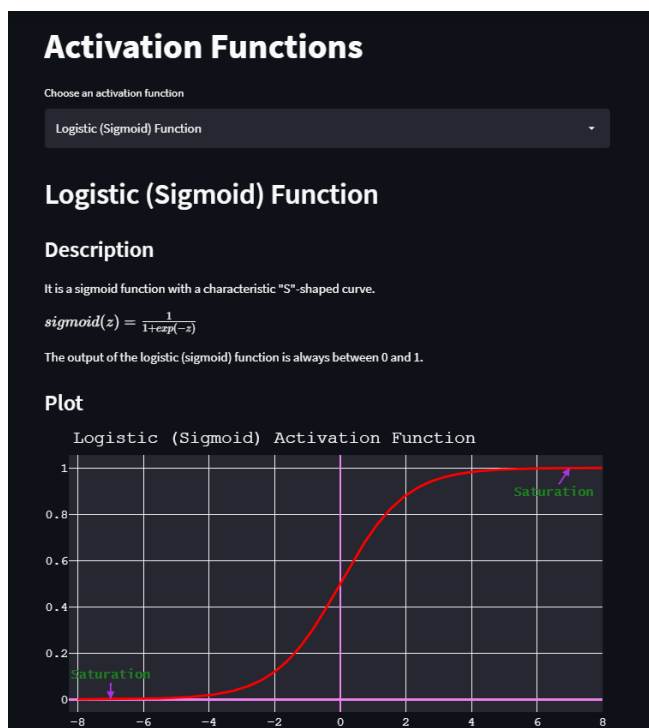


图 4-3 streamlit 网络应用程序示例

Streamlit 开发的主体逻辑是使用其内置的组件来装饰页面，如 `st.title("Title")` 能在程序的对应位置写上标题格式的文字(`st` 为 `streamlit` 的简写，在开发时使用 `import streamlit as st` 来简写 `streamlit`)。使用 `st.file_uploader("choose a file")` 能在程序的对应位置创建一个上传文件的接口，该方法返回上传文件的字节流形式。

这种直观、简单的开发形式让应用程序的开发变得简单，并且其内置的组件更是自带炫酷的特效或美化支持。使用 `streamlit` 能快速搭建一个简易的网络应用程序，快速分享自己的机器学习模型或数据分析结果。

§ 4.2.2 Streamlit 功能组件

如前文所述，`streamlit` 提供了很多内置组件，可让开发者能轻易创建对应的组件模块。现对 `streamlit` 常用的组件模块进行一个简单的介绍。

(1) 文本支持：

- (1) `st.title("My title")`: `title` 字号的文字，如图 4-4 左图所示。
- (2) `st.latex("\sum a_i")`: 支持 `latex` 字体的文字，如图 4-4 中图所示。
- (3) `st.code("a = c + b")`: 代码格式，如图 4-4 右图所示。

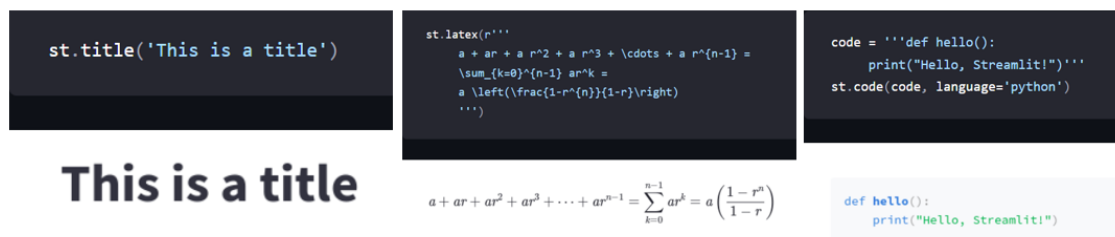


图 4-4 streamlit 文本组件

(2) 输入支持:

(1) `st.file_uploader()`: 文件上传组件, 若有上传文件, 该组件返回装有上传文件内容的 `UploadedFile` 类, 开发者可通过 `UploadedFile().get_value()` 方法获取上传文件内容的字节流, 具体用法如图 4-5 所示。

(2) `st.button("click")`: 按钮组件, 当被点击时返回 `True`。

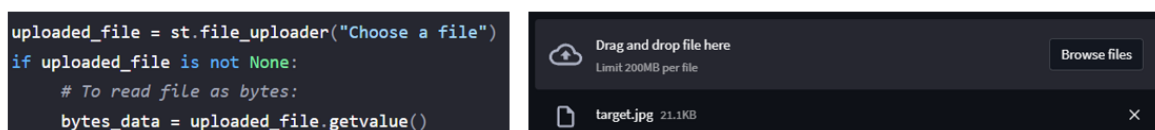


图 4-5 streamlit 上传文件组件

(3) 显示模块:

(1) `st.image()/st.write_image()`: 输入可以是 `numpy` 数组, 图片的字节流, 文件名甚至是 `http` 地址, 该模块都可以图片的形式显示内容。

(2) `st.video()`: 输入视频的字节流, 播放视频, 具体用法如图 4-6 所示。

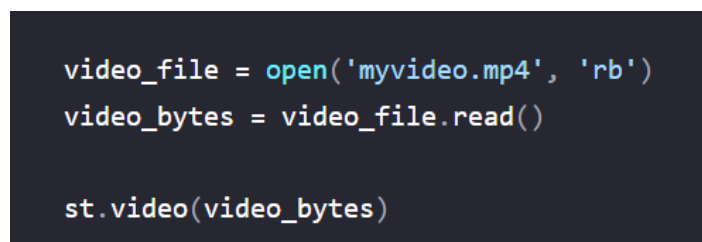


图 4-6 streamlit 视频播放组件

(4) 绘图模块:

streamlit 支持许多 python 的绘图框架, 甚至是 `Pandas` 库的 `DataFrame`。常见的绘图库支持有:

(1) `st.plotly(plotly_figure)`: 将 `plotly` 库绘画的 `figure` 显示出来。

(2) `st.pyplot(pyplot_figure)`: 将 `matplotlib` 库绘画的 `figure` 显示出来。

(3) `st.pydeck(pydeck_chart)`: 将使用 `pydeck` 库绘画的 `chart` 显示出来。

§ 4.2.3 使用 streamlit 搭建证据深度学习网络应用程序

首先, 根据图 4-1 预设的网络应用程序各模块结构图, 网络应用程序应该能够播放证据深度学习的介绍动画, 我们使用 `st.video()` 组件来实现视频动画的播放, 具体效果如图 4-7 所示。

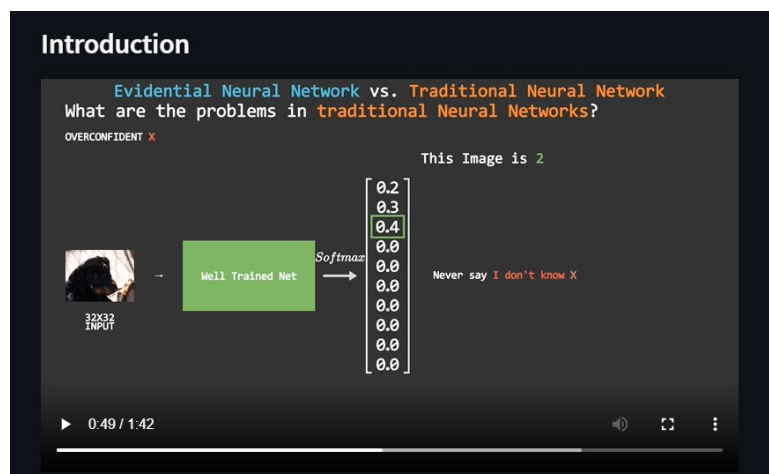


图 4-7 streamlit 播放动画视频

接着, 我们需要让使用者直接体验到证据深度学习与传统深度学习的差别, 需要提供一个图像输入接口, 供使用者输入图片并使用我们预训练好的模型进行训练。另外因为是手写字符的预测, 我们期望提供一个实时在线画板, 供使用者在线手写字符并直接预测, 其次, 为了比较证据深度学习对负样本的预测表现, 我们还提供了一个文件上传模块, 供使用者上传与手写字符集无关的或者其他手写字符图片。提供两种输入图片的方式, 让参与更加灵活, 提升使用者使用体验。

为了实现画板, 我们调用 `streamlit_drawable_canvas` 库的 `st_canvas` 类创建我们的画板, 该库是 `streamlit` 的扩展, 丰富了 `streamlit` 的功能与组件样式。`st_canvas` 使用方法如图 4-8 左侧所示, 画板的显示效果如 4-8 右图所示。

加上文件上传模块, 应用程序的输入模块整体效果如图 4-9 所示。



图 4-8 streamlit 实现画板模块

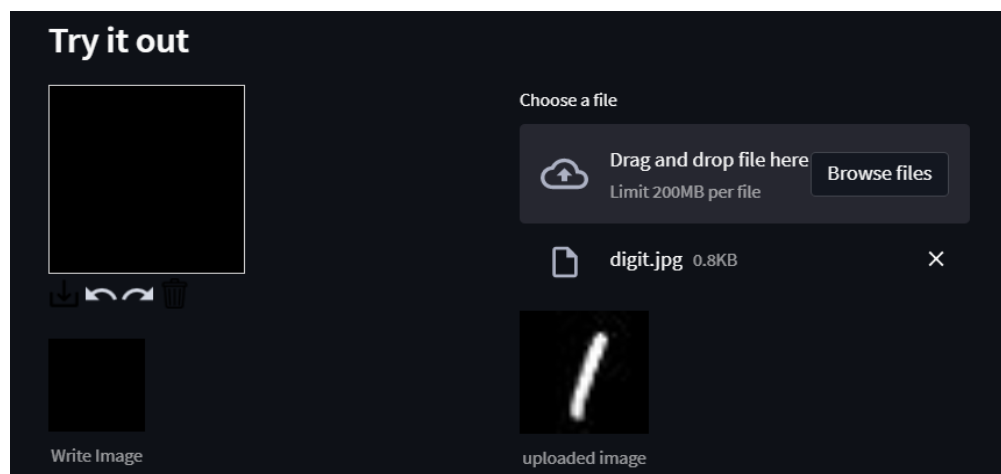


图 4-9 streamlit 实现图片输入模块

最后，在预测结果显示模块中，我们添加一个按钮，供用户点击以期获得两种模型的预测结果。当用户按下按钮时，前端所做的工作应当是：

- (1) 将待预测的图片转变成字节流的形式。
- (2) 将图片的字节流作为参数发送给后端的接口。后端的接口应当接收图片的字节流进行处理，做出预测后返回两种不同模型的预测结果。
- (3) 接受预测结果，并使用 `plotly` 绘制一个柱形图，图中表示有不同模型的对各类的预测概率，以及对应的不确定的值。

当用户点击预测按钮时，应用程序的显示效果如图 4-10 所示。最终，网络应用程序的整体效果如图 4-11 所示。

以上，我们使用 `streamlit` 搭建了一个简易的证据深度学习的网络应用程序的前端界面。在接下来的小节中，我们将会使用 `fastapi` 搭建我们的后端预测接口。

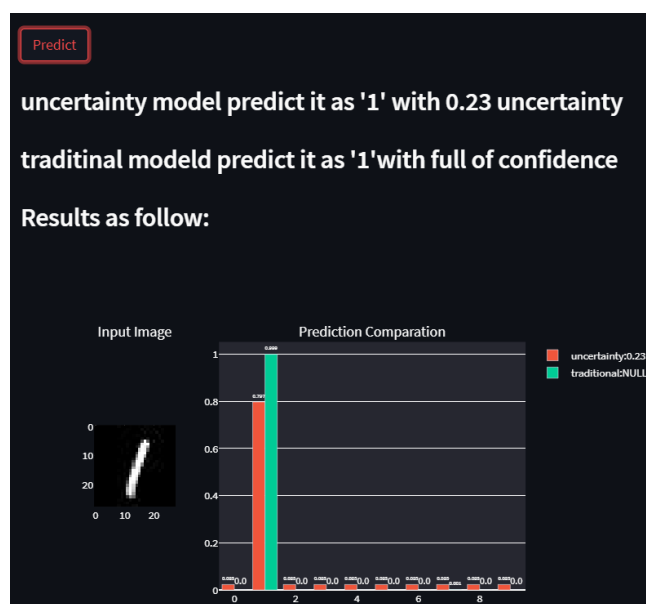


图 4-10 应用程序显示模型预测接口

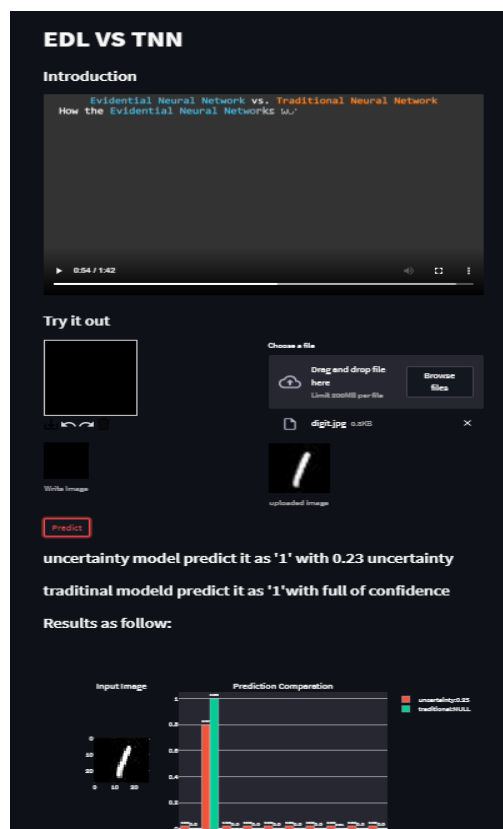


图 4-11 网络应用程序总览

§ 4.3 使用 fastapi 开发简易 API

本节介绍了什么是 Fastapi, Fastapi 开发简易接口的流程, 最后介绍如何使用 Fastapi 搭建我们的证据深度学习预测接口。

§ 4.3.1 Fastapi 概述

首先, 什么是 API? API 全称为 application programming interface 即应用程序接口。API 是一种计算接口, 它定义了:

- (1) 软件中介的交互
- (2) 请求的种类
- (3) 请求的方式
- (4) 交换数据的格式
- (5) 遵循的协议

综上, API 是允许两个软件组件使用一组定义和协议相互通信的机制。如我们的手机天气应用程序与气象局的软件系统“对话”, 获得每天的最新天气信息。而 FastAPI 是专注于构建 API 的, 一个现代的、快速的高性能 Web 框架。Fastapi 的特点有:

- (1) 快速：性能比肩 NodeJS 和 GO，是最快的 Python web 框架之一。
- (2) 简单：易于学习和开发。
- (3) 健壮：生产可用级别的代码，自动生成用于调试的交互式开发文档。

FastApi 的安装简单，安装过程如图 4-12 所示，另外除了 fastapi 外，我们仍然需要一个 ASGI 服务器来启动我们的 fastapi 程序代码，在本次搭建，我们选取 uvicorn 作为启动代码的 ASGI 服务器。

```
$ pip install fastapi
---> 100%

$ pip install "uvicorn[standard]"
---> 100%
```

图 4-12 fastapi 安装

§ 4.3.2 Fastapi 开发简易接口

我们以一个简单的 hello world 程序为例来介绍 Fastapi 的开发流程。

首先，在默认路径即主路径中，我们简单地返回一个 json 格式的数据来向使用者问好，在 hello 路径下，我们提供了一个定制化服务，读取用户输入的名字，返回个性化的问好数据。使用 Fastapi 实现的具体代码如图 4-13 左图所示，使用 uvicorn 启动 Fastapi 代码的命令行语句如图 4-13 右图所示，可以看到，使用 Fastapi，简单几行代码就能实现 api 接口的开发和部署。

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"Hello": "World"}

@app.get("/hello/{name}")
def read_item(name: str):
    return {"hello": name}
```

```
uvicorn hello:app --reload
Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
Started reloader process [16736] using watchgod
Started server process [19572]
Waiting for application startup.
Application startup complete.
```

图 4-13 Fast 代码示例与启动

启动成功后，我们根据 ASGI 服务器搭建代码的地址来访问我们的 API 接口，当我们向默认路径发送请求时，服务器只给我们返回了默认的问好语句如图 4-14

左图所示,当我们向 `hello` 路径发送请求并且另外输入一个名字后,服务器返回了个性化的问好语句,如图 4-14 右图所示。

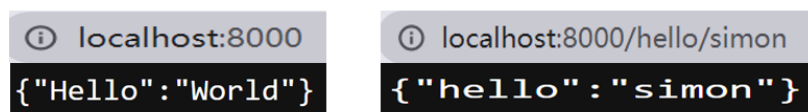


图 4-14 访问接口示例

§ 4.3.3 Fastapi 开发证据深度学习预测 API

因为本次搭建的前端应用程序仅实现了供用户使用两种模型对同一张图片的预测功能,因此,在后端程序中,我们只需要构建一个预测接口供前端使用,如若后续需要添加新的接口,后端只需要根据两者协商的数据格式和协议进行开发,不需要关注前端的实现,再一次体现了前后端分离架构的优势与高效。

根据前端开发所作出的预设,我们对预测接口有以下要求:

- (1) 预测接口在 `/predict` 路径下,从参数 `file` 提取图片的字节流信息。
- (2) 接口接收完所有字节流数据后,对字节流进行解码。
- (3) 接口调用前期训练好的两种模型对该图片进行预测,并将结果以 `json` 格式封装返回。每个模型对图片的预测应该有:

- (1) 模型对图片的预测结果
- (2) 模型对各类预测的概率
- (3) 不确定度(传统深度学习为 1)

在接口完成上述规定动作后,使用 `uvicorn` 对代码进行部署(如 4.3.2 示例所示),我们就可以通过访问 `/docs` 路径进入交互式调试文档进行调试。对接口代码进行发布后,进入到的交互式调试文档如图 4-15 所示。

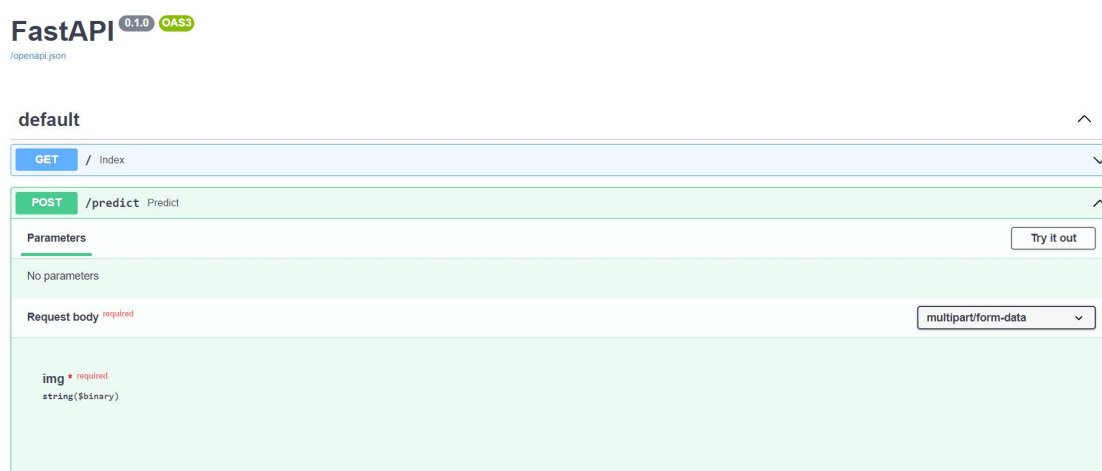


图 4-15 fastapi 交互式开发文档

点击 Try it out 即可对对应接口进行调试, 上传待预测的图片后, 接口返回的数据在 code 200(请求成功代码)对应的响应体中, 如图 4-16 所示。

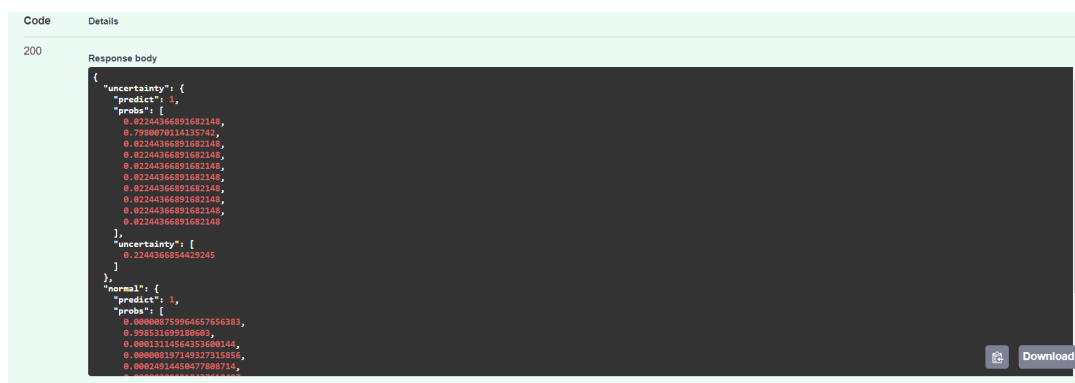


图 4-16 结果放回示例

可见, 这种交互式调试文档极大方便了接口的调试工作, 只要前端和后端对请求和返回的数据格式作出规定, 后端可直接通过 fastapi 生成的交互式调试文档进行调试, 无需理会数据是否成功达到, 前端是否成功处理等其他因素, 大大提高了两端开发的效率。

§ 4.4 使用 Docker 进行程序部署

本节介绍了什么是 docker, 镜像与容器的概念, 如何使用 Dockerfile 创建容器镜像, 用 Docker 部署前一节搭建的网络应用程序, 最后使用 docker-compose 一键部署程序。

§ 4.4.1 Docker 简介

Docker 是一种软件平台, 能够快速地建立, 测试和部署应用程序。Docker 将软件封装到名为容器的标准化单位中, 其中包含程序库, 必要系统工具, 程序代码以及运行时(run time)等执行软件所必须的项目。

Docker 是一种虚拟技术, 其与虚拟机相似, 但又有所区别, Docker 是操作系统层的虚拟化, 而虚拟机是硬件层的虚拟化, Docker 与虚拟机的区别如图 4-17 所示。

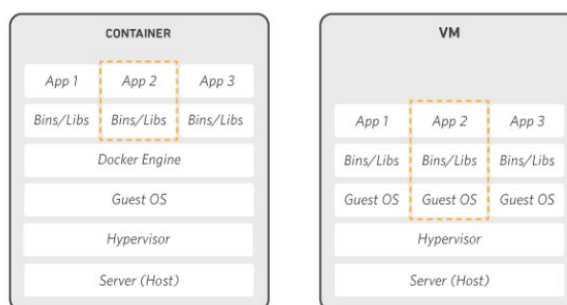


图 4-17 Docker 与虚拟机的区别

使用 Docker，我们能更快速地发布代码，实现应用程序操作的标准化，迁移代码，并提高资源利用率来节省开支。如前文提到，Docker 是操作系统层级的虚拟化，一个容器即是一个进程和运行此进程所需要的资源的集合。Docker 容器的运行是无视基础设施的(Infrastructure)，即我们能在一个 Windows 操作系统运行一个 Ubuntu20.04 的容器，容器内的资源是 Docker Engine 虚拟化 Windows 操作系统的资源获得的。

目前，Docker 在运行分布式微服务架构、使用标准化的持续集成和交付管道部署代码(CI/CD)、构建高度可拓展的数据处理系统，云原生服务等方面都作为一种主流技术被广泛使用。

§ 4.4.2 Docker 中的容器和镜像

Docker 中的容器(container)是程序运行的标准化单元，程序代码在容器中使用 Docker Engine 虚拟化的资源运行，容器与操作系统是隔离的，每个容器有对应的容器名和 ID，Docker Engine 通过对应命令对容器进行管理。Docker 能够为同一个程序创建多个容器，且容器间是相互隔离的，这样的特性为应用程序的建立，测试与部署多维度的调试提供了可利条件。

Docker 中的镜像(images)是一种模板文件，与虚拟机的镜像类似，Docker 通过镜像创建对应的容器，容器是镜像的实例化，通过镜像可以创建多个相同应用程序的容器，这样便利性与云原生中的弹性伸缩思想类似，这也是 Docker 在云原生服务中被广泛使用的原因。

而镜像仓库(Registry)则是存储容器镜像的仓库，Docker 有一个官方的镜像仓库 Docker registry，开发者可在其上面拉取其他服务提供者发布的镜像并直接使用，或者发布自己的镜像版本供他人使用。如我们可以直接拉取 ubuntu 提供的官方镜像，创建对应的 Ubuntu 容器即可直接在容器内开发自己的应用程序。

容器可以通过镜像创建，而镜像也可以通过容器生成对应的模板文件。容器、镜像与镜像仓库的关系如图 4-18 所示。

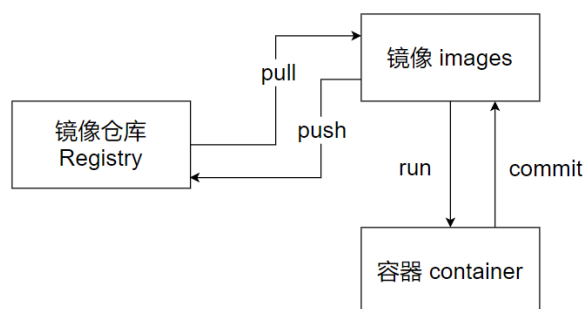
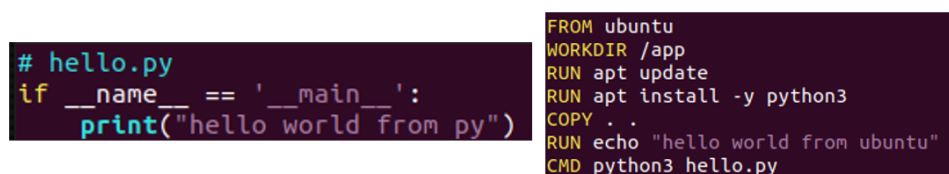


图 4-18 容器、镜像与镜像仓库关系

§ 4. 4. 3 Dockerfile 创建容器镜像

Dockerfile 是构建容器镜像的文本文件。Dockerfile 中的内容由一条条指令组成，每一条指令构建容器中的一层。以一个简单的 hello world 程序作为示例，我们创建一个目录，目录下的内容由一个打印 hello world 的 python 文件(如图 4-19 左图所示)以及一个 Dockerfile(如图 4-19 右图所示)组成。



```
# hello.py
if __name__ == '__main__':
    print("hello world from py")
```

```
FROM ubuntu
WORKDIR /app
RUN apt update
RUN apt install -y python3
COPY . .
RUN echo "hello world from ubuntu"
CMD python3 hello.py
```

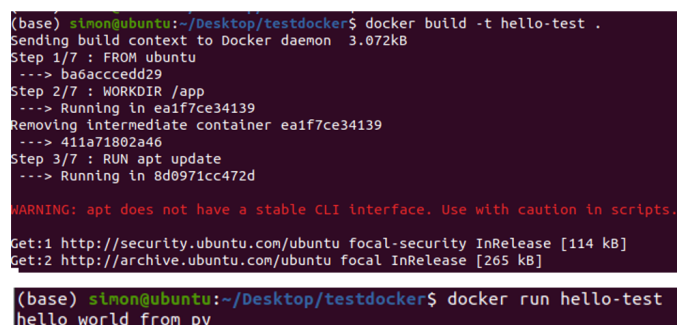
图 4-19 Dockerfile 示例

Dockerfile 中各命令解释说明如下：

- (1) FROM: 容器的基本镜像，本次示例使用的是 ubuntu 的官方镜像。
- (2) WORKDIR: 容器运行的工作目录。
- (3) RUN: 容器执行语句，在本 Dockerfile 中，我们分别执行了 ubuntu 的库管理工具 apt 的更新与安装 python3，最后打印输出 hello world 语句。
- (4) COPY: 复制当前目录到容器的工作目录中。即我们将编写的 hello.py 文件复制到容器的/app 目录下
- (5) CMD: 容器启动时执行的命令。即容器启动时运行 hello.py 文件

准备好上述两个文件后，我们执行 docker build 命令，该命令通过 Dockerfile 文件生成对应的镜像文件，如图 4-20 上图所示，可以看到 docker 通过 Dockerfile 生成镜像文件时，是一层一层地执行文件中规定的操作。镜像文件执行完后，我们通过 docker run 命令运行容器，如图 4-20 下图所示，可以看到，容器启动时产生了一条问候语句，该问候语句是通过 python 文件打印的。

以上是使用 Dockerfile 创建容器镜像并运行的实例，下面我们将使用 Dockerfile 生成应用程序的前后端容器镜像。



```
(base) simon@ubuntu:~/Desktop/testdocker$ docker build -t hello-test .
Sending build context to Docker daemon 3.072kB
Step 1/7 : FROM ubuntu
--> ba6accdd29
Step 2/7 : WORKDIR /app
--> Running in ea1f7ce34139
Removing intermediate container ea1f7ce34139
--> 411a71802a46
Step 3/7 : RUN apt update
--> Running in 8d0971cc472d

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Get:2 http://archive.ubuntu.com/ubuntu focal InRelease [265 kB]

(base) simon@ubuntu:~/Desktop/testdocker$ docker run hello-test
hello world from py
```

图 4-20 docker 创建容器镜像并启动容器示例

§ 4.4.4 使用 Docker 部署前后端程序

如前文提及的 Dockerfile 创建容器镜像，前后端程序的 Dockerfile 应该满足以下条件：

- (1) 基于 ubuntu 镜像
- (2) 安装 python
- (3) 安装对应的 python 库(使用 requirements.txt 存储要安装的库)
- (4) 将当前目录下的文件拷贝到容器中
- (5) 容器启动时，启动对应程序

前端程序对应的 Dockerfile 如图 4-21 所示，其中 EXPOSE 命令是开放容器的 8501 端口以实现与外界通信。

```
FROM ubuntu

RUN apt-get -y update
RUN apt-get -y install python3 python3-pip

WORKDIR /app

RUN pip3 install -r requirements.txt \
-i http://mirrors.aliyun.com/pypi/simple --trusted-host mirrors.aliyun.com
COPY . .

EXPOSE 8501

CMD streamlit run frontend.py
```

图 4-21 前端程序 Dockerfile

在容器的创建时，我们需要将容器的 8501 端口与主机的 8501 端口进行绑定，这样对外界来说，主机的 8501 端口就是提供前端服务的端口，而任何对主机 8501 端口发送的请求都会直接转发给容器的 8501 端口，而在容器中 8501 端口就是提供前端服务的端口，其自然会把所请求的资源发送回去，这样外界使用者就能得到容器所提供的服务。为了实现绑定，在创建容器时我们需要添加 -p 参数以实现对容器的绑定，如 -p 8501:8501 左侧是主机的端口，右侧是容器的端口，容器的端口必须对应提供服务的端口，而主机的端口可以自由调整。创建容器界面如图 4-22 上图所示，创建完成后，当我们访问本地的 8501 端口，我们依然会获得容器 8501 端口提供的前端服务，如图 4-22 下图图所示。

可以发现，使用 Dockerfile 通过预命令指导容器工作，能够让容器运行时直接部署程序而不用人为的操作。然而当要提供服务的容器增多，一个个部署容器的过程是相对繁琐的，接下来我们将会介绍如何使用 docker-compose 容器管理工具来实现程序的一键部署。

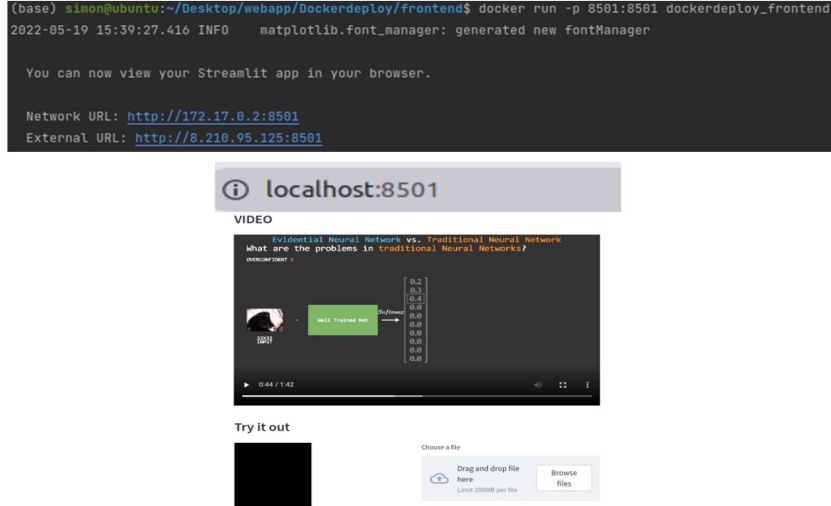


图 4-22 容器的创建与访问

§ 4.4.5 使用 docker-compose 一键部署应用程序

Docker compose 是一个可用于定义和创建多容器的应用程序。Docker Compose 使用 yaml 文件来配置各应用程序提供的服务，定义完成后，只需要通过一个简单的命令 `docker-compose up` 就能使先集群的部署。

不同于单独创建对应的容器，使用 Docker compose 配置的各容器服务属于同一个 compose 为该服务群创建的 docker 网络中，各容器间的通信可以通过容器名直接访问。如假设前端容器名为 `frontend`，后端容器名为 `backend`，那么前端就可以直接把 `backend` 作为目标地址，找到对应的端口以访问后端程序提供的对应服务。这种服务集群内各容器或服务之间相互认识，内部访问的特点保证了后端的安全性，防止后端的暴露导致数据泄露等安全问题。具体如图 4-23 所示。

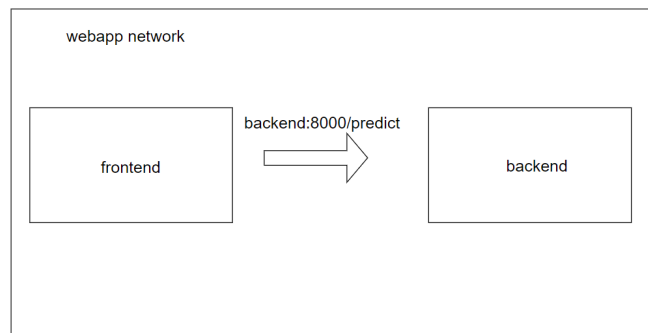


图 4-23 compose 服务集群的通信

为使用 docker compose 配置我们的网络应用程序所配置的 yaml 文件如图 4-24 所示。其中：

- (1) 各应用程序被视为一个服务，通过定义每一个服务的参数来管理。
- (2) 每一个服务需要指明该服务使用的是哪一个镜像，或是创建该服务的

Dockerfile 的位置,如图 4-24 所示,通过 build 的配置,定义了创建该服务的 Dockerfile 所在的目录位置。

(3) 每个服务可以通过 ports 定义主机与服务容器的端口绑定。

(4) 在前端的服务配置中,我们还定义了一个环境变量-BACKEND_URL,该变量对应后端服务的容器名以供前端找到后端的地址,而在前端代码中可直接通过访问对应环境变量获得目标地址。这种通过环境变量修改访问地址的方式使得服务在不同的生产环境中的调试和部署更加地灵活。

配置好 yaml 文件后,使用 docker-compose up 即可一键部署服务集群, Docker Compose 会根据每个服务所预设的镜像或根据对应目录的 Dockerfile 创建容器镜像,并根据设定的参数创建对应的集群,集群创建完后,可直接访问主机 8501 端口访问我们的前端证据深度学习预测服务。

```
version: "3.6"
services:
  backend:
    build: ./backend
    container_name: backend
    ports:
      - "8000:8000"

  frontend:
    build: ./frontend
    ports:
      - "8501:8501"
    environment:
      - BACKEND_URL=backend
```

图 4-24 docker-compose 配置服务集群 yaml 文件示例

第5章 总结与展望

本节从证据深度学习可视化动画和证据深度学习的网络应用程序的应用场景和市场场景入手，分析器市场竞争力和开发潜力。随后，将讨论其不足之处与可优化措施，提出日后相关改进措施。

§ 5.1 本文总结

本文主要从传统深度学习存在的问题引出了证据深度学习的运行原理，并从可视化角度出发，制作了一个证据深度学习的可视化算法介绍动画。为让学习者更加深入地体验证据深度学习的特点，本文还制作了一个简单的证据深度学习网络应用程序，供使用者自行参与到两种模型的预测中，让使用者对证据深度学习有一个更加直观的了解。

§ 5.1.1 本文的主要工作

本文主要研究了证据深度学习原理的可视化过程，工作内容有以下六个方面：

（1）证据深度学习原理的研究学习：对 Sensoy^[7]，袁冰^[8]等人使用证据深度学习计算模型预测的不确定度方法进行学习研究，学习具体的计算过程和证据深度学习与传统深度学习的差异。

（2）数学可视化工作 Manim 的学习：了解 Manim 的主体架构与各组件和方法的使用，制作介绍证据深度学习的可视化动画。

（3）对证据深度学习开源代码的研究：学习模型代码的主体框架和执行流程，尝试自己复现相应的模块。制作模型预测的接口，其可根据不同模型返回对应的模型预测。

（4）对网络应用程序开发工具 streamlit 的学习：了解 streamlit 运行和开发的逻辑，使用 streamlit 搭建证据深度学习网络应用程序，该程序包括介绍视频播放，图片输入以及模型预测结果展示等功能模块，供学习者深度体验证据深度学习与传统深度学习的差异。

（5）对后端 api 接口开发工具 fastapi 的学习：学习 fastapi 快速搭建后端 api 接口，结合（3）中提及的模型预测接口，搭建自己的模型预测 api，其接收图片的字节流信息，返回两种模型对接收图片的预测结果。

（6）使用 docker，docker-compose 部署应用程序：为实现（4）和（5）提及的前后端程序的简易部署，使用 docker 构建不同的容器，并通过 docker-compose 实现程序的一键部署，方便项目的迁移与发布，使程序能够部署在任何架构的服务器中，无需考虑基础设施的限制。

§ 5.1.2 本文的主要创新点

本文的创新点主要有以下五点：

(1) 本文介绍的证据深度学习所做出的模型预测比传统深度学习的预测结果更加的确定；

(2) 本文制作的证据深度学习的介绍动画，相较于传统的教学模式更加直观，通过图形变化，动态地展示了传统深度学习与证据深度学习的运行过程。

(3) 本文所构建的网络应用程序采用前后端分离架构，使得开发效率更高，各端开发更加专注。

(4) 本文最终采用 `docker` 容器技术部署网络应用程序，使得程序能够在任何基础架构上运行，让项目的迁移，开发优化和发布更加方便快捷。

(5) 本文根据开源证据深度学习项目构建自定义的模型预测接口，让开发者无视证据深度学习的内部架构，仅使用预测接口就能实现模型的预测。

§ 5.2 展望

本节从证据深度学习可视化动画和证据深度学习的网络应用程序的应用场景和市场场景入手，分析器市场竞争力和开发潜力。随后，将讨论其不足之处与可优化措施，提出日后相关改进措施。

§ 5.2.1 本文的应用场景与市场前景

本文所制作的证据深度学习的动画与网络应用程序有以下三个应用场景：

(1) 教学场景。证据理论领域教学教师可使用本动画向学生们介绍证据深度学习方法的相关原理。

(2) 学习场景。学生或有兴趣的学习者可以通过动画辅助理解。

(3) 实践场景。学习者可以直接通过本文开发的网络应用程序进行自定义的模型预测体验。

本文的市场价值和市场场景体现在以下几点：

(1) 介绍证据深度学习。证据理论在机器学习领域的应用在近几年逐渐发展起来，通过动画介绍这一基于证据理论的深度学习方法，能让更多有深度学习基础的学习者快速了解证据理论，鼓励更多地人学习了解证据理论。

(2) 本文开发的证据深度学习网络应用程序，能够让学习者通过动画了解相关的理论知识外，还能够让学习者参与模型的预测过程，更加深度感受证据深度学习与传统深度学习的差别。

(3) 本文所制作的动画和网络应用程序还有很大的开发空间，在适当的优化后，能够满足更多深度学习和证据理论相关领域的教学任务和应用场景。

§ 5.2.2 本文的不足及改进措施

虽然本文基本实现了证据深度学习的可视化与相关的网络应用程序开发，但仍存在改进的空间：

(1) 可视化开发中动画的可拓展性和修改性不高。在实际教学任务中，教师可能会根据不同深度学习模型可视化模型内部结构，本文的深度学习模型选取的是 LeNet-5^[4]，在模型的内部细节可视化上，代码的可拓展性仅局限于相同架构下模型结构的调整，本文没有整理出一个更为统一的模型可视化解决方案。因此，在可视化动画的制作中，还需要对模型内部细节展示上抽象出一个更为统一的模型展示类，开发者通过创建对应的类，修改内部细节，可直接动态展示模型的内部结构。

(2) 在证据理论可视化过程中，没有深度体现证据深度学习在损失函数层面上的动态展现，仅仅列出了相应的计算公式。因此，在证据深度学习的可视化过程中，最好能够构建坐标系，动态展示每个样本经过证据深度学习模型处理后对总体损失函数的影响。

(3) 学习者使用网络应用程序的功能单一。除了基本的播放和预测，没有相应的模块供开发者使用自定义的训练集和参数调节训练出对应的证据深度学习模型。其次，学习者能够使用的模型结构单一，没有其他的经过证据调节后的经典模型可使用。因此，可以在本文制作的应用程序的基础上，训练出更多的模型，添加供开发者选择特定的预训练模型进行预测的接口，其次，除了预测外，本文还可以提供给学习者通过自定义的模型结构和参数来对自定义的训练集进行模型训练，并且根据训练后的模型作出相应的预测尝试。

基于上述三点，在日后的研究中应当重视开发过程中的可拓展性，以及考虑更多维度的用户需求，尽可能地让使用者在观看动画或使用应用程序时有一个更加丰富的体验。

致谢

首先，我要谢谢杜水淼老师和陈伦德老师在我的毕设工作中给予我的指导和鼓励，正是有他们的帮助和支持，我才能完成我的毕业设计以及毕业论文的撰写工作。

其次，我要感谢我的好朋友张宇轩，在我的大学生活中，他一直支持我，帮助我，还要感谢跟我一起学习的同学，一起参加比赛的队友，教授我知识的所有老师们。

另外，我还要感谢我的女朋友谢昕还有我的家人们，谢谢他们在我最需要的时候给予我支持和陪伴，让我有勇气面对任何困难和挑战。

最后，我要感谢上海大学，感谢中欧工程技术学院。感谢上海大学的一草一木，感谢上海大学提供的平台，感谢上海大学提供给我的所有机会。我为作为上大人而自豪。

作为即将离校的上大人，行文至此，临表涕零，不知所言，叩首再拜。

参考文献

- [1]. McCulloch W S, Pitts W. A logical calculus of the ideas immanent in nervous activity[J]. The bulletin of mathematical biophysics, 1943, 5(4): 115-133.
- [2]. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain[J]. Psychological review, 1958, 65(6): 386.
- [3]. Rumelhart D E, Hinton G E, Williams R J. Learning representations by back-propagating errors[J]. nature, 1986, 323(6088): 533-536.
- [4]. LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.
- [5]. Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks[J]. Advances in neural information processing systems, 2012, 25.
- [6]. Shafer G. Dempster-shafer theory[J]. Encyclopedia of artificial intelligence, 1992, 1: 330-331.
- [7]. Sensoy M, Kaplan L, Kandemir M. Evidential deep learning to quantify classification uncertainty[J]. Advances in Neural Information Processing Systems, 2018, 31.
- [8]. Yuan B, Yue X, Lv Y, et al. Evidential deep neural networks for uncertain data classification[C] International Conference on Knowledge Science, Engineering and Management. Springer, Cham, 2020: 427-437.
- [9]. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.
- [10]. Jøsang A. Subjective logic[M]. Cham: Springer, 2016.
- [11]. Kendall A, Gal Y. What uncertainties do we need in bayesian deep learning for computer vision?[J]. Advances in neural information processing systems, 2017, 30.
- [12]. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The journal of machine learning research, 2014, 15(1): 1929-1958.
- [13]. Johnson N L, Kotz S, Balakrishnan N. Continuous univariate distributions, volume 2[M]. John wiley & sons, 1995.
- [14]. Dempster A P. Upper and lower probabilities induced by a multivalued mapping[M] Classic works of the Dempster-Shafer theory of belief functions. Springer, Berlin, Heidelberg, 2008: 57-72.
- [15]. Denoeux T. Maximum likelihood estimation from uncertain data in the belief function framework[J]. IEEE Transactions on knowledge and data engineering, 2011, 25(1): 119-130.
- [16]. Simard P Y, Steinkraus D, Platt J C. Best practices for convolutional neural networks applied to visual document analysis[C] Icdar. 2003, 3(2003).
- [17]. Cireşan D C, Meier U, Masci J, et al. High-performance neural networks for visual object classification[J]. arXiv preprint arXiv:1102.0183, 2011.
- [18]. 苏赋, 吕沁, 罗仁泽. 基于深度学习的图像分类研究综述[J]. 电信科学, 2019, 35(11): 58-74.
- [19]. 倪明, 单渊达. 证据理论及其应用[J]. 电力系统自动化, 1996, 20(3): 76-80.

- [20]. 李弼程, 王波, 魏俊, 等. 一种有效的证据理论合成公式[J]. 数据采集与处理, 2002, 17(1): 33-36.
- [21]. Fei-Fei L, Fergus R, Perona P. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories[C]//2004 conference on computer vision and pattern recognition workshop. IEEE, 2004: 178-178.
- [22]. Lee H, Grosse R, Ranganath R, et al. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations[C]//Proceedings of the 26th annual international conference on machine learning. 2009: 609-616.