第 1 章

深度学习与计算机视觉

随着人工智能的快速发展，深度学习已经成为计算机视觉领域的重要支撑技术。深度学习通过模拟人脑神经网络的运作方式，使得计算机能够从大量的图像数据中自动学习特征表示和目标检测，从而实现更加准确和高效的图像处理和分析。

计算机视觉是一门研究如何让计算机从图像或视频中获取信息、理解内容并做出决策的科学。它涉及图像处理、模式识别、机器学习等多个领域，并在工业、医疗、安防等领域有着广泛的应用。

深度学习在计算机视觉中的应用，使得我们可以从大量的图像数据中提取出更加抽象和高级的特征表示，从而提高图像分类、目标检测、图像分割等任务的性能和精度。同时，深度学习还可以通过注意力机制、扩散模型等技术，实现图像生成、目标识别与分割等应用，进一步扩展了计算机视觉的应用范围。

可以想象到，在人工智能的浩瀚星空中，深度学习犹如一颗璀璨的明珠，引领着计算机视觉领域的技术革新。通过模拟人脑神经元之间的微妙连接，深度学习可以构建出复杂而精巧的神经网络模型，赋予机器解读图像、视频等视觉数据的能力。计算机视觉作为人工智能的重要支柱，致力于让机器理解世界，捕捉有价值的信息，进而做出明智的决策。深度学习在计算机视觉中的广泛应用，无疑为该领域注入了一剂强心针，催生了许多突破性的成果。

1.1 深度学习的历史与发展

深度学习起源于人工神经网络的研究，历经数十年的演变和发展，已经在人工智能领域展现出强大的实力。它模拟了人脑神经元之间的连接，通过构建复杂的神经网络模型，使得机器能够从海量数据中自动学习特征表示和目标检测。随着计算机硬件和算法的不断进步，深度学习在图像识别、语音识别、自然语言处理等领域取得了显著的成果。如今，深度学习已经成为计算机视觉领域的重要支撑技术，为图像分类、目标检测、图像分割等任务提供了强大的技术支持。

1.1.1　深度学习的起源

深度学习这一术语的提出，可以追溯到 2006 年。然而，其真正的起源和发展历程要远远早于这个时间点。深度学习起源于人工神经网络的研究，这一探索可以追溯到 20 世纪 80 年代。在那个时期，科学家们受到人脑神经元之间复杂连接的启发，开始尝试使用简单的神经元模型来模拟人脑的学习过程。这些早期的神经网络模型采用了前向传播算法，通过调整神经元之间的连接权重来进行学习。然而，由于当时缺乏足够的数据和计算资源，以及存在梯度消失等问题，这些模型在实际应用中表现并不理想。

随着计算机硬件和算法的不断进步，深度学习在 21 世纪初重新焕发了生机。新的基于深度学习的深度信念网络（DBN）和反向传播算法的提出，使得神经网络可以训练更深层次的模型。这一突破性的进展为深度学习的兴起奠定了坚实的基础。反向传播算法的核心思想是通过计算损失函数对模型参数的梯度，并使用梯度下降等方法来更新模型参数，从而使模型在训练数据上的表现不断优化。这一算法的出现解决了早期神经网络训练过程中的梯度消失问题，使得训练更深层次的神经网络成为可能。

同时，随着大数据时代的到来，人们可以获取到更多的图像、文本和语音数据，为深度学习的发展提供了丰富的数据资源。大数据的涌现使得深度学习模型可以在大规模数据集上进行训练，从而学习到更为复杂和抽象的特征表示。此外，GPU 等高性能计算设备的出现，也加速了深度学习模型的训练过程。这些技术的突破为深度学习的快速发展奠定了坚实的基础。

深度学习发展历程如图 1-1 所示。

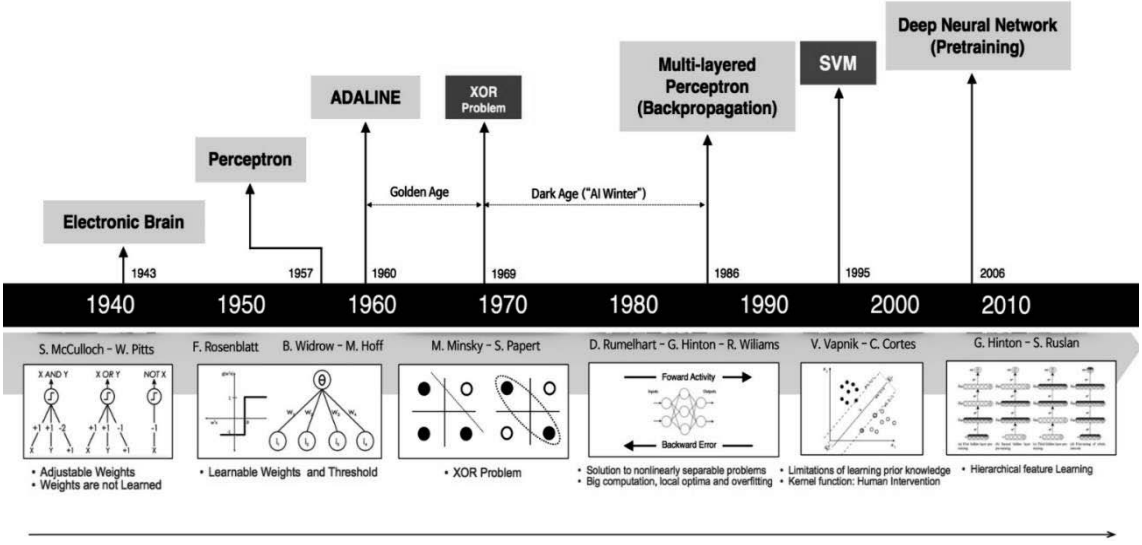

图 1-1　深度学习发展历程

深度学习在图像识别、语音识别、自然语言处理等领域取得了显著的成果。在图像识别领域，深度学习通过卷积神经网络（Convolutional Neural Networks，CNN）等模型，自动学习图像中的特征表示，大幅度提升了图像分类和目标检测的精度。特别是随着人们对深度学习研究的进一步加强，注意力机制受到了重视并被广泛使用，极大地提高了计算机视觉领域的识别率和准确率，取得了显著成果。

1.1.2　深度学习的发展脉络

深度学习的发展可谓是一波三折，经历了多次的兴衰。从早期的神经网络模型，到深度信念网络和反向传播算法的提出，再到大数据和 GPU 等技术的突破，深度学习逐渐发展成为人工智能领域的重要支撑技术之一。

在早期的神经网络研究中，科学家们主要关注如何构建和训练简单的神经网络模型，以模拟人脑的学习过程。然而，由于当时缺乏足够的数据和计算资源，以及存在梯度消失等问题，这些模型在实际应用中表现并不理想。这一时期的研究主要集中在神经网络的基本结构和算法上，为后来的深度学习研究奠定了基础。

随着计算机硬件和算法的不断进步，深度学习在 21 世纪初重新焕发了生机。在反向传播算法的加持下，神经网络可以训练更深层次的模型。这一突破性的进展为深度学习的兴起奠定了坚实的基础。反向传播算法的出现解决了早期神经网络训练过程中的梯度消失问题，使得训练更深层次的神经网络成为可能。

在深度学习的发展过程中，卷积神经网络的出现可谓是浓墨重彩的一笔。卷积神经网络于 20 世纪 90 年代提出，它通过卷积操作和池化操作来提取图像中的特征，并使用全连接层进行分类。卷积神经网络的出现使得深度学习在图像分类、目标检测、图像分割等计算机视觉任务中取得了显著的成果。

如今，新的深度学习模式“注意力机制”被提出，并且一经提出后立刻成为深度学习计算机视觉研究的核心和重要方向。注意力机制最早在自然语言处理领域被提出，后被引入计算机视觉领域。注意力机制通过对图像或视频序列中的特定区域或对象赋予更大的关注度，使得模型能够更为精准地捕捉关键信息，从而提高模型的性能。

除了计算机视觉领域外，注意力机制还在自然语言处理、语音识别、推荐系统等领域发挥着重要的作用。随着技术的不断进步和应用场景的不断拓展，相信注意力机制将会在未来的深度学习研究中发挥更为重要的作用。

1.1.3　为什么是 PyTorch 2.0

在深度学习和计算机视觉的交汇点上，PyTorch 以其独特的优势和强大的功能成为众多研究者和开发者的首选计算框架。特别是随着 PyTorch 2.0 的横空出世，借助于其优秀的性能和庞大的社区生态圈，使得 PyTorch 2.0 成为事实上的深度学习框架的唯一选择。PyTorch 2.0 Logo 如图 1-2 所示。

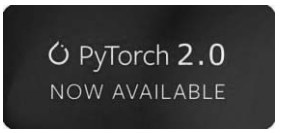

图 1-2　PyTorch 2.0 Logo

相对于其他深度学习框架，PyTorch 有着无可比拟的优点：

- 动态计算图：PyTorch动态计算图的设计，使得模型的开发和调试过程更加灵活和高效。与其他静态计算图框架相比，PyTorch允许在运行时动态构建计算图，从而提供了更大的灵活

性和可扩展性。这种设计使得研究者能够快速地迭代和改进模型，大大提高了研究效率。

- 易于使用和调试：PyTorch提供了简洁易用的API，使得模型的构建和训练过程变得轻松简单。同时，PyTorch还提供了丰富的调试工具，如梯度检查和可视化工具，帮助研究者快速发现和解决问题。这些工具使得研究者能够更加专注于模型的设计和优化，而不必花费过多的时间和精力在处理计算细节上。
- 社区支持和生态系统：PyTorch拥有庞大的社区支持和活跃的生态系统，这使得研究者能够轻松获取大量的开源模型和工具。PyTorch社区提供了丰富的预训练模型和算法实现，以及各种用于数据处理和可视化的工具。此外，PyTorch还与许多其他深度学习库和框架集成，如TensorFlow和ONNX，从而进一步扩大了其生态系统。
- 高性能和可扩展性：PyTorch在性能和可扩展性方面也具有显著优势。PyTorch支持多种硬件平台，包括CPU、GPU和TPU，并提供了高度优化的计算内核，以实现高效的张量运算。此外，PyTorch还支持分布式训练，使得研究者能够利用多个计算节点进行大规模模型的训练。
- 教育和研究资源：PyTorch在教育和研究领域也得到了广泛应用。许多大学和研究机构采用PyTorch作为深度学习教学的主要工具，同时也有大量的研究论文和项目使用PyTorch作为实验平台。这使得研究者能够更方便地获取相关的教育资源和研究成果，进一步推动了深度学习计算机视觉领域的发展。

可以看到，PyTorch 在深度学习计算机视觉领域具有显著的优势和广泛的应用前景。其动态计算图的设计、易于使用和调试的特性、社区支持和生态系统、高性能和可扩展性以及丰富的教育和研究资源，使得 PyTorch 成为深度学习计算机视觉领域的首选计算框架。

1.2　计算机视觉之路

计算机视觉（见图 1-3）是人工智能领域的一个重要分支，旨在让机器能够“看懂”世界。从早期的图像处理，到现代的深度学习，计算机视觉的发展经历了多次变革。早期的计算机视觉研究主要集中在图像处理的基本方法上，如滤波、边缘检测等。随着技术的发展，人们开始关注如何从图像中提取出更为抽象和高级的特征，这导致了特征工程的兴起。

然而，特征工程需要大量的手工设计和调整，效率较低。深度学习的出现，为计算机视觉的发展提供了新的思路和方法。通过自动学习图像中的特征，深度学习在图像分类、目标检测、图像分割等任务中取得了显著的成果。如今，计算机视觉已经广泛应用于安防、医疗、自动驾驶等领域，为人类社会的发展带来了深远的影响。

1.2.1　计算机视觉的基本概念

计算机视觉（见图 1-3）是一门研究如何让机器从图像或视频中获取信息、理解内容并做出决策的科学。它是人工智能领域的一个重要分支，涉及多个学科的知识，如数学、物理、信号处理、计算机科学等。

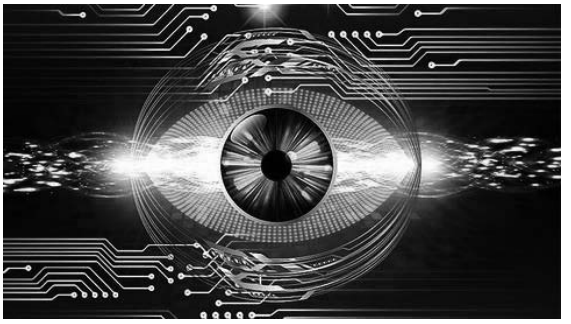

图 1-3　计算机视觉

计算机视觉的基本概念可以从以下几个方面来理解：

首先，计算机视觉的研究对象是图像和视频。图像是由像素组成的二维数组，每个像素包含颜色、亮度等信息。视频则是一系列连续的图像帧，通过播放这些帧可以呈现出动态的画面。计算机视觉的任务就是从这些图像和视频中提取出有用的信息，如物体、场景、动作等。

其次，计算机视觉的研究内容包括图像处理、图像分析和图像理解三个层次。图像处理关注于图像的预处理和基本特征的提取，如滤波、边缘检测等；图像分析则关注于从图像中提取出更为抽象和高级的特征，如物体识别、场景分析等；图像理解则关注于如何让机器真正“看懂”图像，如语义理解、情感分析等。

此外，计算机视觉的研究方法可以分为传统方法和深度学习方法两类。传统方法主要依赖于手工设计的特征提取算法，如 SIFT、HOG 等；深度学习方法则通过自动学习图像中的特征来提高模型的性能。近年来，随着深度学习技术的不断发展，计算机视觉的研究已经取得了显著的成果。

最后，计算机视觉的应用范围非常广泛。它可以应用于安防领域的人脸识别、行为分析，医疗领域的医学图像处理、病灶检测，自动驾驶领域的车辆检测、交通流分析等多个领域。同时，计算机视觉还可以与其他领域的技术相结合，如自然语言处理、语音识别等，推动了人工智能技术的不断发展。

1.2.2　计算机视觉深度学习的主要任务

计算机视觉的主要研究内容可以从以下几个方面来理解：

首先，图像处理是计算机视觉的基础。它关注于图像的预处理和基本特征的提取，如滤波、边缘检测、图像增强等。这些处理技术可以帮助我们去除图像中的噪声、改善图像的质量，并为后续的图像分析和理解奠定基础。

其次，图像分析是计算机视觉的核心。它关注于从图像中提取出更为抽象和高级的特征，如物体识别、场景分析、动作识别等。图像分析可以帮助我们理解图像中的内容，并为后续的决策和判断提供依据。

最后，图像理解是计算机视觉的最高层次。它关注于如何让机器真正“看懂”图像，如语义理解、情感分析等。图像理解可以帮助我们理解图像中的语义信息，并为后续的自然语言处理、语音识别等奠定基础。

除了主要研究内容外，计算机视觉还具有广泛的应用领域。以下是一些主要的应用领域：

- 安防领域：计算机视觉在安防领域的应用非常广泛，如人脸识别、行为分析、视频监控等。通过计算机视觉技术，我们可以实现自动化监控、智能预警等功能，从而提高安全管理的效率和准确性。
- 医疗领域：计算机视觉在医疗领域的应用也非常广泛，如医学图像处理、病灶检测、辅助诊断等。通过计算机视觉技术，我们可以实现自动化诊断、精准治疗等功能，从而提高医疗服务的水平和效率。
- 自动驾驶领域：计算机视觉在自动驾驶领域的应用也非常重要，如车辆检测、交通流分析、道路识别等。通过计算机视觉技术，我们可以实现自动化驾驶、智能交通等功能，从而提高交通运输的安全性和效率。
- 智能制造领域：计算机视觉在智能制造领域的应用也非常广泛，如产品质量检测、自动化装配、生产线监控等。通过计算机视觉技术，我们可以实现自动化生产、精准控制等功能，从而提高生产效率和质量。

总之，计算机视觉是一门涉及多个学科的综合性科学，其主要研究内容包括图像处理、图像分析和图像理解三个层次。同时，计算机视觉也具有广泛的应用领域，为人类社会的发展带来了深远的影响。随着深度学习技术的不断发展，计算机视觉的研究已经取得了显著的成果，并为人类社会的发展带来了更多的可能性。

1.2.3　计算机视觉中的深度学习方法

深度学习方法已经成为计算机视觉领域的主流技术，为图像分类、目标检测、图像分割等任务提供了强有力的支持。在计算机视觉中，常用的深度学习方法主要包括卷积神经网络（CNN）和注意力机制，如图 1-4 所示。

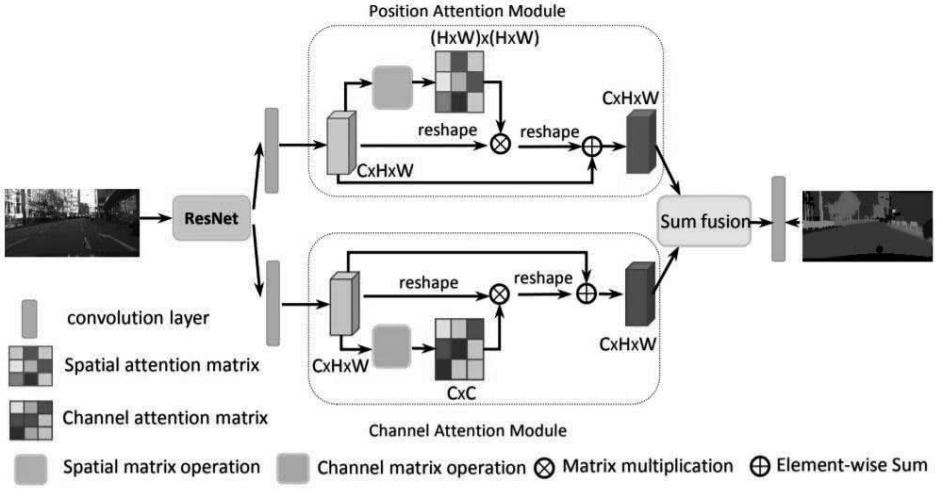

图 1-4　计算机视觉中的卷积与注意力模块

1. 卷积神经网络

卷积神经网络的基本结构包括卷积层、池化层和全连接层。卷积层负责在原始图像上滑动一

个卷积核，并通过卷积操作提取出图像中的特征。池化层则负责对卷积后的特征图进行下采样，以减少计算量和过拟合的风险。全连接层则用于将提取到的特征映射到最终的分类结果上。

在卷积神经网络中，卷积核的选择和设计是非常重要的。不同的卷积核可以提取出不同的特征，如边缘、纹理等。同时，卷积神经网络的深度和宽度也是影响模型性能的重要因素。通过增加网络的深度和宽度，可以提取到更为抽象和高级的特征，但也会增加模型的复杂度和计算成本。

2. 注意力机制

注意力机制通过对图像或视频序列中的特定区域或对象赋予更大的关注度，使得模型能够更为精准地捕捉关键信息，从而提高模型的性能。注意力机制可以分为空间注意力、通道注意力和混合注意力等多种类型。空间注意力关注图像中的空间位置信息，通过对不同位置赋予不同的权重来提高模型的表现；通道注意力则关注图像中的不同通道信息，通过对不同通道赋予不同的权重来优化模型的表现；混合注意力则是将空间注意力和通道注意力结合起来，共同优化模型的表现。

在计算机视觉领域中，注意力机制已经被广泛应用于图像分类、目标检测、图像分割等任务中。在图像分类任务中，注意力机制可以帮助模型更好地捕捉图像中的关键区域，从而提高分类的准确率；在目标检测任务中，注意力机制可以帮助模型更好地定位目标对象，从而提高检测的精度；在图像分割任务中，注意力机制可以帮助模型更好地分割出图像中的不同区域，从而提高分割的准确度。

1.2.4　深度学习在计算机视觉中的应用

深度学习在计算机视觉中的应用已经取得了显著的成果，为图像分类、目标检测、图像分割等任务提供了强有力的支持。以下是一些主要的计算机视觉任务及其深度学习方法的应用。

- 图像分类：图像分类是计算机视觉中最基本的任务之一，旨在将输入图像自动分类到预定义的类别中。深度学习方法已经成为图像分类的主流技术，其中最常用的是卷积神经网络。通过在大规模图像数据集上进行训练，卷积神经网络可以自动学习图像中的特征，并在图像分类任务中取得显著的成果。
- 目标检测：目标检测是计算机视觉中的一项重要任务，旨在从图像或视频序列中检测出感兴趣的目标对象，并确定其位置和大小。深度学习方法已经在目标检测任务中取得了显著的成果，其中最常用的是基于卷积神经网络的模型，如Faster R-CNN、YOLO等。这些模型可以通过自动学习图像中的特征来准确定位目标对象，并在目标检测任务中取得良好的性能。
- 图像分割：图像分割是计算机视觉中的一项重要任务，旨在将图像分割成不同的区域或对象。深度学习方法已经在图像分割任务中取得了显著的成果，其中最常用的是基于卷积神经网络的模型，如Unet、SwinUnet等。这些模型可以通过自动学习图像中的特征来精确分割出不同的区域或对象，并在图像分割任务中取得良好的性能。
- 人脸识别：人脸识别是计算机视觉中的一项重要任务，旨在从图像或视频序列中识别出特定的人脸。深度学习方法已经在人脸识别任务中取得了显著的成果，其中最常用的是基于卷积神经网络的模型。这些模型可以通过自动学习人脸的特征来精确识别出不同的人脸，并在人脸识别任务中取得良好的性能。

可以看到，深度学习已经成为计算机视觉领域的主流技术，为各种计算机视觉任务提供了强有力的支持。通过合理的数据预处理、模型训练和调参优化，可以实现深度学习方法在计算机视觉任务中的最佳性能。

1.3　本章小结

本章主要介绍了深度学习和计算机视觉的相关背景知识和主要内容。首先，介绍了深度学习的基本概念和发展历程，并讲解了为何选择 PyTorch 2.0，然后介绍了计算机视觉的基本概念和研究内容，以及计算机视觉的主要应用领域。此外，本章还介绍了卷积神经网络和注意力机制等深度学习方法在计算机视觉中的应用，强调了深度学习在计算机视觉中的重要性。

展望未来，我们有理由相信，随着技术的不断进步和创新，深度学习将在计算机视觉领域发挥更加重要的作用，将为计算机视觉领域的发展注入新的活力，为人类社会的发展带来更多的可能性。正如一幅宏伟的画卷在我们眼前展开，深度学习和计算机视觉的美妙交融将为我们揭示出更多的奥秘和可能性。

第 2 章

PyTorch 2.0 深度学习环境搭建

工欲善其事，必先利其器。上一章介绍了为何选择 PyTorch 2.0，从本章开始，我们将正式深入 PyTorch 2.0 的世界，揭示其强大的能力。

首先，对于任何一位想要构建深度学习应用程序或是将训练好的模型应用到具体项目的读者，都需要使用编程语言来实现设计意图。在本书中，将使用 Python 语言作为主要的开发语言。

Python 在深度学习领域中被广泛采用，这得益于许多第三方提供的集成了大量科学计算类库的 Python 标准安装包，其中最常用的便是 Miniconda。Python 是一种脚本语言，如果不使用 Miniconda，那么第三方库的安装可能会变得相当复杂，同时各个库之间的依赖性也很难得到妥善的处理。因此，为了简化安装过程并确保库之间的良好配合，我们推荐安装 Miniconda 来替代原生的 Python 语言安装。

PyTorch 是一个开源的机器学习库，基于 Torch 开发，主要用于自然语言处理和其他应用程序。它是一个以 Python 优先的深度学习框架，支持动态图和延迟执行，提供了 Python 接口。PyTorch 的设计思想注重代码的运行效率和灵活性，封装了许多用于实现神经网络的函数和类。PyTorch 既可以看作加入了 GPU 支持的 NumPy，同时也可以看作一个拥有自动求导功能的强大的深度神经网络。

在本章中，首先引导读者完成 Miniconda 的完整安装，然后完成 PyTorch 2.0 的安装，最后通过一个实践项目来帮助读者进一步熟悉 PyTorch 2.0。这个项目将生成可控的手写数字，通过这个项目，读者将能够初步体验到 PyTorch 2.0 的强大功能及其灵活性。

2.1 环境搭建 1：安装 Python

2.1.1 Miniconda 的下载与安装

1. 下载和安装Miniconda

在 Miniconda 官方网站打开下载页面，如图 2-1 所示。

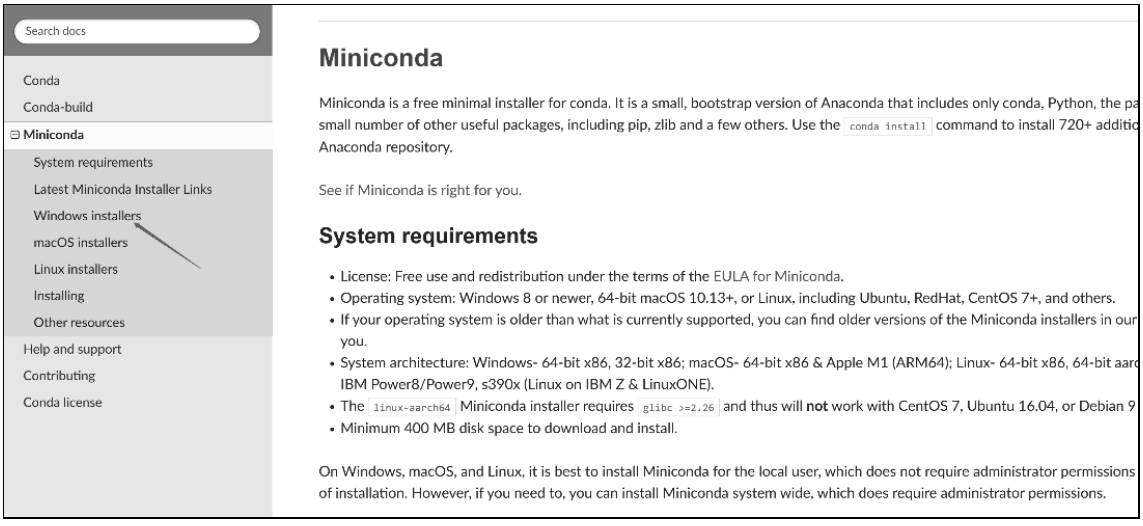

图 2-1　Miniconda 下载页面

读者可以根据不同的操作系统选择不同的 Miniconda 下载，目前提供的是最新集成了 Python 3.11 64-bit 版本的 Miniconda。如果读者使用的是以前的 Python 版本，例如 Python 3.10，也是完全可以的，笔者进行过测试，无论 3.11 还是 3.10 版本的 Python，都不影响 PyTorch 的使用。

这里笔者推荐使用的是 Windows Python 3.11 64-bit 的版本，相对于 3.10 版本，3.11 版本集成了目前最新 Python 的一些优化技术。当然，读者也可以根据自己的喜好或者具体计算机配置进行选择。集成 Python 3.11 版本的 Miniconda 可以在官方网站下载，打开后如图 2-2 所示。

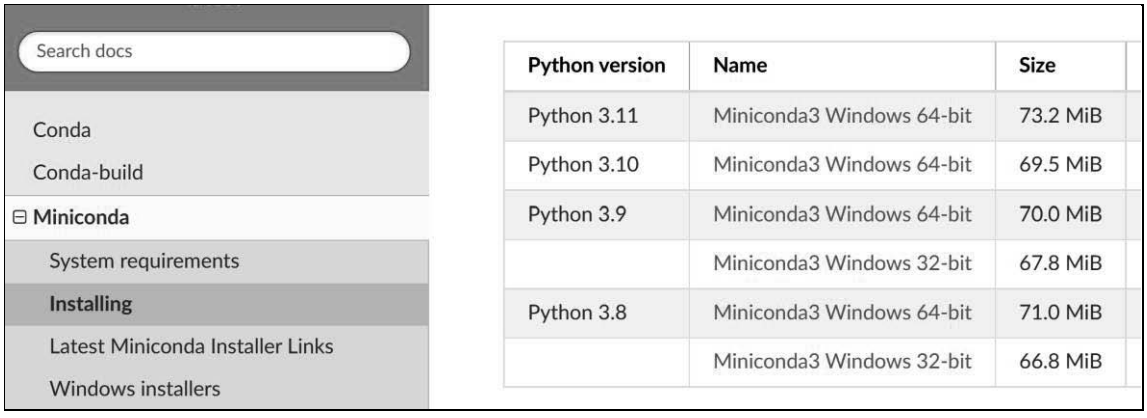

图 2-2　官方 Miniconda 网站提供的下载

注意：如果是 64 位操作系统，则选择以 Miniconda3 开头、以 64 结尾的安装文件，不要下载错了！

下载完成后得到的是 EXE 文件，直接运行即可进入安装过程。安装完成以后，出现如图 2-3 所示的目录结构，说明安装正确。

图 2-3　Miniconda 安装目录

2. 打开控制台

在计算机桌面上依次单击“开始”→“所有程序”→“Miniconda3”→“Miniconda Prompt”命令，打开 Miniconda Prompt 窗口，它与 CMD 控制台类似，输入命令就可以控制和配置 Python。在 Miniconda 中最常用的是 conda 命令，该命令可以执行一些基本操作。

3. 验证Python

在控制台中输入 python，如果安装正确，会打印出版本号以及控制符号。在控制符号下输入代码：

```
print("hello Python")
```

输出结果如图 2-4 所示。

```
(base) C:\Users\xiaohua>python
Python 3.11.4 | packaged by Anaconda, Inc. | (main, Jul  5 2023, 13:47:18) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello Python")
hello Python
>>>
```

图 2-4　验证 Miniconda Python 安装成功

4. 使用pip命令

使用 Miniconda 的好处在于，它能够很方便地帮助我们安装和使用大量的第三方类库。查看已安装的第三方类库的代码是：

```
pip list
```

注意：如果此时命令行还在>>>状态，可以输入 exit()退出。

在 Miniconda Prompt 控制台输入 pip list 代码，结果如图 2-5 所示。

```
(base) C:\Users\xiaohua>pip list
WARNING: Ignoring invalid distribution -qdm (c:\miniforge3\lib\site-packages)
WARNING: Ignoring invalid distribution -harset-normalizer (c:\miniforge3\lib\site-packages)
WARNING: Ignoring invalid distribution -ensorflow-gpu (c:\miniforge3\lib\site-packages)
Package                       Version
----------------------------- ----------------------
absl-py                       1.0.0
aiofiles                      0.8.0
aiohttp                       3.8.1
aiosignal                     1.2.0
alabaster                     0.7.12
altair                        4.2.0
altgraph                      0.17.2
anyio                         3.5.0
argon2-cffi                   21.1.0
arrow                         1.1.1
```

图 2-5　列出已安装的第三方类库

Miniconda 中使用 pip 进行操作的方法还有很多，其中最重要的是安装第三方类库，命令如下：

```
pip install name
```

这里的 name 是需要安装的第三方类库名，假设需要安装 NumPy 包（这个包已经安装过），那么输入的命令就是：

```
pip install numpy
```

结果如图 2-6 所示。

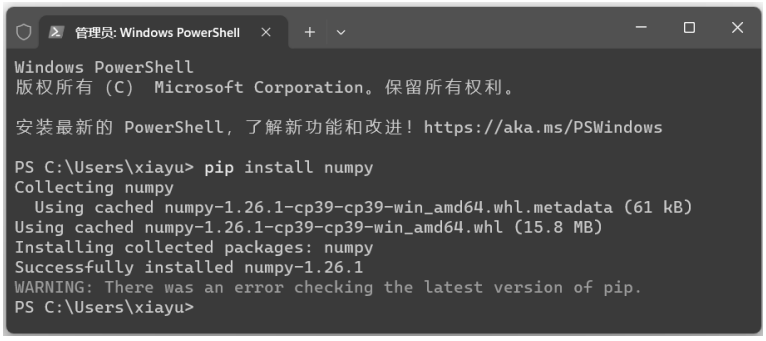

图 2-6　举例自动获取或更新依赖类库

使用 Miniconda 的一个特别的好处就是已经默认安装好了大部分学习所需的第三类库，这样就避免了使用者在安装和使用某个特定类库时，可能出现的依赖类库缺失的情况。

2.1.2　PyCharm 的下载与安装

和其他语言类似，Python 也可以使用 Windows 自带的控制台进行程序编写。但是这种方式对于较为复杂的程序工程来说，容易混淆相互之间的层级和交互文件。因此，在编写程序工程时，笔者建议使用专用的 Python 编译器 PyCharm。

1. 下载和安装PyCharm

进入 PyCharm 官方网站的 Download 页面后可以选择不同的版本，如图 2-7 所示，PyCharm 有收费的专业版和免费的社区版。这里建议读者选择免费的社区版即可。双击下载的安装文件，即可进入安装界面，如图 2-8 所示。直接单击“Next”按钮，采用默认安装即可。

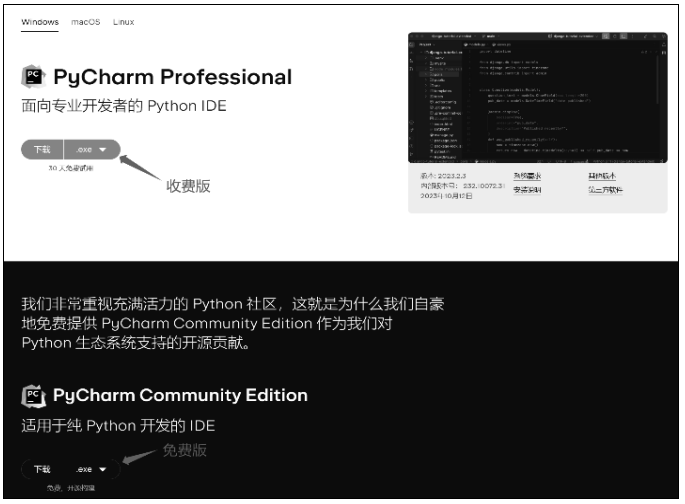

图 2-7　PyCharm 的免费版

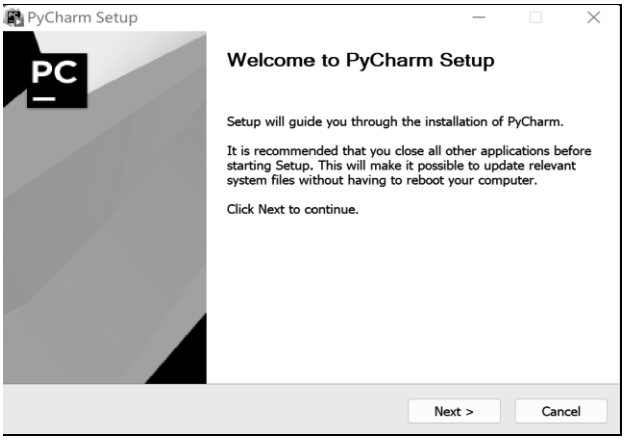

图 2-8　PyCharm 的安装界面

在安装 PyCharm 的过程中需要对配置参数进行选择，建议直接使用默认配置，如图 2-9 所示。

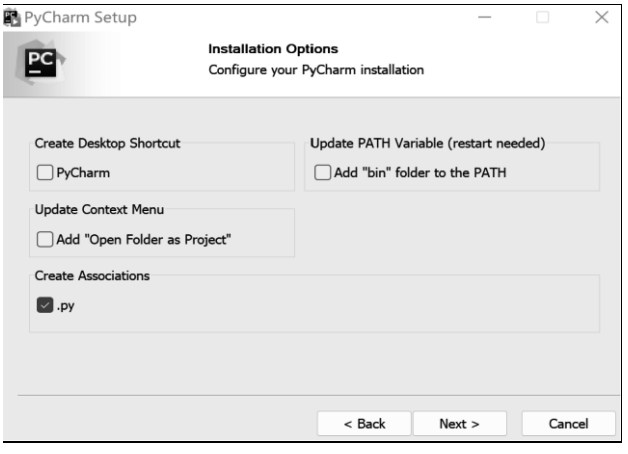

图 2-9　PyCharm 的配置选择（按个人真实情况选择）

安装完成后出现“Finish”按钮，单击该按钮完成安装，如图 2-10 所示。

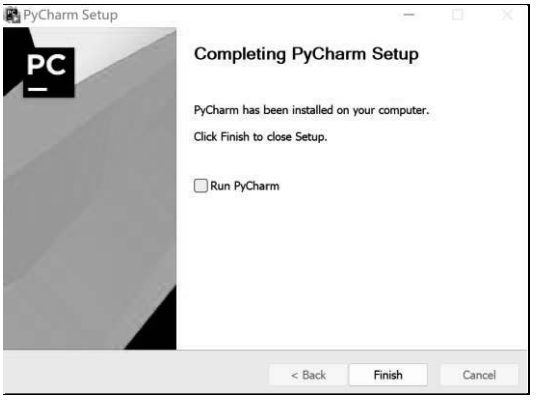

图 2-10　PyCharm 安装完成

2. 使用PyCharm创建程序

单击桌面上新生成的图标，打开 PyCharm，由于是第一次启动 PyCharm，需要接受相关的协议，读者在勾选下方接受协议的复选框后单击“Continue”按钮，进行下一步操作，如图 2-11 所示。

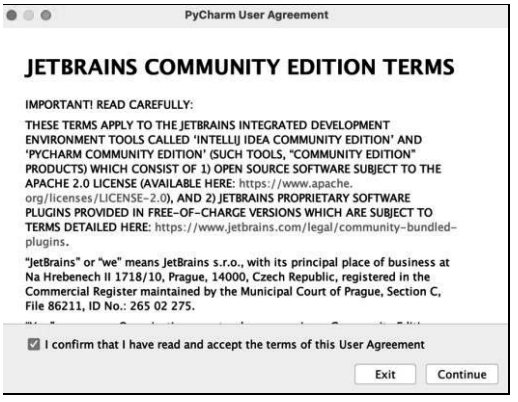

图 2-11　PyCharm 启动

接下来就是创建新的项目，这里可以直接单击“New Project”按钮创建一个新项目，或者单击“Open”按钮打开一个已有的文件夹，如图 2-12 所示。

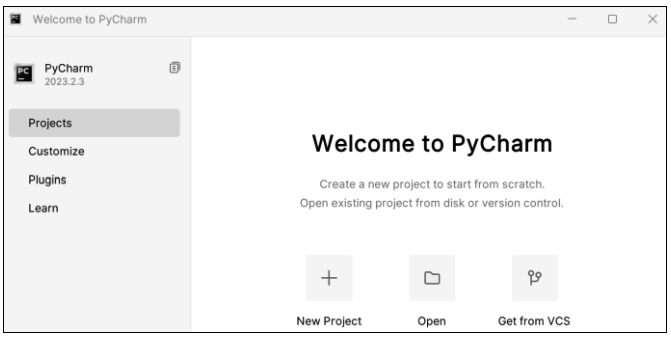

图 2-12　PyCharm 工程创建界面

这里单击“Open”按钮打开一个空文件夹，下面就需要配置 Python 环境路径，填写好 Python 执行文件 python.exe 地址（就是 2.1.2 节安装的 Miniconda 中的 python.exe）后，单击“OK”按钮，如图 2-13 所示。

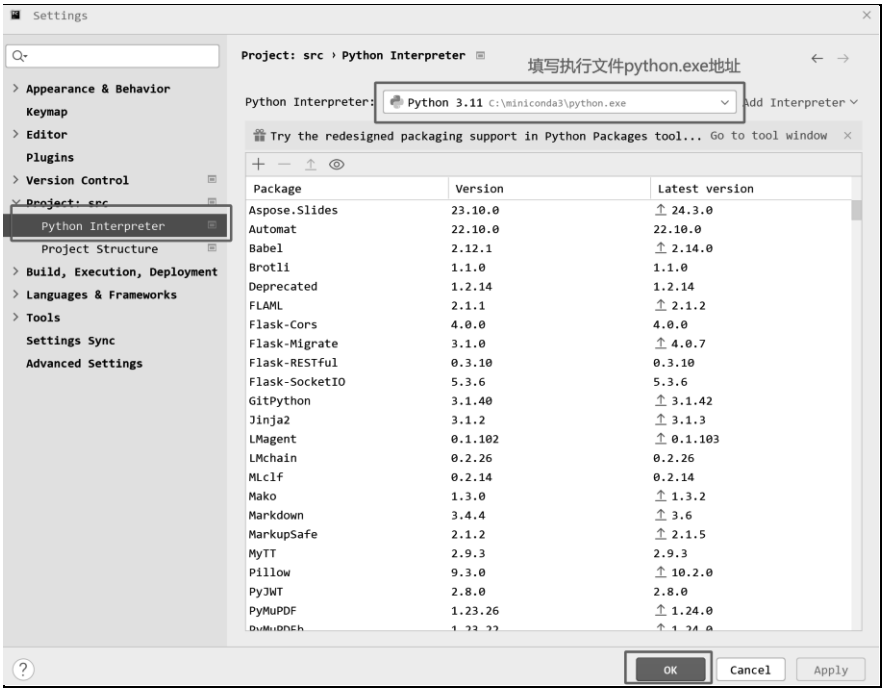

图 2-13　PyCharm 新建文件界面

对于创建的新项目或者打开的空文件夹，PyCharm 默认提供了一个测试项目 main.py，内容如图 2-14 所示。

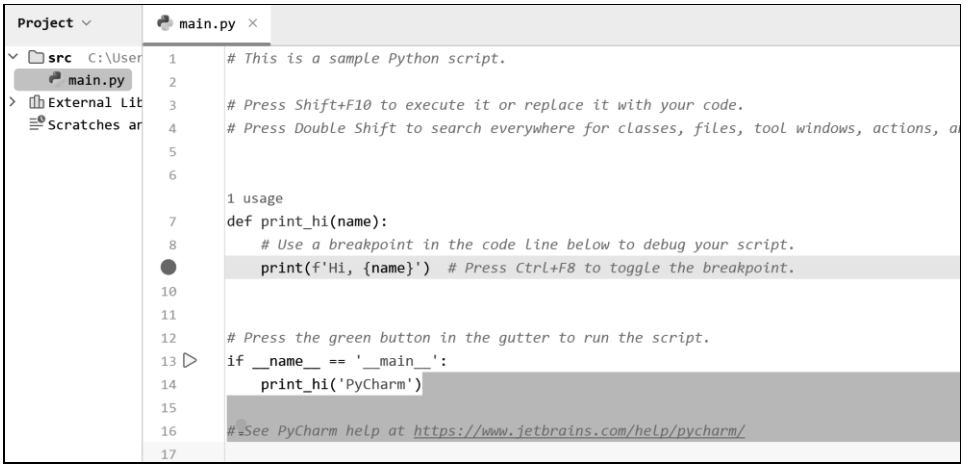

图 2-14　PyCharm 提供的默认测试项目

输入代码并单击菜单栏的“Run”→“run…”运行代码，或者直接右击 main.py 文件名，在弹出的快捷菜单中选择“Run‘main’”命令。如果成功，则输出“Hi，PyCharm”，如图 2-15 所示。

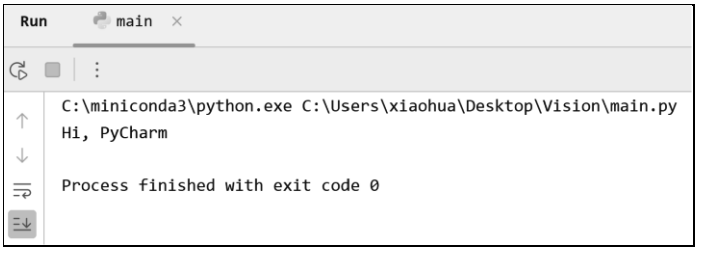

图 2-15　运行成功

至此，Python 与 PyCharm 的配置就完成了。

2.1.3　Python 代码小练习：计算 softmax 函数

对于 Python 科学计算来说，最简单的想法就是可以将数学公式直接表达成程序语言，可以说，Python 满足了这个想法。本小节将使用 Python 实现和计算一个深度学习中最为常见的函数——softmax 函数。至于这个函数的作用，现在不加以说明，笔者只是带领读者尝试实现其程序的编写。

softmax 计算公式如下所示：

$$S_i = \frac{e^{v_i}}{\sum_{0}^{j} e^{v_i}}$$

其中 v_i 是长度为 j 的数列 v 中的一个数，带入 softmax 的结果其实就是先对每一个 v_i 进行以 e 为底的指数计算，变成非负，然后除以所有项之和进行归一化，之后每个 v_i 就可以解释成：在观察到的数据集类别中，特定的 v_i 属于某个类别的概率，或者称作似然（Likelihood）。

提示：softmax 用以解决概率计算中概率结果大而占绝对优势的问题。例如，函数计算结果中有两个值 a 和 b，且 a>b，如果简单地以值的大小为单位进行衡量的话，那么在后续的使用过程中，a 永远被选用，而 b 由于数值较小而不会被选用，但有时候也需要使用数值小的 b，softmax 就可以解决这个问题。

softmax 按照概率选择 a 和 b，由于 a 的概率值大于 b，因此在计算时 a 经常会被取得，而 b 由于概率较小，取得的可能性也较小，但是也有概率被取得。

公式 softmax 的代码如下：

```
import numpy
def softmax(inMatrix):
    m,n = numpy.shape(inMatrix)
    outMatrix = numpy.mat(numpy.zeros((m,n)))
    soft_sum = 0
    for idx in range(0,n):
        outMatrix[0,idx] = math.exp(inMatrix[0,idx])
        soft_sum += outMatrix[0,idx]
    for idx in range(0,n):
        outMatrix[0,idx] = outMatrix[0,idx] / soft_sum
```

```
        return outMatrix
```

可以看到，当传入一个数列后，分别计算每个数值所对应的指数函数值，之后将其相加后计算每个数值在数值和中的概率。

```
    a = numpy.array([[1,2,1,2,1,1,3]])
```

结果请读者自行打印验证。

2.2　环境搭建 2：安装 PyTorch 2.0

Python 运行环境调试完毕后，下面的重点就是安装本书的主角——PyTorch 2.0。

2.2.1　NVIDIA 10/20/30/40 系列显卡选择的 GPU 版本

由于 40 系显卡的推出，因此，目前市场上有 NVIDIA 10、20、30、40 系列显卡并存的情况。对于需要调用专用编译器的 PyTorch 来说，不同的显卡需要安装不同的依赖计算包，笔者在此总结了不同显卡的 PyTorch 版本以及 CUDA 和 cuDNN 的对应关系，如表 2-1 所示。

表 2-1　NVIDIA 10/20/30/40 系列显卡的版本对比

显卡型号	PyTorch GPU 版本	CUDA 版本	cuDNN 版本
10 系列及以前	PyTorch 2.0 以前版本	11.1	7.65
20/30/40 系列	PyTorch 2.0 向下兼容	11.6+	8.1+

注意：这里的区别主要在于显卡运算库 CUDA 与 cuDNN 的区别，当在 20/30/40 系列显卡上使用 PyTorch 时，可以安装 11.6 版本以上以及 cuDNN 8.1 版本以上的计算包，而在 10 系版本的显卡上，笔者还是建议优先使用 2.0 版本以前的 PyTorch。

下面以 PyTorch 2.0 为例演示完整的 CUDA 和 cuDNN 的安装步骤，不同版本的安装过程基本一致。

2.2.2　PyTorch 2.0 GPU NVIDIA 运行库的安装

如果要从 CPU 版本的 PyTorch 开始深度学习之旅，这是可以的，但却不是笔者推荐的一种方式。相对于 GPU 版本的 PyTorch 来说，CPU 版本在运行速度上存在着极大的劣势，很有可能会让我们的深度学习止步不前。

PyTorch 2.0 CPU 版本的安装命令如下：

```
    pip install numpy --pre torch torchvision torchaudio --force-reinstall
--extra-index-url https://download.pytorch.org/whl/nightly/cpu
```

下面就是本节的重头戏，我们以 CUDA 11.7+cuDNN 8.2.0 为例，讲解 PyTorch 2.0 GPU 版本的安装。对于 GPU 版本的 PyTorch 来说，由于调用了 NVIDA 显卡作为其代码运行的主要工具，因此额外需要 NVIDA 提供的运行库作为运行基础。

对于 PyTorch 2.0 的安装来说，最好的安装方法是根据官方提供的安装代码进行安装。PyTorch 官方提供了两种安装模式。使用 conda 安装 CUDA 11.7 的代码如下：

```
    conda install pytorch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2
pytorch-cuda=11.7 -c pytorch -c nvidia
```

使用 pip 安装 CUDA 11.7 的代码如下：

```
    pip install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu117
```

当然，读者也可以根据自己计算机的 GPU 配置要求，查阅 PyTorch 和 CUDA 官网，找到合适的 PyTorch、CUDA、cuDNN 软件版本进行搭配安装。

下面以 CUDA 11.7 为例讲解安装方法。

首先是 CUDA 的安装。百度搜索 CUDA 11.7 download，进入官方下载页面，选择适合的操作系统安装方式（推荐使用 local（本地化）安装方式），如图 2-16 所示。

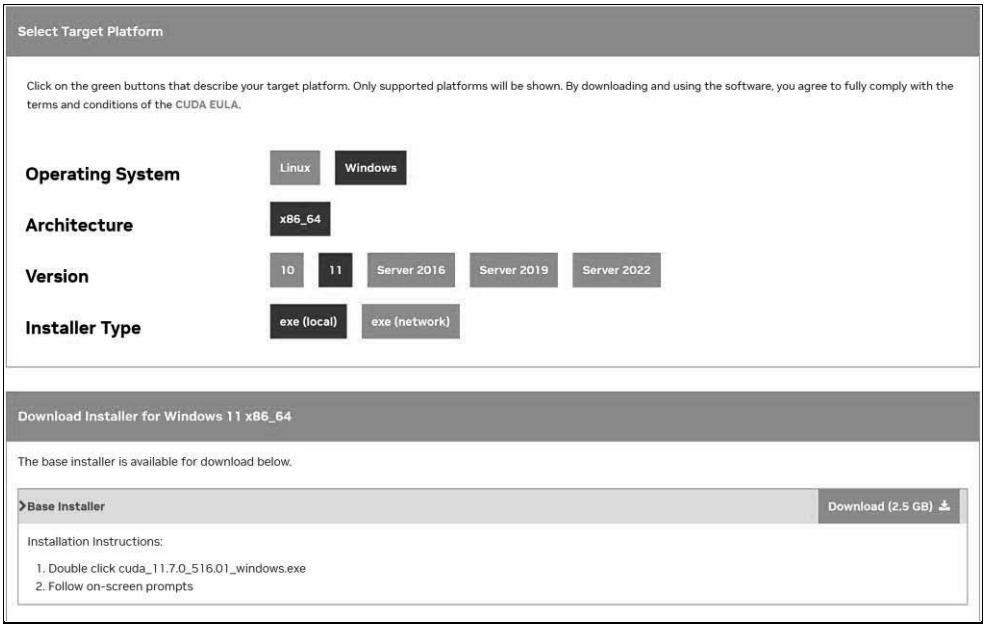

图 2-16　CUDA 下载页面

此时下载下来的是一个 EXE 文件，读者自行安装，不要修改其中的路径信息，完全使用默认路径安装即可。

下一步就是下载和安装对应的 cuDNN 文件。cuDNN 的下载需要先注册一个用户，相信读者可以很快完成，之后直接进入下载页面，如图 2-17 所示。从下载页面上可以看到，CUDA 11.x 对应的是 cuDNN v8.2.0 版本。

注意：不要选择错误的版本，一定要找到对应的版本号。另外，如果读者使用的是 Windows 64 位的操作系统，直接下载 x86_64 版本的 cuDNN 即可。

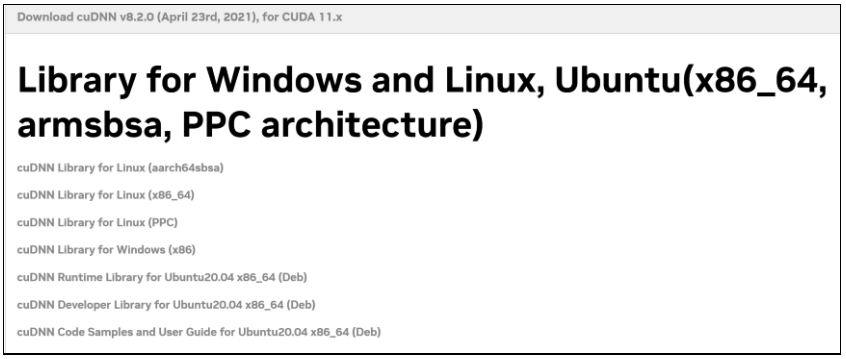

图 2-17　cuDNN 下载页面

下载的 cuDNN 是一个压缩文件，将其解压到 CUDA 安装目录，如图 2-18 所示。然后就是配置环境变量，这里需要将 CUDA 的运行路径加载到环境变量的 PATH 路径中，如图 2-19 所示。

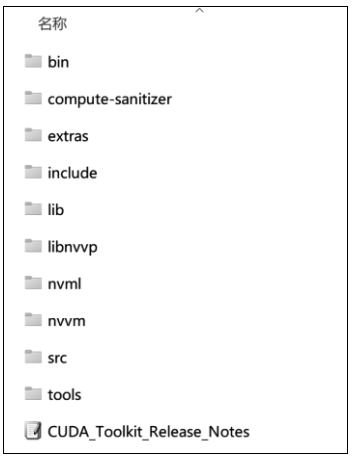

图 2-18　CUDA 安装目录

图 2-19　将 CUDA 路径加载到 PATH 中

最后完成 PyTorch 2.0 GPU 版本的安装，只需要输入本小节开始的 PyTorch 安装代码即可。

2.2.3　Hello PyTorch

在上一小节，我们已经完成了 PyTorch 2.0 的安装，本小节将使用 PyTorch 2.0 进行一个小练习。首先打开 CMD，依次输入如下命令验证安装是否成功：

```
import torch
result = torch.tensor(1) + torch.tensor(2.0)
result
```

结果如图 2-20 所示。

```
(base) C:\Users\xiaohua>python
Python 3.9.6 | packaged by conda-forge | (default, Jul 11 2021, 03:37:25) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> result = torch.tensor(1) + torch.tensor(2.0)
>>> result
tensor(3.)
>>>
```

图 2-20　验证安装是否成功

或者打开前面安装的 PyCharm IDE，新建一个项目，再新建一个 hello_pytorch.py 文件，输入如下代码：

```
import torch
result = torch.tensor(1) + torch.tensor(2.0)
print(result)
```

最终结果请读者自行验证。

2.3　Unet 图像降噪——第一个深度学习项目实战

对于 2.2.3 节的小练习，可能有读者感觉过于简单，仅仅是调用库函数并输入命令来实现所需要的功能。然而，实际上，深度学习程序设计并不是这么简单。为了向读者展示如何使用 PyTorch 进行深度学习的全貌，笔者准备了一个实战示例，详细演示进行深度学习任务所需要的整体流程。读者可能对这里的程序设计和编写不甚熟悉，不用着急，在这里只需要了解每个过程所需完成内容以及涉及的步骤即可。

2.3.1　MNIST 数据集的准备

“HelloWorld”是所有编程语言入门的基础程序，在开始编程学习时，我们打印的第一句话通常就是这个“HelloWorld”。本书也不例外，在深度学习编程中也有其特有的“HelloWorld”，一般就是采用 MNIST 完成一项特定的深度学习项目。

MNIST 是一个手写数字图像数据库，如图 2-21 所示，它有 60 000 个训练样本集和 10 000 个测试样本集。读者可直接使用本书源码库提供的 MNIST 数据集，它位于配套源码的 dataset 文件夹中，如图 2-22 所示。

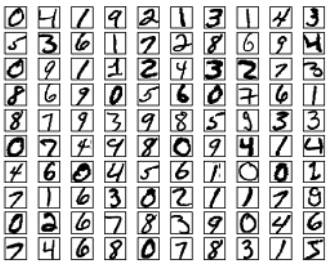

图 2-21　MNIST 文件手写数字

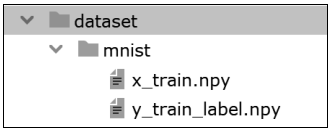

图 2-22　本书源码库提供的 MNIST 数据集

然后使用 NumPy 数据库进行数据的读取，代码如下：

```
import numpy as np
x train = np.load("./dataset/mnist/x train.npy")
y_train_label = np.load("./dataset/mnist/y_train_label.npy")
```

或者读者也可以在网上搜索 MNIST 的下载地址，下载 MNIST 文件中包含的数据集 train-images-idx3-ubyte.gz（训练图片集）、train-labels-idx1-ubyte.gz（训练标签集）、t10k-images-idx3-ubyte.gz（测试图片集）和 t10k-labels-idx1-ubyte.gz（测试标签集），如图 2-23 所示。

```
Four files are available on this site:

train-images-idx3-ubyte.gz:   training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:   training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:     test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:     test set labels (4542 bytes)
```

图 2-23　MNIST 文件中包含的数据集

将下载的 4 个文件进行解压缩。解压缩后，会发现这些文件并不是标准的图像格式，而是二进制文件，其中训练图片集的部分内容如图 2-24 所示。

```
0000 0803 0000 ea60 0000 001c 0000 001c
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

图 2-24　MNIST 文件的二进制表示

MNIST 训练图片集内部的文件结构如图 2-25 所示。

```
TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  60000            number of images
0008     32 bit integer  28               number of rows
0012     32 bit integer  28               number of columns
0016     unsigned byte   ??               pixel
0017     unsigned byte   ??               pixel
........
xxxx     unsigned byte   ??               pixel
```

图 2-25　MNIST 训练集文件结构

MNIST 训练图片集中有 60 000 个实例，也就是说这个文件里面包含了 60 000 个标签内容，每一个标签的值的范围为 0~9。这里我们先解析每一个属性的含义，首先该数据是以二进制格式存储的，我们读取的时候要以 rb 方式读取；其次，真正的数据只有[value]这一项，其他如[type]等字段

只是用来描述的，并不真正包含在数据文件里面。

也就是说，在读取真实数据之前，要读取 4 个 32 bit integer。由[offset]可以看出，真正的 pixel 是从 0016 开始的，一个 int 32 位，所以在读取 pixel 之前要读取 4 个 32 bit integer，也就是 magic number（魔数）、number of images（图片数）、number of rows（行数）、number of columns（列数）。

结合图 2-24 的原始二进制数据内容和图 2-25 的文件结构可以看到，图 2-24 起始的 4 字节数 0000 0803 对应图 2-25 中列表的第一行，类型是 magic number，这个数字为文件校验数，用来确认这个文件是不是 MNIST 里面的 train-images-idx3-ubyte 文件。图 2-24 中的 0000 ea60 对应图 2-25 中列表的第二行，转换为十进制数为 60 000，这是文件总的容量数（number of images）。

下面依次对应，在图 2-24 中，从第 8 字节开始有一个 4 字节数 0000 001c，转换为十进制数为 28，表示的是每幅图片的行数（number of rows）；从第 12 字节开始的 0000 001c 表示每幅图片的列数（number of columns），值也为 28；从第 16 字节开始则是每幅图片像素值的具体内容。这里使用每 784 字节代表一幅图片，如图 2-26 所示。

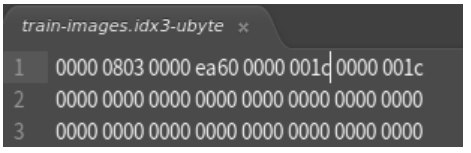

图 2-26　每个手写体被分成 28×28 个像素

2.3.2　MNIST 数据集特征介绍

首先对于数据库的获取，前面介绍了两种不同的 MNIST 数据集的获取方式，笔者推荐使用本书配套源码中的 MNIST 数据集进行数据读取，代码如下：

```
import numpy as np
x_train = np.load("./dataset/mnist/x_train.npy")
y_train_label = np.load("./dataset/mnist/y_train_label.npy")
```

在这里，numpy 函数会根据输入的地址将数据自动分解成训练集和验证集。打印训练集的维度如下：

```
(60000, 28, 28)
(60000, )
```

这里是进行数据处理的第一个步骤，有兴趣的读者可以进一步完成数据的训练集和测试集的划分。

回到 MNIST 数据集，每个 MNIST 实例数据单元也是由两部分构成的：一幅包含手写数字的图片和一个与之相对应的标签。可以将其中的标签特征设置成“y”，而图片特征矩阵以“x”来代替，即所有的训练集和测试集中都包含 x 和 y。

图 2-27 用更为一般化的形式解释了 MNIST 数据实例的展开形式。在这里，图片数据被展开成矩阵的形式，矩阵的大小为 28×28。至于如何处理这个矩阵，常用的方法是将它展开，而展开的方式和顺序并不重要，只需要将它按同样的方式展开即可。

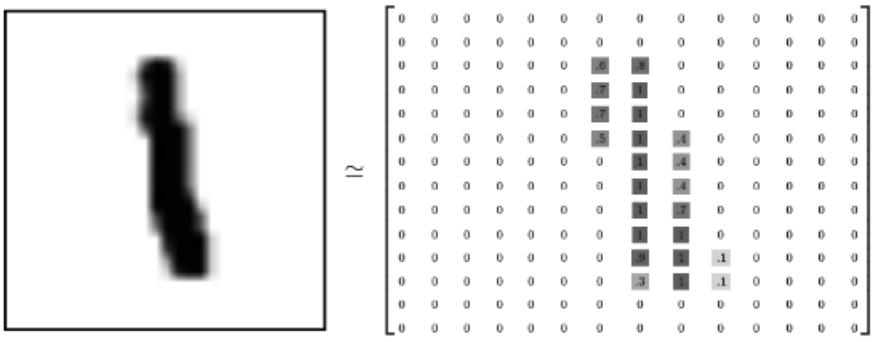

图 2-27　图片转换为向量模式

下面回到对数据的读取，MNIST 数据集实际上就是一个包含着 60 000 幅图片的、大小为 60000×28×28 的矩阵张量[60000,28,28]，如图 2-28 所示。

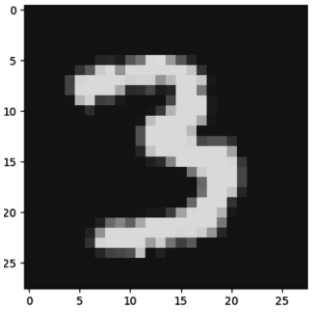

图 2-28　MNIST 数据集的矩阵表示

矩阵中行数指的是图片的索引，用以对图片进行提取；而后面的 28×28 个向量用以对图片特征进行标注。实际上，这些特征向量就是图片中的像素点，每幅手写图片是[28,28]的大小，每个像素转换为 0~1 的一个浮点数，构成矩阵。

2.3.3　Hello PyTorch 2.0——模型的准备和介绍

对于使用 PyTorch 进行深度学习的项目来说，一个非常重要的内容是模型的设计，模型决定着深度学习在项目进行过程中采用何种方式达到目标的主体设计。在本例中，我们的目的是输入一幅图像之后对它进行去噪处理。

对于模型的选择，一个非常简单的思路就是，图像输出的大小就应该是输入的大小。因此，在这里选择 Unet 作为我们的主要模型。

注意：对于模型的选择，现在并不是读者需要考虑的目标，当读者随着对本书学习的深入，见识到更多处理问题的手段后，对模型的选择自然心领神会。

我们可以整体看一下 Unet 的结构（读者目前只需要知道 Unet 的输入大小和输出大小是同样维度的即可），如图 2-29 所示。

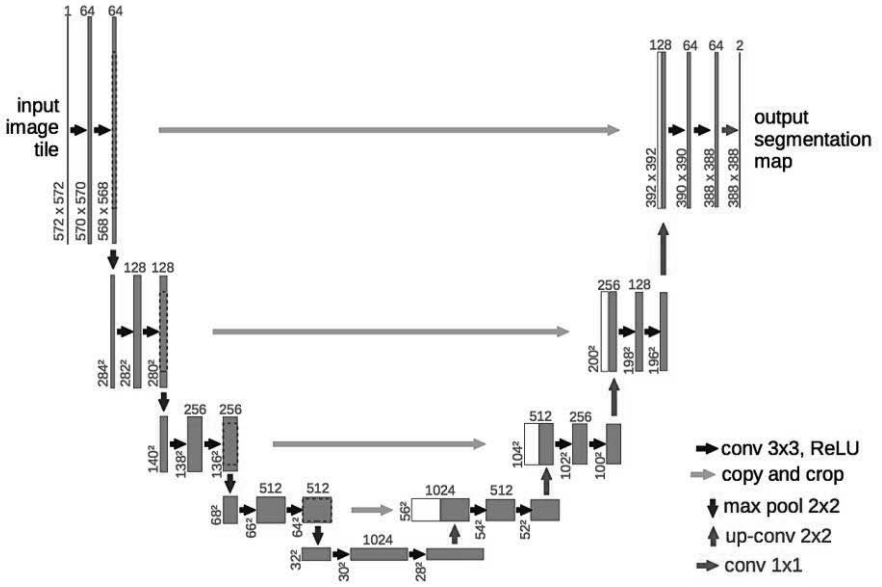

图 2-29　Unet 的结构

可以看到对于整体模型架构来说，它通过若干个“模块”（block）与“直连”（residual）进行数据处理。这部分内容我们在后面章节会讲到，目前读者只需要知道模型有这种结构即可。Unet 模型整体代码如下：

```
    import torch
    import einops.layers.torch as elt

    class Unet(torch.nn.Module):
        def __init__(self):
            super(Unet, self).__init__()

            #模块化结构，这也是后面常用到的模型结构
            self.first_block_down = torch.nn.Sequential(

torch.nn.Conv2d(in_channels=1,out_channels=32,kernel_size=3,padding=1),torch.nn
.GELU(),
                torch.nn.MaxPool2d(kernel_size=2,stride=2)
            )

            self.second_block_down = torch.nn.Sequential(

torch.nn.Conv2d(in_channels=32,out_channels=64,kernel_size=3,padding=1),torch.n
n.GELU(),
                torch.nn.MaxPool2d(kernel_size=2,stride=2)
            )

            self.latent_space_block = torch.nn.Sequential(

torch.nn.Conv2d(in_channels=64,out_channels=128,kernel_size=3,padding=1),torch.
```

```
nn.GELU(),
            )

            self.second block up = torch.nn.Sequential(
                torch.nn.Upsample(scale factor=2),
                torch.nn.Conv2d(in channels=128, out channels=64, kernel size=3,
padding=1), torch.nn.GELU(),
            )

            self.first block up = torch.nn.Sequential(
                torch.nn.Upsample(scale factor=2),
                torch.nn.Conv2d(in_channels=64, out_channels=32, kernel_size=3,
padding=1), torch.nn.GELU(),
            )

            self.convUP_end = torch.nn.Sequential(

torch.nn.Conv2d(in_channels=32,out_channels=1,kernel_size=3,padding=1),
                torch.nn.Tanh()
            )

        def forward(self,img_tensor):
            image = img_tensor

            image = self.first_block_down(image)#;print(image.shape)
#torch.Size([5, 32, 14, 14])
            image = self.second_block_down(image)#;print(image.shape)
#torch.Size([5, 16, 7, 7])
            image = self.latent_space_block(image)#;print(image.shape)
#torch.Size([5, 8, 7, 7])

            image = self.second_block_up(image)#;print(image.shape)
#torch.Size([5, 16, 14, 14])
            image = self.first_block_up(image)#;print(image.shape)
#torch.Size([5, 32, 28, 28])
            image = self.convUP_end(image)#;print(image.shape)
#torch.Size([5, 32, 28, 28])
            return image

    if __name__ == '__main__':          #main 是 Python 进行单文件测试的技巧，请读者记住
        image = torch.randn(size=(5,1,28,28))
        Unet()(image)
```

在这里笔者通过一个 main 架构标识了可以在单个文件中对文件进行测试，请读者记住这种写法。

2.3.4　对目标的逼近——模型的损失函数与优化函数

除了模型之外，想要完成一个深度学习项目，另一个非常重要的内容就是设定模型的损失函数

与优化函数。这部分内容对于初学者来说可能不是太熟悉，在这里初学者只需要知道有这部分内容即可。

首先是对于损失函数的选择，在这里选用 MSELoss 作为损失函数，MSELoss 损失函数中文名字为均方损失函数。

MSELoss 的作用是计算预测值和真实值之间的欧式距离。预测值和真实值越接近，两者的均方差就越小，均方差函数常用于线性回归模型的计算。在 PyTorch 中使用 MSELoss 的代码如下：

```
    loss = torch.nn.MSELoss(reduction="sum")(pred, y_batch)
```

下面就是优化函数的设定，在这里采用了 Adam 优化器。对于 Adam 优化函数请读者自行学习，这里只提供使用 Adam 优化器的代码，如下所示。

```
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)
```

2.3.5　Let's do it！——基于深度学习的模型训练

在进行了深度学习的数据集准备、模型介绍以及损失函数与优化函数的介绍之后，下面就是使用 PyTorch 训练出一个可以实现去噪功能的深度学习模型，完整代码如下（本代码位于随书提供的源码文件中的“第 2 章”，读者可以直接运行）：

```
    import os
    os.environ['CUDA_VISIBLE_DEVICES'] = '0' #指定使用 GPU
    import torch
    import numpy as np
    import unet
    import matplotlib.pyplot as plt
    from tqdm import tqdm

    batch_size = 320                          #设定每次训练的批次数
    epochs = 1024                             #设定训练次数

    #device = "cpu"     #PyTorch 的特性，需要指定计算的硬件，如果没有 GPU 的存在，就使用
CPU 进行计算
    device = "cuda"     #这里默认使用 GPU，如果出现运行问题，可以将其改成 CPU 模式

    model = unet.Unet()                  #导入 Unet 模型
    model = model.to(device)             #将计算模型传入 GPU 硬件等待计算
    #model = torch.compile(model)        #PyTorch 2.0 的特性，加速计算速度，选择性使用
    optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)   #设定优化函数

    #载入数据
    x_train = np.load("../dataset/mnist/x_train.npy")
    y_train_label = np.load("../dataset/mnist/y_train_label.npy")

    x_train_batch = []
    for i in range(len(y_train_label)):
        if y_train_label[i] < 2:                  #为了加速演示，笔者只对数据集中小于 2
的数字（也就是 0 和 1）进行运行，读者可以自行增加训练个数
            x_train_batch.append(x_train[i])
```

```
    x_train = np.reshape(x_train_batch, [-1, 1, 28, 28])   #修正数据输入维度：
([30596, 28, 28])
    x_train /= 512.
    train_length = len(x_train) * 20                       #增加数据的单词循环次数


    for epoch in range(epochs):
        train_num = train_length // batch_size             #计算有多少批次数

        train_loss = 0                                     #用于损失函数的统计
        for i in tqdm(range(train_num)):                   #开始循环训练
            x_imgs_batch = []                              #创建数据的临时存储位置
            x_step_batch = []
            y_batch = []
            #对每个批次内的数据进行处理
            for b in range(batch_size):
                img = x_train[np.random.randint(x_train.shape[0])]  #提取单幅图片内容
                x = img
                y = img

                x_imgs_batch.append(x)
                y_batch.append(y)

            #将批次数据转换为 PyTorch 对应的 tensor 格式，并将其传入 GPU 中
            x_imgs_batch = torch.tensor(x_imgs_batch).float().to(device)
            y_batch = torch.tensor(y_batch).float().to(device)


            pred = model(x_imgs_batch)                     #对模型进行正向计算
            loss = torch.nn.MSELoss(reduction=True)(pred, y_batch)/batch_size
    #使用损失函数进行计算

            #读者记住下面就是固定格式，一般而言这样使用即可
            optimizer.zero_grad()                          #对结果进行优化计算
            loss.backward()                                #损失值的反向传播
            optimizer.step()                               #对参数进行更新

            train_loss += loss.item()                      #记录每个批次的损失值
        #计算并打印损失值
        train_loss /= train_num
        print("train_loss:", train_loss)

        #下面是对数据进行打印
        image = x_train[np.random.randint(x_train.shape[0])]    #随机挑选一条数据进
行计算
        image = np.reshape(image,[1,1,28,28])          #修正数据维度

        image = torch.tensor(image).float().to(device) #挑选的数据传入硬件中等待计算
        image = model(image)                           #使用模型对数据进行计算
```

```
        image = torch.reshape(image, shape=[28,28])     #修正模型输出结果
        image = image.detach().cpu().numpy()  #将计算结果导入 CPU 中进行后续计算或者展示

        #展示或存储数据结果
        plt.imshow(image)
        plt.savefig(f"./img/img_{epoch}.jpg")
```

这里展示了完整的模型训练过程，首先是传入数据，然后使用模型对数据进行计算，再将计算结果与真实值的误差回传到模型中，最后 PyTorch 框架根据回传的误差对整体模型参数进行修正。训练结果如图 2-30 所示。

图 2-30　训练结果

从图 2-30 中可以很清楚地看到，随着训练的进行，模型逐渐学会对输入的数据进行整形和输出。此时，模型的输出结果已经能够很好地对输入的图形细节进行修正，有兴趣的读者可以自行完成这部分代码的运行。

2.4　本章小结

本章是 PyTorch 实战程序设计的起点。在这一章中，笔者引导读者熟悉了 PyTorch 程序设计的

环境，并指导安装了必备的软件。此外，还展示了如何使用 PyTorch 框架完成第一个计算机视觉任务的整体设计，并对部分组件进行了详细介绍。

实际上，深度学习程序设计是由众多小组件构成的。每个组件都在整个程序中发挥着不可或缺的作用。本书的后续章节将对每个组件进行深入剖析，帮助读者全面理解并掌握它们的用法。通过本书的学习，读者将能够自如地运用 PyTorch 框架进行计算机视觉任务的设计与实践。