# 摘要

随着社会的发展和生活质量的提高，人们对健康的关注度不断提升，乳腺癌作为一种常见的恶性肿瘤，其早期诊断和治疗显得尤为重要。乳腺癌的发病率在全球范围内逐年上升，早期发现和治疗对于改善患者的预后具有重要意义。近年来，卷积神经网络（CNN）在医学影像分析中取得了显著进展，特别是在乳腺癌的自动化诊断中表现出色。

本报告介绍了一种基于CNN的乳腺癌检测方法。该方法通过构建一个包含大量乳腺癌影像的标注数据集，并采用经典的CNN模型（如VGG16和ResNet-50）进行训练与分类，能够自动判别乳腺癌影像中的良性和恶性肿瘤。通过对数据进行预处理和增强处理，提升了模型的训练效果和泛化能力。实验结果表明，该系统在乳腺癌影像分类中取得了较高的准确率，尤其是在恶性肿瘤的识别上表现优异。这种方法不仅提高了乳腺癌的诊断效率，也为临床医生提供了一个有力的辅助工具，有助于实现乳腺癌的早诊早治。

# 题目

**基于CNN的乳腺癌检测**

# 乳腺癌的研究背景及意义

乳腺癌是全球女性中最常见的恶性肿瘤之一，其发病率和死亡率均居高位。根据IARC发布的数据，2020年全球新发乳腺癌病例达226.1万例，占全球新发癌症病例的11.7%，居女性癌症首位。乳腺癌的高发病率和死亡率使其成为全球女性健康的重大威胁。

早期诊断对于提高乳腺癌患者的生存率至关重要。传统的乳腺癌诊断方法依赖于放射科医生的经验，但这些方法存在准确率和一致性不足的问题。近年来，随着深度学习技术的发展，卷积神经网络（CNN）在医学影像分析中取得了显著进展。CNN能够自动提取影像特征，提高乳腺癌影像的分类准确率。例如，VGG16和ResNet-50等经典模型在乳腺癌影像的良性和恶性肿瘤分类中表现出色。

基于CNN的乳腺癌检测系统不仅能够提高诊断的准确性和效率，还能为临床医生提供有力的辅助工具。这有助于实现乳腺癌的早期筛查和精准治疗，从而改善患者的预后。此外，随着技术的不断进步，基于CNN的检测系统在乳腺癌的早期发现和个性化治疗中将发挥越来越重要的作用。

# CNN

卷积神经网络（Convolutional Neural Networks，简称CNN）是一种深度学习模型，广泛应用于图像识别、物体检测等计算机视觉任务中。其设计灵感来源于人脑的视觉神经系统，通过卷积和池化等操作来提取和分析图像特征。CNN的发展始于1998年，当时Yann LeCun提出了LeNet-5，这是最早的卷积神经网络之一，主要用于手写数字识别。2012年，AlexNet在ImageNet挑战赛中取得了突破性成绩，标志着深度学习和CNN的兴起。此后，GoogleNet、VGGNet、ResNet等模型相继出现，推动了CNN在性能和效率上的不断提升。

CNN的核心技术包括卷积层、激活函数、池化层和全连接层。卷积层通过卷积核与输入图像进行点积运算，提取图像的局部特征。激活函数（如ReLU）引入非线性，增强模型的表达能力。池化层则通过降维操作减少特征的空间维度，同时保留重要信息。全连接层用于将提取的特征进行整合，完成最终的分类或回归任务。

典型的CNN架构如AlexNet和ResNet，通过多层卷积和池化层的组合，实现了对图像的深层次特征提取。AlexNet首次在大规模图像识别任务中展示了深度卷积网络的强大能力，而ResNet则通过引入残差连接解决了深层网络训练中的梯度消失问题，使得网络可以更深地扩展。

CNN的工作机制基于卷积运算和反向传播算法。卷积操作通过滤波器与输入图像的滑动窗口计算，提取局部特征。反向传播算法用于更新网络的权重，通过计算损失函数的梯度来优化模型。

# 需求分析与系统方案设计

# 4.1图像场景多分类需求分析

### 4.1.1开发环境搭建

Anaconda 是 python 的一个集成管理工具，提供了包管理与环境管理的功能，可以很方便地解决多版本 python 并存、切换以及各种第三方包安装问题；Cuda 和 Cudnn 加速库的配置也可以充分发挥 GPU 计算的优势；再配合 Tensorflow 的 keras 高阶 API 用于构建和训练深度学习神经网络模型。

### 4.1.2模型参数绘制

任务设计分析过程中经常需要绘制图像与标签的对应关系、模型指标结果可视化、变参条件下的结果分析等操作，配合 matplotlib 的 pyplot 模块用于绘制实验所需的结果显示及比较评估。

### 4.1.3深度学习神经网络搭建

通过 Tensorflow 的 Keras 的高效 API 接口搭建实验设计所需的 CNN 网络模型并进行模型的训练和评估，最后利用模型进行预测和结果显示。

### 4.1.4系统比较评估

小组间分组进行两个不同参数的 CNN 网络的训练，分析实验结果并进行系统评估比较。

## 4.2方案设计

### 4.2.1 CNN 模型设计

本实验设计拟通过构建 CNN 网络模型进行训练进而分析对场景图像进行分析的准确性。充分利用 Keras 这一 API 进行 CNN 网络模型的构建：1）使用 Sequential 按层顺序构建模型；2）添加各类 Keras 中定义好的 layer 构建 CNN 模型；3）用 compile 方法对模型进行编译，指定优化器，目标函数，评估指标，以及回调函数；4）用 summary 方法来查看模型的结构和参数数量

### 4.2.2模型训练和评估

使用 GPU 训练模型，安装 CUDA 和 cuDNN 以支持相应计算，用模型的 fit 方法训练模型并查看后台 GPU 占用情况确认使用 GPU 加速计算；在模型训练完成后，可以用evaluate 方法对模型进行评估。

**4.2.3绘制模型指标**

损失函数除了作为模型训练时候的优化目标，也能够作为模型好坏的一种评价指标。当然，在实验过程中，模型好坏的指标有 Keras 类中的 metrics 模块定义了许多评价模型指标的参数，最常见的就是 accuracy 用于反馈模型识别分类的准确性，可以通过pyplot 编写代码绘制模型训练过程中每训练一个epoch 之后accuracy 的变化趋势从而掌握模型的收敛情况等；除此之外，利用测试数据集对训练好的模型进行评估的结果显示也可以通过图像绘制直观地进行结果分析。

## 4.3可行性分析

技术层面上，深度学习框架主要使用 Python 脚本语言，而本科课程中已学习过Python，故只需熟悉各类计算机视觉处理及深度学习网络框架的库便可上手，除此之外，课程上已经讲解了深度学习的部分基础知识点，如损失函数等，也便于对程序结果进行分析评估；资源上，小组成员的机器环境大多都具有 NVIDIA 显卡，便于加速训练神经网络模型，所以整体课程设计的进度时间安排能满足需求。根据实际情况及可行性分析可知，选择此方案能实现需求功能，可以达到预期的结果。

1. **任务清单**

|  |  |  |
| --- | --- | --- |
| 名称 | 具体事项 | 完成期限 |
| 前期准备 | 查阅资料了解深度学习及其应用领域；完成开发设计环境的搭建 | 11 月 12 日 |
| 方案设计 | 通过需求分析制定相应的设计方案；同时与组员进行商议改进并分工 | 11 月 15 日 |
| 实验过程设计 | 设计实验的整体流程并进行组内分工完善，做好组内成果的交流分析 | 11 月 22 日 |
| 程序编写 | 查阅资料并设计算法设计框架训练模型。 | 11 月 29 日 |
| 实验结果分析 | 与组员一起对各自程序运行结果进行分析并对系统进行比较评估 | 12 月 22 日 |
| 报告撰写 | 在课程设计的流程设计过程中跟进报告内容的整理并最后完善报告的撰写 | 12 月24 日 |

# 任务完成过程

## 6.1环境搭建

**安装Anaconda：**

访问Anaconda官网，下载并安装适合您操作系统的Anaconda版本。在安装过程中，确保勾选“添加到环境变量”的选项.

**创建虚拟环境：**

安装完成后，打开Anaconda Prompt。

输入命令 `conda create -n tf python=3.7` 来创建一个名为“tf”的虚拟环境，并指定Python版本为3.7。

**安装CUDA和Cudnn：**

确认您的NVIDIA显卡支持的CUDA版本，然后从NVIDIA官网下载并安装相应版本的CUDA。

安装完成后，下载与CUDA版本匹配的Cudnn库，并将其文件复制到CUDA的安装目录下，替换原有文件。

**安装Tensorflow：**

在Anaconda Prompt中，激活“tf”虚拟环境，输入命令 `conda activate tf`。

更改conda的下载源为清华镜像源，输入命令 `conda config --add channels https://mirrors.tuna.tsinghua.edu.cn/anaconda/pkgs/main/。

安装GPU版本的Tensorflow，输入命令 `conda install tensorflow-gpu`。

安装完成后，编写测试语句验证Tensorflow是否安装成功。

至此，Anaconda、CUDA、Cudnn和Tensorflow的环境搭建完成，您可以使用Pycharm作为IDE进行后续的软件开发。

## 6.2数据库的载入与预处理

### 6.2.1乳腺癌威斯康星州数据集

**数据集特征：**

| **实例数** | **569** | |
| --- | --- | --- |
| **属性数量** | 30个数值型预测属性和1个类别标签 | |
| **属性信息** | - 半径（从中心到周长上的点的距离的平均值） - 纹理（灰度值的标准差） - 周长 - 面积 - 平滑度（半径长度的局部变化） - 紧密度（周长\*周长 /面积-1.0） - 凹度（轮廓凹陷的严重程度） - 凹点（轮廓的凹入部分的数量） - 对称 - 分形维数（“海岸线近似”-1） 为每个图像计算这些特征的平均值，标准和“最差”（最大值），从而得到30个特征。 例如，字段0是平均半径，字段10是半径标准差，字段20是最差半径。 标签分类： -- WDBC-恶性 -- WDBC-良性 | |
| **统计摘要** | | IMG_256 | |

### 6.2.2载入和预处理

1. *# 载入数据*
2. loaded\_data = load\_breast\_cancer()
3. X = loaded\_data.data
4. y = loaded\_data.target
5. *# 将X正则化*
6. X = preprocessing.scale(X)
7. np.random.seed(120)  *# 设置随机种子，让每次结果都一样，方便对照*
8. np.random.shuffle(X)  *# 使用shuffle()方法，让输入x\_train乱序*
9. np.random.seed(120)  *# 设置随机种子，让每次结果都一样，方便对照*
10. np.random.shuffle(y)  *# 使用shuffle()方法，让输入y\_train乱序*
11. tf.random.set\_seed(120)  *# 让tensorflow中的种子数设置为120*

## 6.3算法设计

### 6.3.1 CNN模型的搭建

采用CNN的原因:若输入数据集有较多的属性，导致整个神经网络有非常多的待训练参数，很可能导过拟合的现象发生，那么将输入的属性进行卷积核的特征提取，则可以有效改善该问题。

①Conv1D层

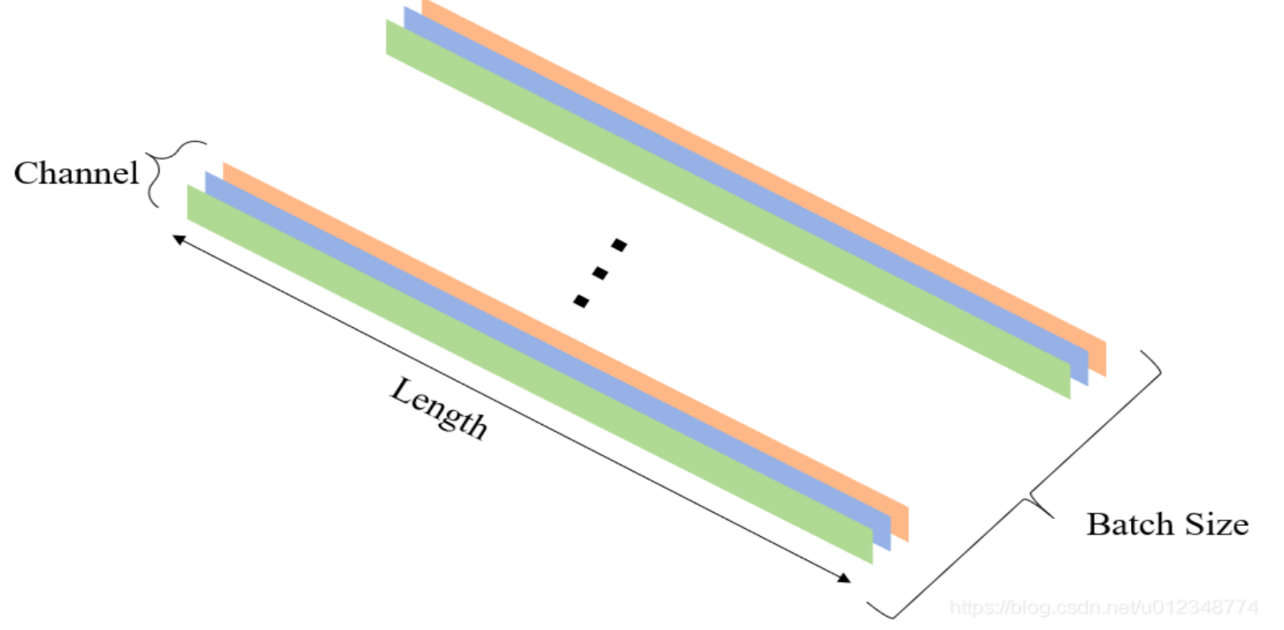


图 6.1 CNN输入数据可视化

1D 卷积层 (例如时序卷积)。  
该层创建了一个卷积核，该卷积核以 单个空间（或时间）维上的层输入进行卷积， 以生成输出张量。 如果 use\_bias 为 True， 则会创建一个偏置向量并将其添加到输出中。 最后，如果 activation 不是 None，它也会应用于输出。

**参数**

（1）filters: 整数，输出空间的维度 （即卷积中滤波器的输出数量）。

（2）kernel\_size: 一个整数，或者单个整数表示的元组或列表， 指明 1D 卷积窗口的长度。

（3）strides: 一个整数，或者单个整数表示的元组或列表， 指明卷积的步长。 指定任何 stride 值 != 1 与指定 dilation\_rate 值 != 1 两者不兼容。

（4）padding: “valid”, “causal” 或 “same” 之一 (大小写敏感) “valid” 表示「不填充」。 “same” 表示填充输入以使输出具有与原始输入相同的长度。 “causal” 表示因果（膨胀）卷积， 例如，output[t] 不依赖于 input[t+1:]， 在模型不应违反时间顺序的时间数据建模时非常有用。

（5）data\_format: 字符串, “channels\_last” (默认) 或 “channels\_first” 之一。输入的各个维度顺序。 “channels\_last” 对应输入尺寸为 (batch, steps, channels) (Keras 中时序数据的默认格式) 而 “channels\_first” 对应输入尺寸为 (batch, channels, steps)。

（5）dilation\_rate: 一个整数，或者单个整数表示的元组或列表，指定用于膨胀卷积的膨胀率。 当前，指定任何 dilation\_rate 值 != 1 与指定 stride 值 != 1 两者不兼容。

（6）activation: 要使用的激活函数。 如未指定，则不使用激活函数 (即线性激活： a(x) = x)。

（7）use\_bias: 布尔值，该层是否使用偏置向量。

（8）kernel\_initializer: kernel 权值矩阵的初始化器 。

（9）bias\_initializer: 偏置向量的初始化器 。

（10）kernel\_regularizer: 运用到 kernel 权值矩阵的正则化函数。

（11）bias\_regularizer: 运用到偏置向量的正则化函数。

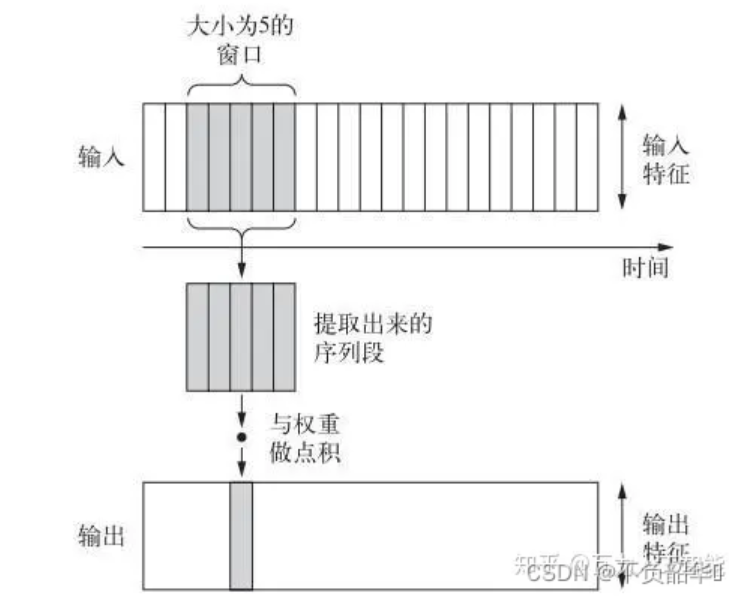
（12）activity\_regularizer: 运用到层输出（它的激活值）的正则化函数 。

（13）kernel\_constraint: 运用到 kernel 权值矩阵的约束函数。

（14）bias\_constraint: 运用到偏置向量的约束函数。

②池化层

它实际上是一种形式的降采样。有多种不同形式的非线性池化函数，而其中“最大池化（Max pooling）”是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。直觉上，这种机制能够有效地原因在于，在发现一个特征之后，它的精确位置远不及它和其他特征的相对位置的关系重要。池化层会不断地减小数据的空间大小，因此参数的数量和计算量也会下降，这在一定程度上也控制了过拟合。通常来说，CNN的卷积层之间都会周期性地插入池化层。



pool\_size表示池化窗口的大小，参数strides表示移动步长，默认为None，表示和pool\_size的大小一样。

图 6.2 CNN中采样方式

③Flatten层

作用：Flatten层用来将输入“压平”，即把多维的输入一维化，常用在从卷积层到全连接层的过渡。Flatten不影响batch的大小。

④全连接层Dense 层一可在 model 中添加神经网络层，model.add(Dense())。

下图（从网上找的图）的黑圆圈为神经元一，输入层的神经元会将输入数据直接传递给下一层，在下一层的神经元中对输入值进行处理。因此用 Dense 层作为输入层时，同时增了输入层和隐藏层。

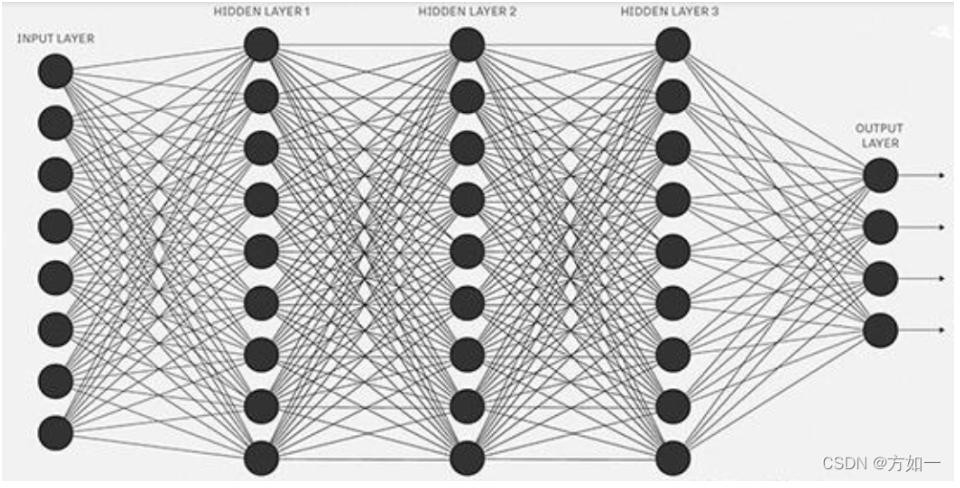


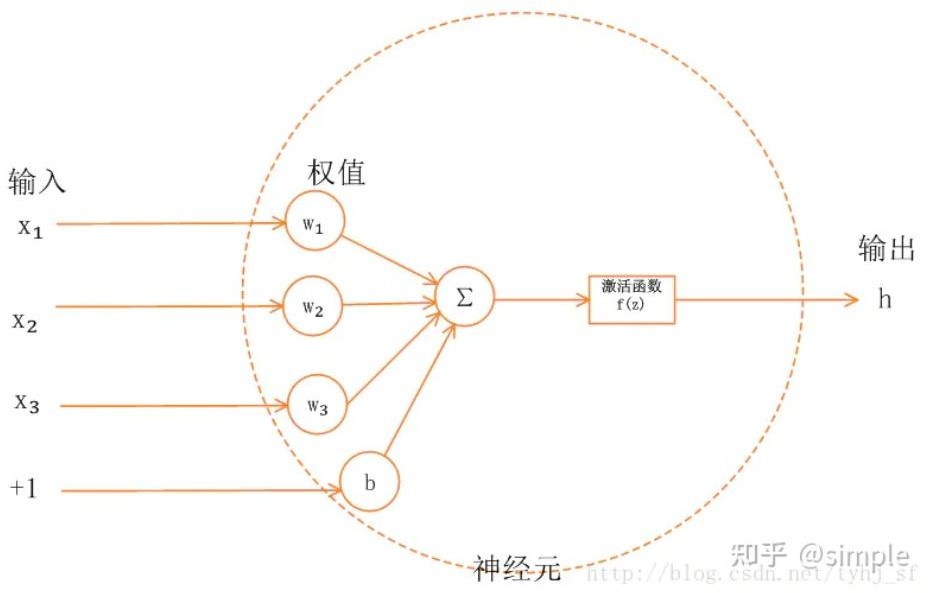
图 6.3 神经网络图

由上图可以，在输入层没有对输入数据进行处理，输入层神经元节点会将输入数据直接传递给下一层(隐层或输出层)。由于 Dense 层的输出公式为: Out= Activation( Input.Kernel )+Bias，因此 Dense 层参数计算公式为: Param=(上-层神经元数量) x (本层的神经元数量) + (本层的神经元数量) 。其中，(上一层神经元数量) x

(本层的神经元数量) 代表的是 input . kernel 的参数个数,加上的本层的神经元数量代表的是 bias 的参数。

⑤激活函数

（1）Relu 激活函数

图 6.4 神经元示意图

如上图所示，神经网络中的每个神经元节点接受上一层神经元的输出值作为本神经元的输入值，并将输入值传递给下一层，输入层神经元节点会将输入属性值直接传递给下一层(隐层或输出层)。在多层神经网络中，上层节点的输出和下层节点的输入之间具有一个函数关系，这个函数称为激活函数。

简单来说，激活函数，并不是去激活什么，而是指如何把“激活的神经元的特征” 通过函数把特征保留并映射出来，即负责将神经元的输入映射到输出端。

引入 Relu 的原因：

第一，采用 sigmoid 等函数，算激活函数时（指数运算)，计算量大，反向传播求误差梯度时，求导涉及除法，计算量相对大，而采用 Relu 激活函数，整个过程的计算量节省很多。

第二，对于深层网络,sigmoid 函数反向传播时，很容易就会出现梯度消失的情况

〈(在 sigmoid 接近饱和区时，变换太缓慢，导数趋于 0，这种情况会造成信息丢失)， 从而无法完成深层网络的训练。

第三，ReLu 会使一部分神经元的输出为 0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生。

Relu 函数的优势：没有饱和区，不存在梯度消失问题，防止梯度弥散;稀疏性;没有复杂的指数运算，计算简单、效率提高;实际收敛速度较快，比 Sigmoid/tanh 快很多;比 Sigmoid 更符合生物学神经激活机制。

（2）Softmax

Softmax函数与正常的max函数不同：max函数仅输出最大值，但Softmax函数确保较小的值具有较小的概率，并且不会直接丢弃。我们可以认为它是arg max ⁡ \argmaxargmax函数的概率版本或“soft”版本。Softmax函数的分母结合了原始输出值的所有因子，这意味着Softmax函数获得的各种概率彼此相关。

Softmax激活函数的特点：

1在零点不可微。

2负输入的梯度为零，这意味着对于该区域的激活，权重不会在反向传播期间更新，因此会产生永不激活的死亡神经元。

⑥模型可视化

1. model = keras.Sequential([
2. keras.layers.Conv1D(36, 5, activation='relu', input\_shape=(30, 1)),
3. keras.layers.MaxPool1D(1),
4. keras.layers.Conv1D(20, 5, activation='relu'),
5. keras.layers.MaxPool1D(1),
6. keras.layers.Conv1D(20, 5, activation='relu'),
7. keras.layers.MaxPool1D(1),
8. keras.layers.Flatten(),
9. keras.layers.Dense(32, activation='relu'),
10. keras.layers.Dense(2, activation='softmax')
11. ])

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

conv1d (Conv1D) (None, 26, 36) 216

max\_pooling1d (MaxPooling1D (None, 26, 36) 0

)

conv1d\_1 (Conv1D) (None, 22, 20) 3620

max\_pooling1d\_1 (MaxPooling (None, 22, 20) 0

1D)

conv1d\_2 (Conv1D) (None, 18, 20) 2020

max\_pooling1d\_2 (MaxPooling (None, 18, 20) 0

1D)

flatten (Flatten) (None, 360) 0

dense (Dense) (None, 32) 11552

dense\_1 (Dense) (None, 2) 66

=================================================================

## 6.4优化器和损失函数

1.*# 优化器和损失函数*

2.model.compile(

3.    optimizer='rmsprop',

4.    loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=False),

5.    metrics=['sparse\_categorical\_accuracy']

6.)

1. 优化器

①优化器分类

**基本梯度下降法**：包括标准梯度下降法(GD, Gradient Descent)，随机梯度下降法(SGD, Stochastic Gradient Descent)及批量梯度下降法(BGD, Batch Gradient Descent)；

**动量优化法**：包括标准动量优化方法（MomentumOptimizer)、牛顿加速梯度动量优化方法(NAG, Nesterov accelerated gradient)等；Momentum（历史梯度乘以一个权重）-> Nesterov（向前看一步梯度)

**自适应学习率优化法**：包括Adagrad(Adaptive gradient algorithm)算法，RMSProp(root mean square prop)算法，Adam算法等；dagrad（首次引入二阶动量) -> adadelta （只关注过去一段时间）->RMSProp（修改AdaGrad的梯度积累为一阶指数加权的移动平均）

②常见的优化器

（i）AdaDelta

是一种自适应学习率优化算法，旨在解决传统的随机梯度下降（SGD）和相关自适应方法（如 AdaGrad）在训练深度学习模型时可能遇到的一些问题。

（ii）RMSProp

RMSProp是AdaDelta的一种扩展。当γ = 0.5 \gamma=0.5γ=0.5的时候就变成了RMSProp。但是RMSProp仍然依赖于全局学习率。效果介于AdaGrad和AdaDelta之间。

1. 损失函数：简单的理解就是每一个样本经过模型后会得到一个预测值，然后得到的预测值和真实值的差值就成为损失（当然损失值越小证明模型越是成功），我们知道有许多不同种类的损失函数，这些函数本质上就是计算预测值和真实值的差距的一类型函数，然后经过库（如pytorch，tensorflow等）的封装形成了有具体名字的函数。

损失函数一般分为5种：

1.HingeLoss 0-1 损失函数：感知机就是用的这种损失函数；

2.对数损失函数：逻辑回归的损失函数就是log对数损失函数；

3.MSE平方损失函数：线性回归的损失函数就是MSE；

4.Hinge 损失函数：SVM就是使用这个损失函数；

5.交叉熵损失函数 ：逻辑回归损失函数，用sigmoid作为激活函数，常用于二分类和多分类问题中。

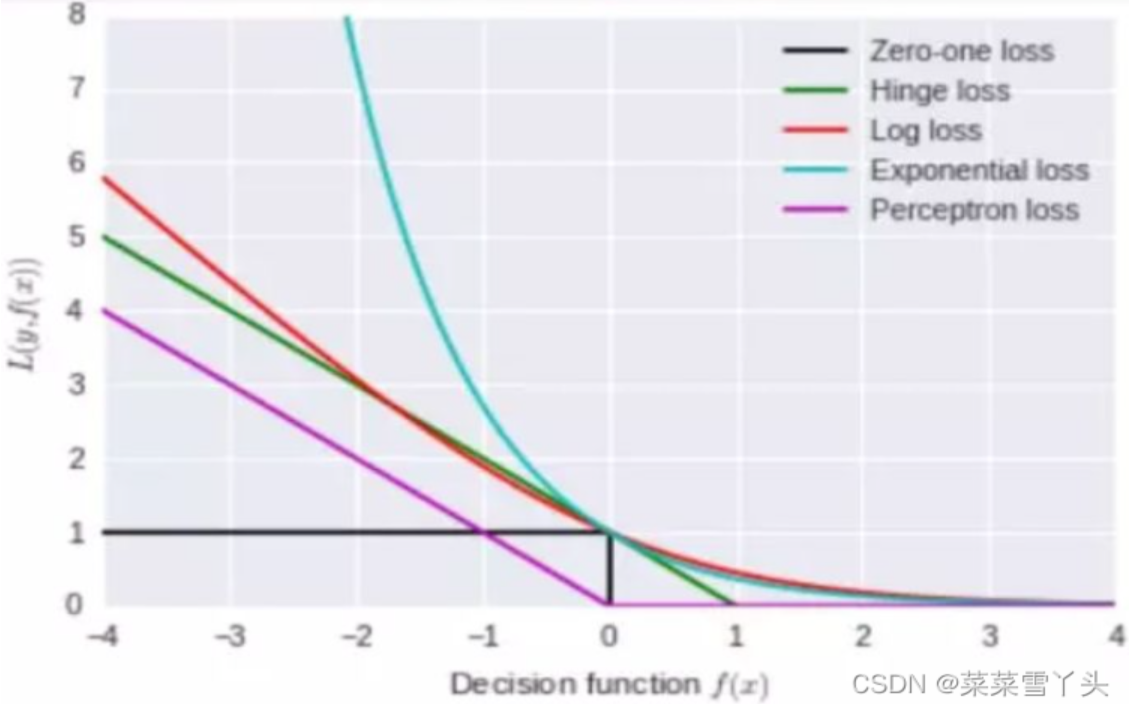


图 6.4 损失函数对比图

# 6.5结果展示和分析

我们使用了普通的神经网络和卷积神经网络分别对该数据集进行测试，并对两种模型进行对比，效果如下所示：

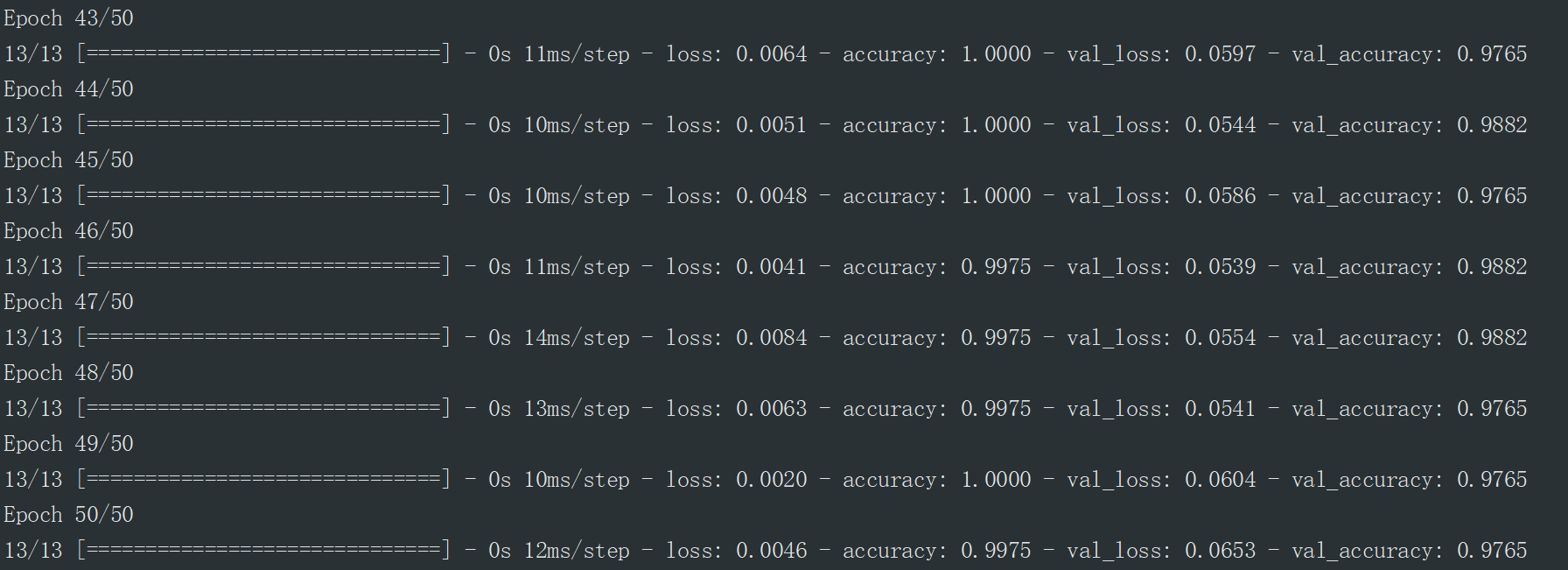


图 6.5 训练示意图

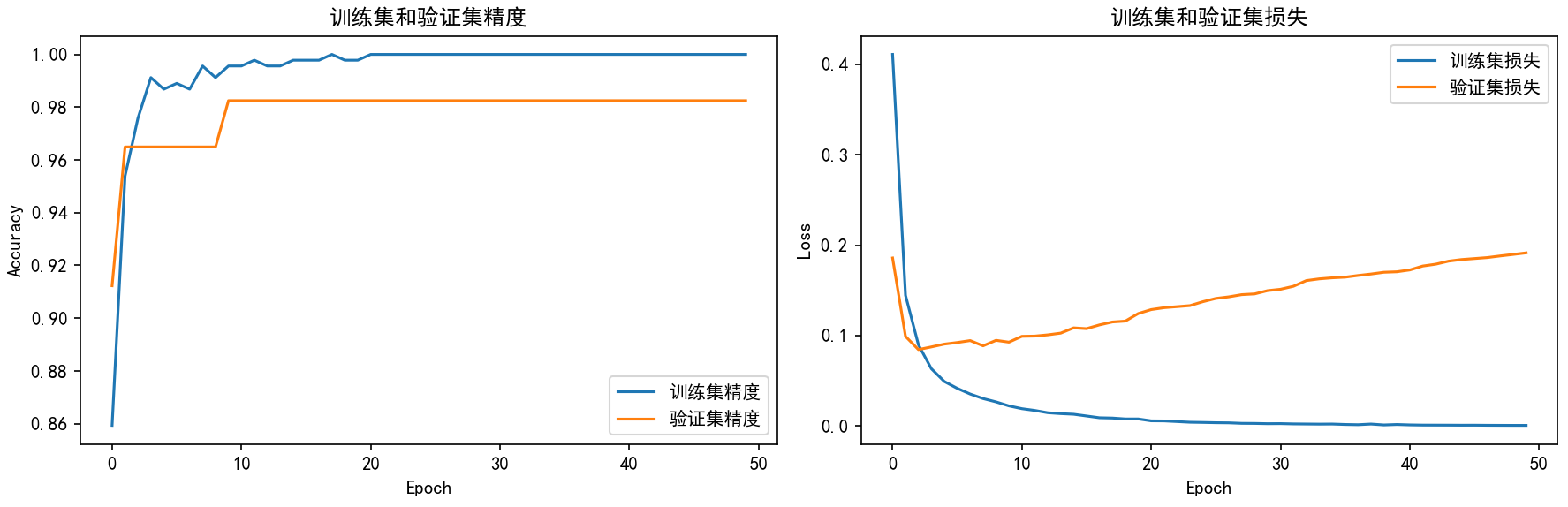
c

图 6.6 普通神经网络中精度和损失

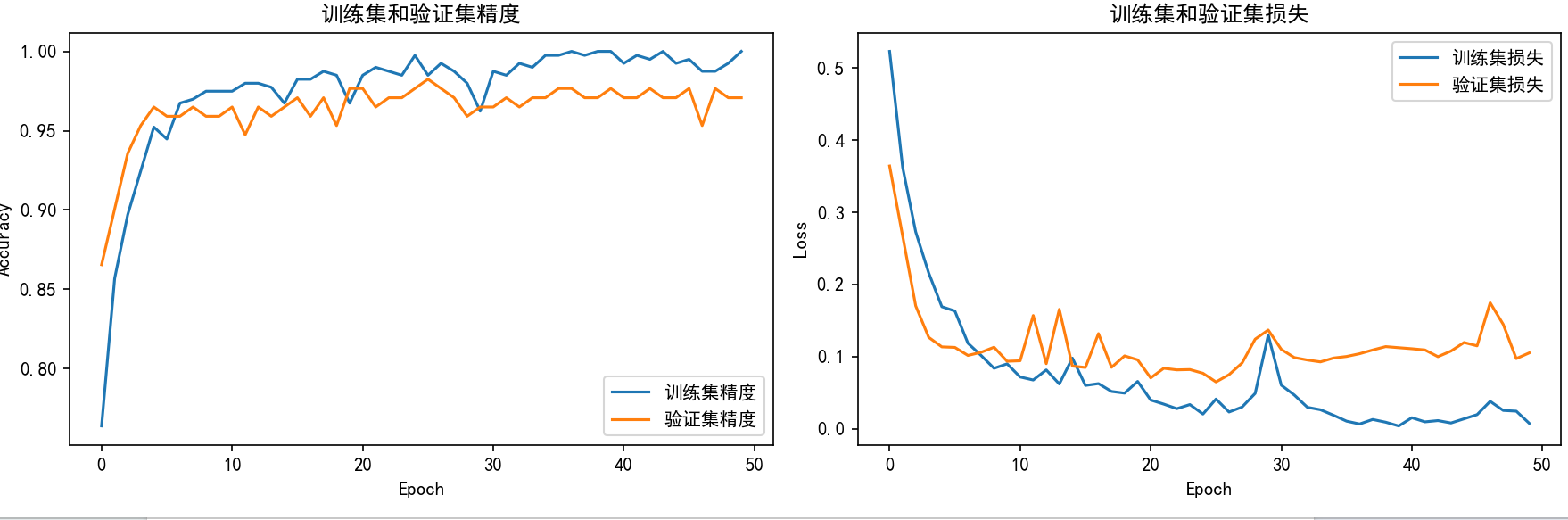


图 6.7 卷积神经网络中精度和损失

根据模型中，普通神经网络输出结果：测试集损失: 0.3960，测试集精度: 0.9419

卷积神经网络 (CNN) 结果：测试集损失: 0.2245，测试集精度: 0.9535。CNN 的测试集损失和精度均优于普通神经网络，表明 CNN 在处理这个特定数据集（乳腺癌数据集）时，能更有效地学习特征并进行分类。普通神经网络虽然精度也相对较高，但相较于 CNN 仍有提升空间。

根据两模型的精度和损失随着训练轮次的增加，我们发现两模型中精度都随之上升，损失都随之下降，说明模型能够在训练过程中学习到有效知识。模型在精度上CNN得训练集和验证集差别更小，模型能够很好地学习到数据中的规律。同时，我们发现在普通神经网络的损失函数中，验证集很快出现了上升的现象，说明出现了过拟合的现象，然而在卷积神经网络中拟合程度较好。

普通神经网络是一个简单的全连接模型，适合处理线性关系，结构较为简单。卷积神经网络则通过多层卷积和池化操作，能够捕捉更复杂的特征。

## CNN模型的优化

**7.1模型训练优化策略**

在本节中，我们探讨了不同的模型训练优化策略，包括交叉验证、早停（Early Stopping）、以及学习率调度器（Learning Rate Scheduler）的应用。这些策略旨在提高模型的泛化能力，减少过拟合，并加速模型的收敛。

**交叉验证：**

交叉验证通过将数据集划分为多个子集，使得每个子集都有机会作为验证集，从而提供了模型性能的更稳健估计。这种方法有助于评估模型在不同数据子集上的表现，确保模型的稳定性和可靠性。

**早停：**

早停策略通过监控验证集的性能，在模型性能不再提升时提前结束训练。这有助于避免过拟合，节省训练时间，并可能提高模型在未见数据上的表现。

**学习率调度器：**

学习率调度器在训练过程中动态调整学习率，以帮助模型更快地收敛到最优解。在训练初期使用较大的学习率可以加快收敛速度，而在训练后期减小学习率有助于模型进行更精细的调整。

**7.2模型优化策略实现**

**交叉验证实现：**

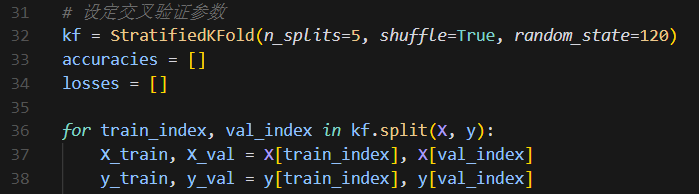


图7.1交叉验证代码

在模型训练过程中，我们采用了交叉验证方法来评估模型的泛化能力。交叉验证通过将数据集划分为多个子集，使得每个子集都有机会作为验证集，从而提供了模型性能的更稳健估计。具体实现如下：

首先，我们设置了交叉验证的参数，选择了5折交叉验证（n\_splits=5），并启用了数据乱序（shuffle=True），以确保每次分割的随机性，random\_state=120 用于保证结果的可复现性。

接着，我们初始化了两个空列表，accuracies 和 losses，用于存储每一折交叉验证的精度和损失值。

通过循环遍历由 kf.split(x, y) 生成的训练集和验证集索引，我们对数据进行了分割，并将分割后的数据分别赋值给 X\_train, X\_val, y\_train, y\_val，以便于后续的训练和验证过程。

**早停、学习率调度器实现：**

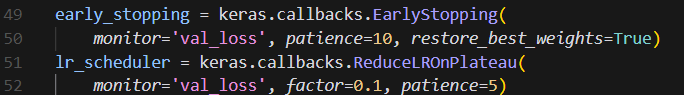


图7.2早停、学习率调度器代码

早停策略用于防止模型过拟合。当验证集的损失在连续10个epoch（patience=10）内没有显著下降时，训练将被提前终止。此外，该策略还会恢复到最佳权重，即在验证集上表现最好的模型状态。

学习率调度器用于在训练过程中动态调整学习率。当验证集的损失在连续5个epoch（patience=5）内没有显著下降时，学习率将按照0.1的因子减少，以期望跳出局部最优，继续训练寻找更好的解。

**7.3对比结果展示和分析部分**

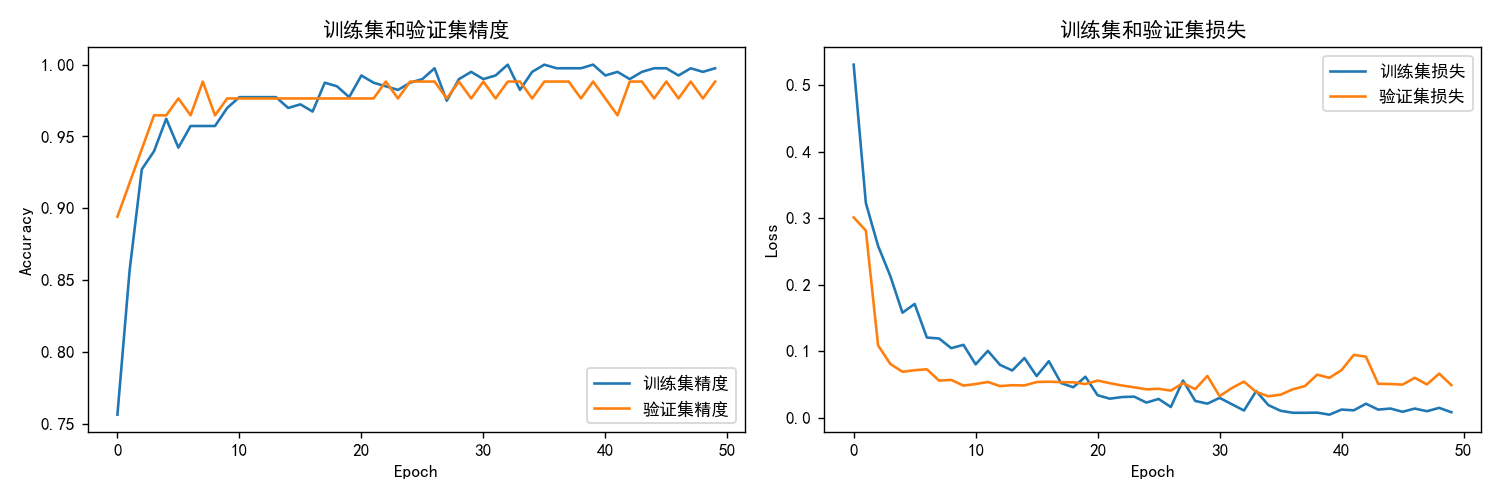
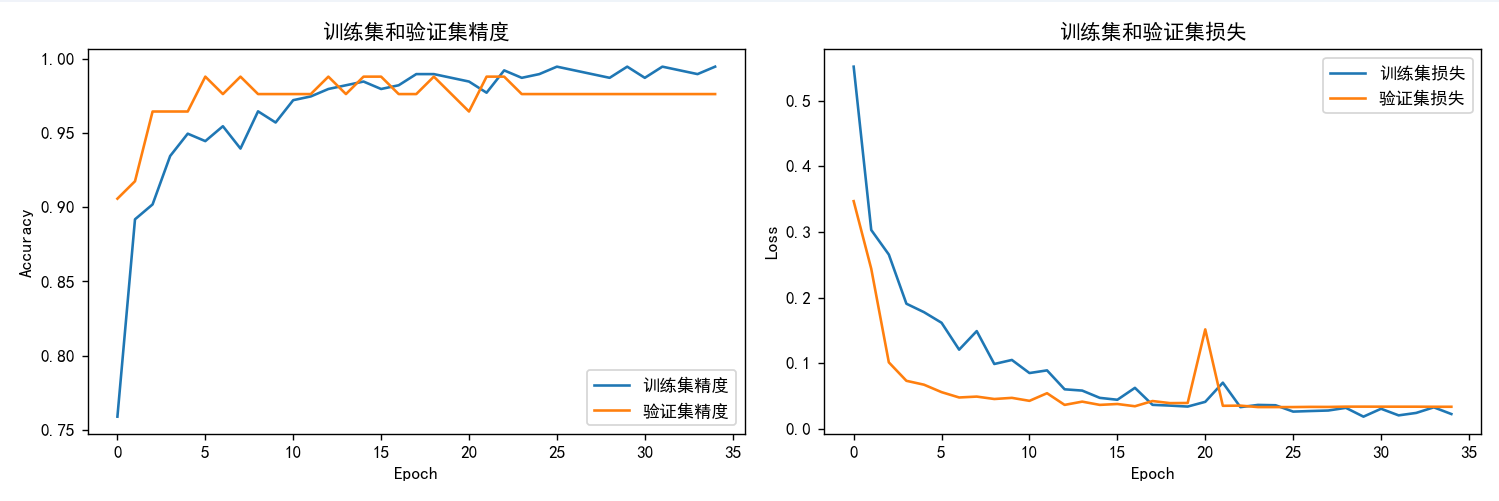
****

图7.3卷积神经网络中精度和损失

图7.3展示了初始模型在测试集上的表现。初始模型的测试集损失为0.2456，精度为0.9686。这为我们提供了一个基准，以便我们评估其他优化策略的效果。

****

**图7.4增加早停、学习率调度器后**卷积神经网络中精度和损失

图7.4展示了在初始模型基础上增加早停和学习率调度器后的模型表现。测试集损失为0.1555，精度为0.9551。早停策略通过在训练过程中监测验证集的性能，当验证集的性能不再提升时停止训练，有效防止了模型的过拟合。学习率调度器则通过动态调整学习率，使模型在训练初期能够快速收敛，而在训练后期逐步减小学习率，从而更精细地调整模型参数，避免陷入局部最优。结合这两种策略，虽然模型的精度略有下降（从0.9686降至0.9551），但是模型在训练过程中能够更有效地利用数据，减少不必要的训练epoch，提高训练效率。

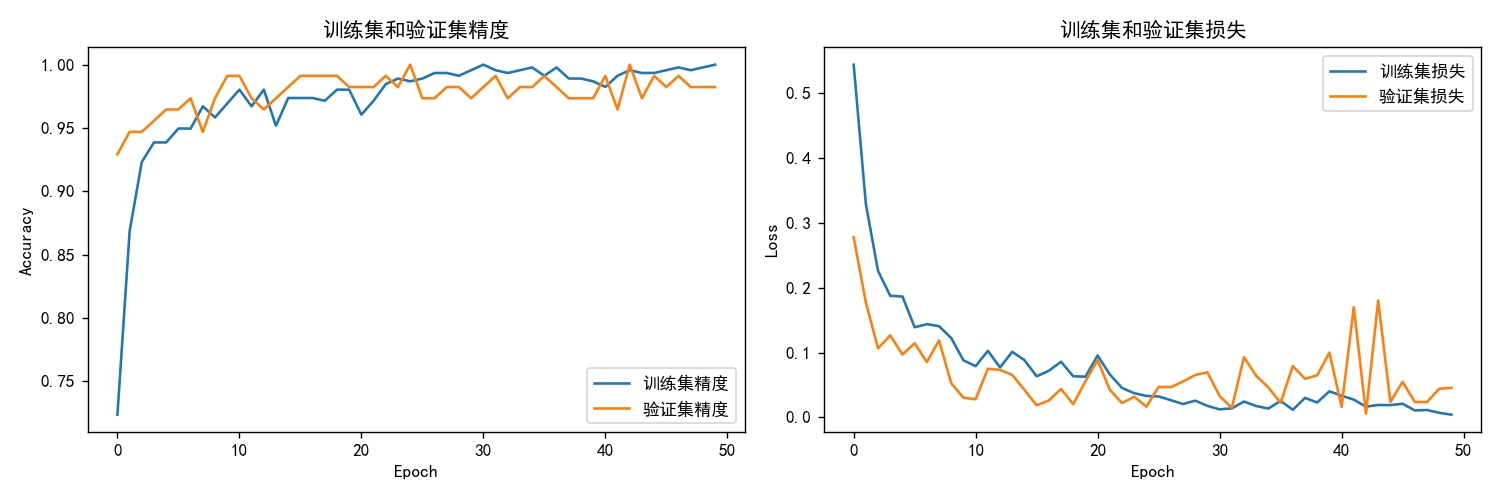


图7.5增加交叉验证后卷积神经网络中精度和损失

图7.5展示了在初始模型基础上增加交叉验证后的模型性能。测试集损失进一步降低至0.0380，精度提高至0.9884。交叉验证通过将数据集划分为多个子集，并在不同的子集上进行训练和验证，确保了模型在不同数据分布下的稳健性。这种方法不仅提高了模型的泛化能力，还减少了因数据划分方式不同而导致的性能波动，使得模型在实际应用中更具可靠性。

综合来看，早停、学习率调度器和交叉验证这三种策略的结合使用，显著提高了模型的训练效率和预测准确性。早停策略有效防止了过拟合，学习率调度器优化了训练过程，而交叉验证则增强了模型的稳定性和泛化能力。这些技术的协同作用，不仅提升了模型在乳腺癌检测任务中的性能，也为其他深度学习模型的训练提供了一种高效且可靠的方法。在实际应用中，通过进一步优化这些策略的参数设置和组合方式，有望进一步提升模型的表现，为乳腺癌的早期检测和诊断提供更为准确的依据，具有重要的临床应用价值和研究意义。