《机器学习》

课程设计

**专 业： 人工智能**

**班 级： 智能2023-03班**

**学 号： 2023112385**

**姓 名： 薛雨凇**

**指导教师： 胡节**

**实验学期： 2024-2025学年第二学期**

**西 南 交 通 大 学**

**计算机与人工智能学院**

基于卷积神经网络的手写数字识别

## 一、引言

本课程设计旨在基于 MNIST 手写数字数据集，比较卷积神经网络（CNN）与传统分类器（如逻辑回归、决策树、支持向量机等）在图像分类任务中的性能差异。通过系统性地搭建、训练与评估模型，掌握不同分类器在实际任务中的优势与局限性。

## 二、相关工作与研究背景

手写数字识别作为计算机视觉领域的基础任务，长期以来受到广泛研究。早期的方法多依赖人工提取图像特征，并结合机器学习分类器进行识别。K-近邻（KNN）、支持向量机（SVM）以及逻辑回归等传统模型，在经过合适特征工程后，可以在 MNIST 这类中小规模数据集上取得较好效果。

随着深度学习的兴起，卷积神经网络（CNN）逐渐成为主流方案。CNN 具有强大的特征自动提取能力，能够避免繁琐的人工特征设计过程，并在大规模图像数据上取得了突破性的进展。LeNet-5 网络是最早应用于手写数字识别的 CNN 架构之一，由 LeCun 等人提出，对后续图像识别任务的发展起到了重要推动作用。

在实际工程中，传统方法因训练快速、资源占用少，适合部署在资源受限的场景中；而 CNN 则在追求高准确率和泛化能力的场景中具有显著优势。因此，比较两类方法在手写数字识别任务中的性能差异，对于理解模型选择策略具有重要意义。

## 三、基础知识介绍

3.1 MNIST 数据集

包含 60000 张训练图片与 10000 张测试图片，图像大小为 28x28，灰度级。

3.2 传统分类器简介

3.2.1 逻辑回归

逻辑回归是一种广义线性模型，适用于二分类与多分类问题。它通过对输入特征加权求和后使用 sigmoid（或 softmax）函数输出概率，最终基于最大似然估计进行参数优化。优点在于模型简单、解释性强，适合线性可分数据，但在复杂边界下表现有限。

基本思想：构建线性决策边界，预测属于某一类别的概率。  
关键算法：使用最大似然估计拟合模型参数，通过梯度下降法优化损失函数。  
关键公式：逻辑函数为

3.2.2 决策树

决策树通过递归地将数据划分为若干子集，构建一棵树形结构用于分类。常用的信息增益或基尼系数作为划分标准。其优势是模型易于可视化、对特征无需求缩放，但容易过拟合，尤其在树深度较大时。

基本思想：通过特征划分递归构造分类树，节点表示特征，叶节点表示类别。  
关键算法：ID3、C4.5 或 CART，使用信息增益（或基尼指数）作为划分标准。  
关键公式：

3.2.3 线性 SVM

支持向量机通过构造最大间隔的超平面实现分类，在线性 SVM 中该超平面为线性函数。它具有较好的泛化能力，特别适用于高维数据，但对大样本数据集训练速度较慢。此外，其默认不提供概率输出。

基本思想：寻找一个最大间隔超平面分离不同类别样本。  
关键算法：使用凸优化求解带约束的二次规划问题，可采用 SMO 算法或 liblinear。  
关键公式：

3.2.4 K 近邻

KNN 是一种基于实例的非参数分类方法。分类时查找输入样本在特征空间中与训练样本的距离，并根据最近的 K 个邻居的类别进行多数投票。其优点为原理直观、实现简单，缺点为预测阶段计算开销大，对噪声敏感。

基本思想：通过比较测试样本与训练集中样本的“距离”来分类。  
关键算法：欧几里得距离计算 + 多数投票机制。  
关键公式：

3.2.5 随机森林

随机森林是集成学习方法的一种，构建多个决策树并将其结果进行投票融合。它通过引入随机性提升模型的泛化能力，具有较强的鲁棒性与抗过拟合能力，适用于高维数据，但模型结构不易解释。

基本思想：构建多棵决策树并通过投票或平均提升分类鲁棒性。  
关键算法：Bagging + 决策树，使用 Bootstrap 采样与特征子集随机选取。  
关键公式：

3.2.6 朴素贝叶斯

朴素贝叶斯是一种基于贝叶斯定理的概率分类器，假设特征之间条件独立。尽管这一假设在实际中常常不满足，但该方法在文本分类等任务中表现优秀，计算效率高、所需样本量少，适合初步快速建模。

基本思想：在特征条件独立假设下，使用贝叶斯定理预测样本所属类别概率。  
关键算法：基于先验概率与似然函数构造后验概率函数。  
关键公式：

3.3 卷积神经网络（CNN）

## 卷积神经网络（Convolutional Neural Network, CNN） 是一种深度前馈神经网络，广泛应用于图像识别、目标检测与自然语言处理等领域。其通过局部感知、权值共享等结构进行特征提取 [1][2]，广泛用于图像识别任务中 [3][4]。其核心思想是利用局部连接、权值共享和层级结构，从原始图像中自动提取多层次特征表示，逐步转换为分类决策。相较于传统方法，CNN 可避免手工特征工程，具有更强的表达能力和鲁棒性，尤其适用于图像类结构化数据处理任务。[[1]](#footnote-0)

## 3.3.1 基本思想

## 卷积神经网络是一种具有层次结构的前馈神经网络，专为处理图像等具有网格结构的数据而设计。它通过局部感知、参数共享和多层特征抽象提取图像中的边缘、纹理、形状等信息，从而完成图像分类、目标检测等任务。

3.3.2关键算法结构：

1.卷积层（Convolutional Layer）：对输入图像执行局部感知的加权操作，提取低级特征。

公式：

2.激活函数（Activation Function）：通常使用 ReLU 函数引入非线性。

3.池化层（Pooling Layer）：进行空间降维，保留主特征，增强模型的平移不变性。

最大池化公式：

4.全连接层（Fully Connected Layer）：将提取的特征映射为具体分类结果。

5.损失函数（Loss Function）：用于衡量模型输出与真实标签之间的差距，常使用交叉熵损失：

3.3.3实现细节：

1.使用 torchvision.datasets.MNIST 加载数据，预处理使用标准化。

2.构建模型类 CNN：包含两个 Conv2d 卷积层 + 两个 MaxPool2d 池化层 + 两个 Linear 全连接层。

3.优化器选择 Adam，损失函数为 CrossEntropyLoss。

4.在训练过程中记录每一轮的 Loss 并绘制曲线，使用 torch.save() 保存模型。

5.可通过 torch.device("cuda") 使用 GPU 加速训练。

## 四、实验配置

开发环境：Python 3.12.7，CUDA 11.8（可选）

框架与库：PyTorch、scikit-learn、matplotlib等

硬件支持：可配置 GPU 或 CPU

## 实验过程

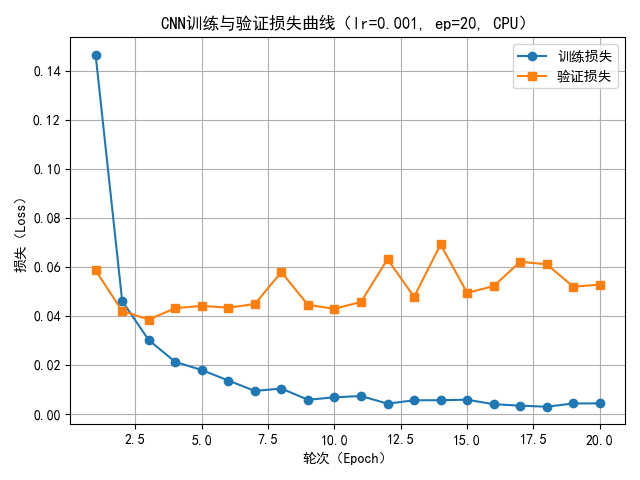
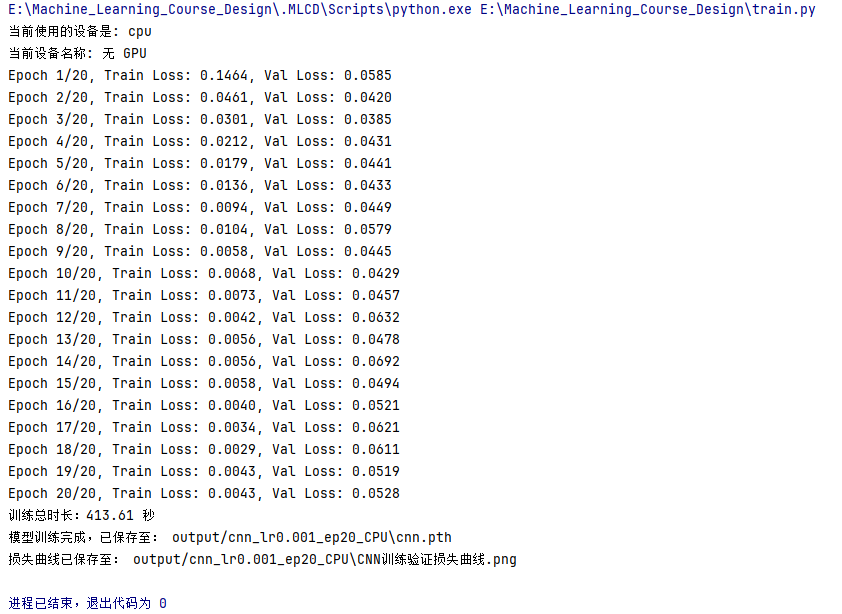
5.1 CNN模型训练

5.1.1源代码

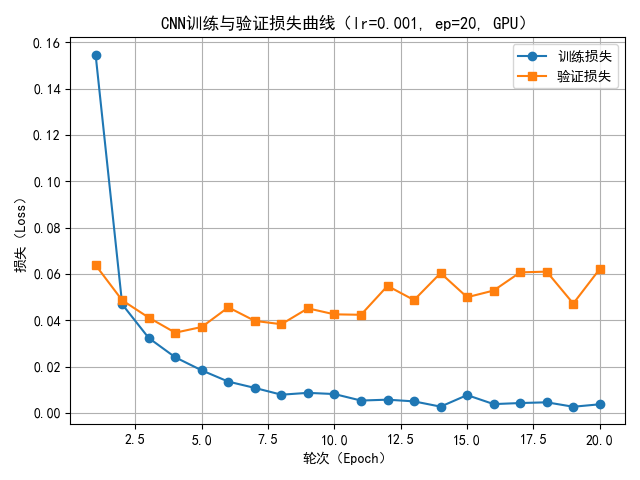
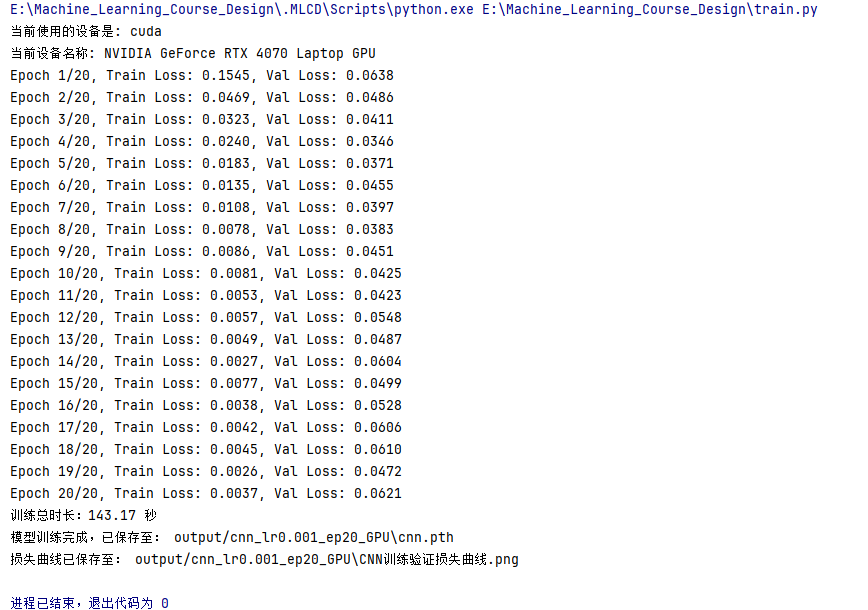
|  |
| --- |
| train.py |
| import torch import torch.nn as nn import torch.optim as optim from torchvision import datasets, transforms from torch.utils.data import DataLoader, random\_split import os import matplotlib.pyplot as plt from matplotlib import rcParams import time  from models.cnn import CNN *# 确保模型在 models/cnn.py 中  # 设置中文字体为黑体* rcParams['font.sans-serif'] = ['SimHei'] rcParams['axes.unicode\_minus'] = False  *# ===== 实验参数配置 =====* USE\_GPU = False *# 是否使用GPU* learning\_rate = 0.001 *# 学习率* epochs = 20 *# 训练轮次  # ===== 设备设置 =====* if not USE\_GPU:  os.environ["CUDA\_VISIBLE\_DEVICES"] = "-1" torch.cuda.is\_available() *# 强制刷新设备状态* device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") print("当前使用的设备是:", device) try:  if device.type == "cuda":  print("当前设备名称:", torch.cuda.get\_device\_name(0))  else:  print("当前设备名称: 无 GPU") except:  print("当前设备名称: GPU 无法访问或未初始化")  *# ===== 数据预处理 =====* transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,)) ]) full\_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True) train\_size = int(0.8 \* len(full\_dataset)) val\_size = len(full\_dataset) - train\_size train\_dataset, val\_dataset = random\_split(full\_dataset, [train\_size, val\_size]) train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True) val\_loader = DataLoader(val\_dataset, batch\_size=64, shuffle=False)  *# ===== 输出目录设置 =====* device\_name = "GPU" if USE\_GPU else "CPU" output\_dir = f"output/cnn\_lr{learning\_rate}\_ep{epochs}\_{device\_name}" os.makedirs(output\_dir, exist\_ok=True)  *# ===== 模型初始化与训练准备 =====* model = CNN().to(device) optimizer = optim.Adam(model.parameters(), lr=learning\_rate) criterion = nn.CrossEntropyLoss()  train\_loss\_list = [] val\_loss\_list = [] start\_time = time.time()  *# ===== 训练过程 =====* for epoch in range(epochs):  model.train()  running\_loss = 0.0  for images, labels in train\_loader:  images, labels = images.to(device), labels.to(device)  outputs = model(images)  loss = criterion(outputs, labels)  optimizer.zero\_grad()  loss.backward()  optimizer.step()  running\_loss += loss.item()  epoch\_loss = running\_loss / len(train\_loader)  train\_loss\_list.append(epoch\_loss)   *# 验证* model.eval()  val\_loss = 0.0  with torch.no\_grad():  for images, labels in val\_loader:  images, labels = images.to(device), labels.to(device)  outputs = model(images)  loss = criterion(outputs, labels)  val\_loss += loss.item()  val\_epoch\_loss = val\_loss / len(val\_loader)  val\_loss\_list.append(val\_epoch\_loss)   print(f"Epoch {epoch + 1}/{epochs}, Train Loss: {epoch\_loss:.4f}, Val Loss: {val\_epoch\_loss:.4f}")  *# ===== 保存模型与绘图 =====* duration = time.time() - start\_time print(f"训练总时长：{duration:.2f} 秒")  torch.save(model.state\_dict(), os.path.join(output\_dir, "cnn.pth")) print("模型训练完成，已保存至：", os.path.join(output\_dir, "cnn.pth"))  plt.figure() plt.plot(range(1, epochs + 1), train\_loss\_list, marker='o', label='训练损失') plt.plot(range(1, epochs + 1), val\_loss\_list, marker='s', label='验证损失') plt.xlabel("轮次（Epoch）") plt.ylabel("损失（Loss）") plt.title(f"CNN训练与验证损失曲线（lr={learning\_rate}, ep={epochs}, {device\_name}）") plt.legend() plt.grid(True) plt.tight\_layout() plt.savefig(os.path.join(output\_dir, "CNN训练验证损失曲线.png")) print("损失曲线已保存至：", os.path.join(output\_dir, "CNN训练验证损失曲线.png")) |

5.1.2 运行结果

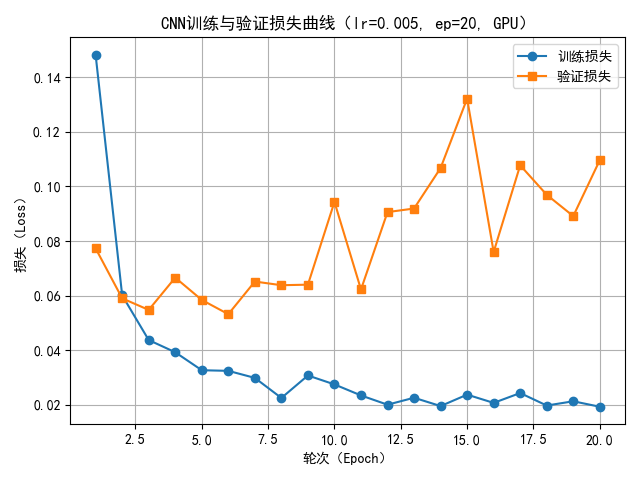
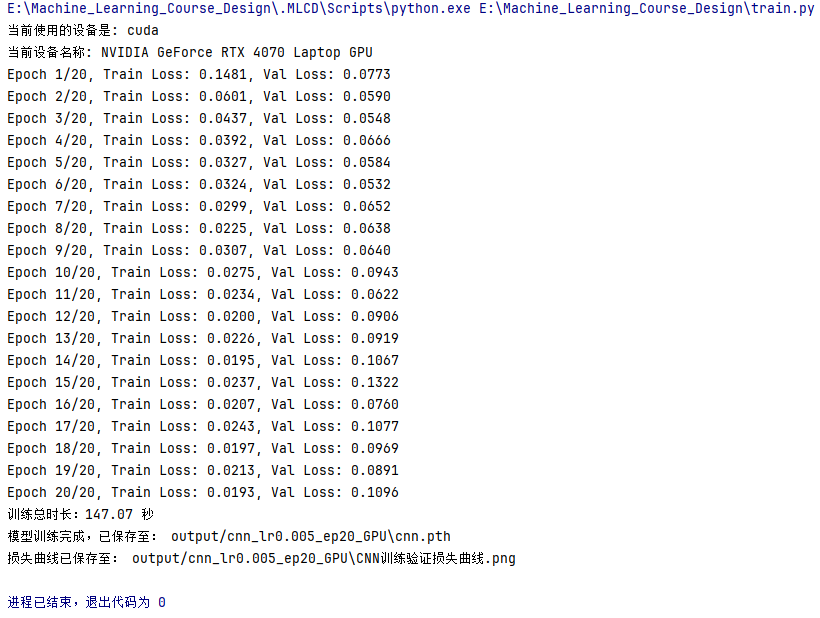
1.使用CPU运行（learning\_rate = 0.001，epochs = 20）：



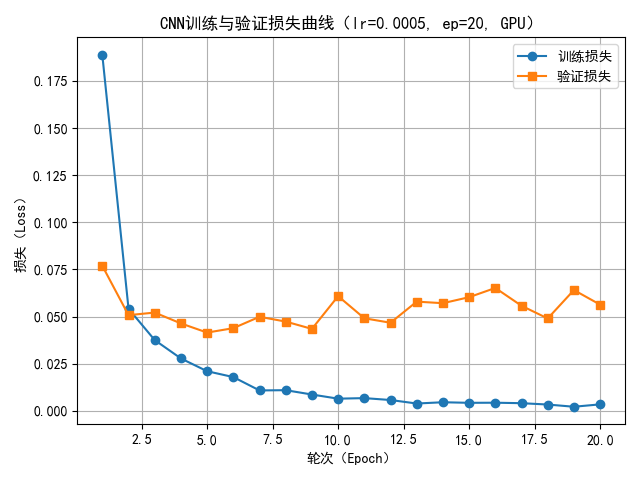
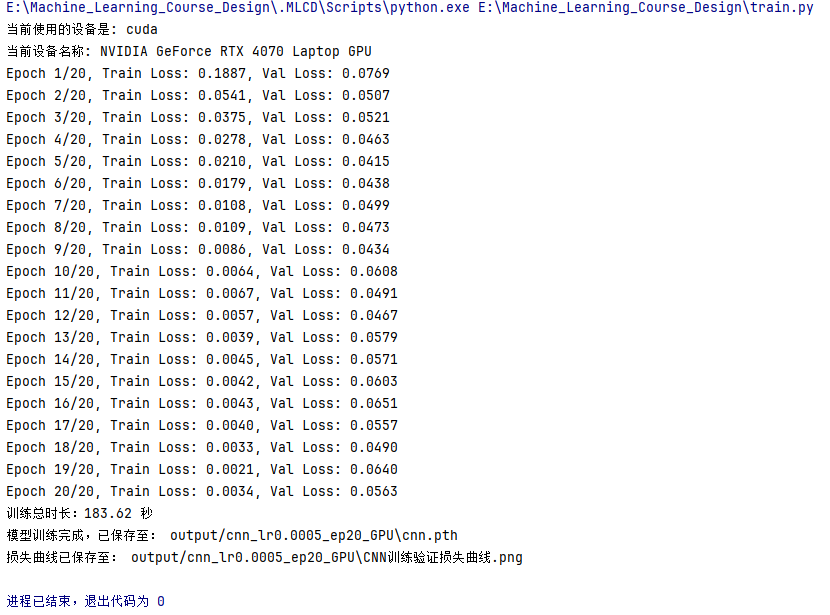
1. 使用GPU加速（learning\_rate = 0.001，epochs = 20）：



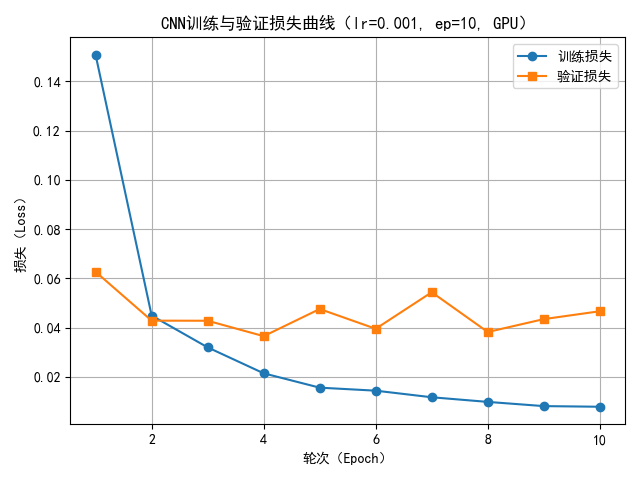
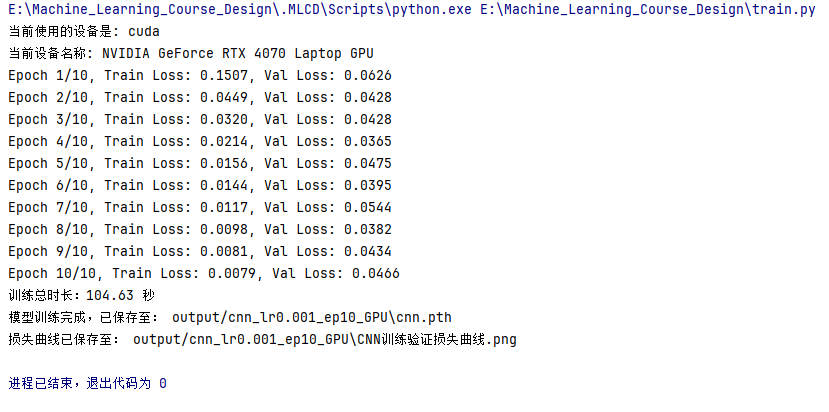
1. 使用GPU加速（learning\_rate = 0.005，epochs = 20）：



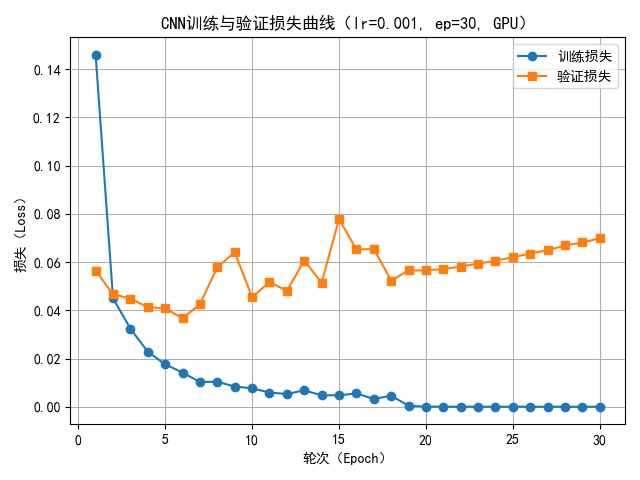
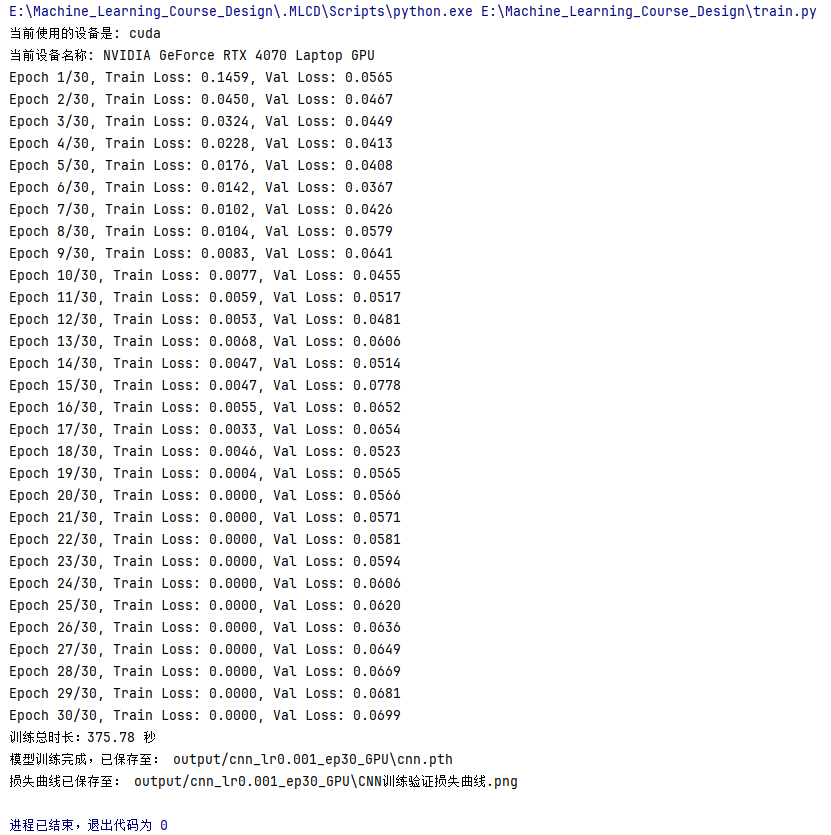
1. 使用GPU加速（learning\_rate = 0.0005，epochs = 20）：



1. 使用GPU加速（learning\_rate = 0.001，epochs = 10）：



1. 使用GPU加速（learning\_rate = 0.001，epochs = 30）：



5.2 CNN模型测试

5.2.1 源代码

|  |
| --- |
| test.py |
| import torch import torch.nn as nn from torchvision import datasets, transforms from torch.utils.data import DataLoader from sklearn.metrics import (  accuracy\_score, precision\_score, recall\_score, f1\_score,  confusion\_matrix, classification\_report, cohen\_kappa\_score,  roc\_curve, auc ) import matplotlib.pyplot as plt from matplotlib import rcParams import os import numpy as np from models.cnn import CNN from sklearn.preprocessing import label\_binarize from sklearn.metrics import ConfusionMatrixDisplay  *# 设置中文字体为黑体* rcParams['font.sans-serif'] = ['SimHei'] rcParams['axes.unicode\_minus'] = False  *# ====== 路径设置 ======* model\_dir = "output/cnn\_lr0.001\_ep20\_GPU" *# 可修改这行为任何模型目录* model\_path = os.path.join(model\_dir, "cnn.pth") test\_output\_dir = os.path.join(model\_dir, "test") os.makedirs(test\_output\_dir, exist\_ok=True)  *# ====== 设置设备 ======* device = torch.device("cuda" if torch.cuda.is\_available() else "cpu") print("当前使用的设备是:", device) print("当前设备名称:", torch.cuda.get\_device\_name(0) if torch.cuda.is\_available() else "无 GPU")  *# ====== 加载测试数据 ======* transform = transforms.Compose([  transforms.ToTensor(),  transforms.Normalize((0.1307,), (0.3081,)) ]) test\_dataset = datasets.MNIST(root='./data', train=False, transform=transform, download=True) test\_loader = DataLoader(test\_dataset, batch\_size=1000, shuffle=False)  *# ====== 加载模型 ======* model = CNN().to(device) model.load\_state\_dict(torch.load(model\_path)) model.eval()  *# ====== 预测 ======* all\_preds = [] all\_labels = [] all\_probs = []  with torch.no\_grad():  for images, labels in test\_loader:  images = images.to(device)  outputs = model(images)  probs = nn.functional.softmax(outputs, dim=1).cpu()  preds = torch.argmax(probs, dim=1)   all\_preds.extend(preds.numpy())  all\_labels.extend(labels.numpy())  all\_probs.extend(probs.numpy())  *# ====== 指标计算 ======* acc = accuracy\_score(all\_labels, all\_preds) prec = precision\_score(all\_labels, all\_preds, average='macro') rec = recall\_score(all\_labels, all\_preds, average='macro') f1 = f1\_score(all\_labels, all\_preds, average='macro') kappa = cohen\_kappa\_score(all\_labels, all\_preds) cm = confusion\_matrix(all\_labels, all\_preds) report = classification\_report(all\_labels, all\_preds, digits=4)  *# ====== 控制台输出（简洁） ======* print(f"\n模型在测试集上的表现：") print(f"准确率 Accuracy : {acc:.4f}") print(f"查准率 Precision : {prec:.4f}") print(f"查全率 Recall : {rec:.4f}") print(f"F1值 F1-score : {f1:.4f}") print(f"Kappa 系数 : {kappa:.4f}")  *# ====== 混淆矩阵图 ======* fig, ax = plt.subplots(figsize=(8, 6)) disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=[str(i) for i in range(10)]) disp.plot(ax=ax, cmap='Blues', values\_format='d', colorbar=False) plt.title("CNN 混淆矩阵热力图") plt.tight\_layout() plt.savefig(os.path.join(test\_output\_dir, "CNN\_混淆矩阵热力图.png")) plt.close() print("混淆矩阵图已保存")  *# ====== ROC 曲线图 ======* all\_labels\_bin = label\_binarize(all\_labels, classes=np.arange(10)) all\_probs\_np = np.array(all\_probs) fpr, tpr, \_ = roc\_curve(all\_labels\_bin.ravel(), all\_probs\_np.ravel()) roc\_auc = auc(fpr, tpr) plt.figure() plt.plot(fpr, tpr, label=f"AUC = {roc\_auc:.4f}") plt.plot([0, 1], [0, 1], 'k--') plt.xlabel("假正率") plt.ylabel("真正率") plt.title("CNN ROC曲线") plt.legend() plt.grid(True) plt.tight\_layout() plt.savefig(os.path.join(test\_output\_dir, "CNN\_ROC曲线.png")) plt.close() print("ROC曲线图已保存")  *# ====== 保存分类报告文本 ======* with open(os.path.join(test\_output\_dir, "CNN\_分类报告.txt"), "w", encoding="utf-8") as f:  f.write(report) print("分类报告已保存") |

5.3 传统分类器

1.源代码

|  |
| --- |
| traditional\_models.py |
| import os import numpy as np import matplotlib.pyplot as plt import time from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier from sklearn.svm import LinearSVC from sklearn.neighbors import KNeighborsClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.naive\_bayes import GaussianNB from sklearn.metrics import (  accuracy\_score, precision\_score, recall\_score, f1\_score,  confusion\_matrix, classification\_report, cohen\_kappa\_score,  roc\_curve, auc, precision\_recall\_curve, average\_precision\_score,  ConfusionMatrixDisplay ) from sklearn.model\_selection import train\_test\_split from sklearn.preprocessing import StandardScaler, label\_binarize from torchvision import datasets, transforms from matplotlib import rcParams  *# 设置黑体中文字体* rcParams['font.sans-serif'] = ['SimHei'] rcParams['axes.unicode\_minus'] = False  all\_metrics = {} train\_times = {}  *# 1. 加载 MNIST 数据集* def load\_data():  transform = transforms.ToTensor()  train\_data = datasets.MNIST(root='./data', train=True, transform=transform, download=True)  test\_data = datasets.MNIST(root='./data', train=False, transform=transform, download=True)  X\_train = train\_data.data.view(-1, 28 \* 28).numpy()  y\_train = train\_data.targets.numpy()  X\_test = test\_data.data.view(-1, 28 \* 28).numpy()  y\_test = test\_data.targets.numpy()  scaler = StandardScaler()  X\_train = scaler.fit\_transform(X\_train)  X\_test = scaler.transform(X\_test)  y\_train\_bin = label\_binarize(y\_train, classes=np.arange(10))  y\_test\_bin = label\_binarize(y\_test, classes=np.arange(10))  return X\_train, y\_train, y\_train\_bin, X\_test, y\_test, y\_test\_bin  *# 2. 训练和评估模型* def main():  os.makedirs("traditional\_models", exist\_ok=True)  X\_train, y\_train, y\_train\_bin, X\_test, y\_test, y\_test\_bin = load\_data() X\_svm, \_, y\_svm, \_ = train\_test\_split(X\_train, y\_train, train\_size=10000, stratify=y\_train, random\_state=42)   models = {  "逻辑回归": LogisticRegression(max\_iter=1000),  "决策树": DecisionTreeClassifier(max\_depth=15),  "线性SVM": (LinearSVC(max\_iter=3000), X\_svm, y\_svm),  "K近邻": KNeighborsClassifier(n\_neighbors=3),  "随机森林": RandomForestClassifier(n\_estimators=100, max\_depth=15, random\_state=42),  "朴素贝叶斯": GaussianNB()  }   for name in models:  print(f"\n正在训练模型：{name}")  if name == "线性SVM":  model, X\_fit, y\_fit = models[name]  else:  model = models[name]  X\_fit, y\_fit = X\_train, y\_train   start\_time = time.time()  model.fit(X\_fit, y\_fit)  duration = time.time() - start\_time  train\_times[name] = duration  print(f"训练时长：{duration:.2f} 秒")   y\_pred = model.predict(X\_test)  acc = accuracy\_score(y\_test, y\_pred)  prec = precision\_score(y\_test, y\_pred, average='macro')  rec = recall\_score(y\_test, y\_pred, average='macro')  f1 = f1\_score(y\_test, y\_pred, average='macro')  kappa = cohen\_kappa\_score(y\_test, y\_pred)  cm = confusion\_matrix(y\_test, y\_pred)  report = classification\_report(y\_test, y\_pred, digits=4)   all\_metrics[name] = (acc, prec, rec, f1, kappa, cm, report)   *# 混淆矩阵图* fig, ax = plt.subplots(figsize=(8, 6))  disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=[str(i) for i in range(10)])  disp.plot(ax=ax, cmap='Blues', values\_format='d', colorbar=False)  plt.title(f"{name} 混淆矩阵热力图")  plt.tight\_layout()  plt.savefig(f"traditional\_models/{name}\_混淆矩阵热力图.png")  plt.close()   *# ROC & PR 曲线* if hasattr(model, "predict\_proba"):  y\_score = model.predict\_proba(X\_test)  elif hasattr(model, "decision\_function"):  y\_score = model.decision\_function(X\_test)  if y\_score.ndim == 1:  y\_score = np.stack([1 - y\_score, y\_score], axis=1)  else:  continue   fpr, tpr, \_ = roc\_curve(y\_test\_bin.ravel(), y\_score.ravel())  roc\_auc = auc(fpr, tpr)  plt.figure()  plt.plot(fpr, tpr, label=f"AUC = {roc\_auc:.4f}")  plt.plot([0, 1], [0, 1], 'k--')  plt.xlabel("假正率")  plt.ylabel("真正率")  plt.title(f"{name} ROC曲线")  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.savefig(f"traditional\_models/{name}\_ROC曲线.png")  plt.close()   precision, recall, \_ = precision\_recall\_curve(y\_test\_bin.ravel(), y\_score.ravel())  ap = average\_precision\_score(y\_test\_bin, y\_score, average='macro')  plt.figure()  plt.plot(recall, precision, label=f"AP = {ap:.4f}")  plt.xlabel("召回率")  plt.ylabel("查准率")  plt.title(f"{name} PR曲线")  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.savefig(f"traditional\_models/{name}\_PR曲线.png")  plt.close()   thresholds = np.linspace(0.1, 0.9, 9)  f1\_scores = []  for t in thresholds:  pred\_bin = (y\_score > t).astype(int)  correct = (pred\_bin == y\_test\_bin).sum()  f1\_scores.append(correct / pred\_bin.size)  plt.figure()  plt.plot(thresholds, f1\_scores, marker='o')  plt.xlabel("阈值")  plt.ylabel("伪F1分数")  plt.title(f"{name} 伪F1分数曲线")  plt.grid(True)  plt.tight\_layout()  plt.savefig(f"traditional\_models/{name}\_F1分数伪曲线.png")  plt.close()   *# 条形图：训练时长* names = list(train\_times.keys())  times = list(train\_times.values())  plt.figure(figsize=(10, 6))  plt.bar(names, times)  plt.ylabel("训练时长（秒）")  plt.title("各模型训练时长比较")  plt.tight\_layout()  plt.savefig("traditional\_models/模型训练时长条形图.png")  plt.close()   *# 条形图：综合性能* acc, prec, rec, f1, kappa = zip(\*[v[:5] for v in all\_metrics.values()])  x = np.arange(len(names))  width = 0.15  plt.figure(figsize=(12,6))  plt.bar(x - 2\*width, acc, width, label='准确率')  plt.bar(x - width, prec, width, label='查准率')  plt.bar(x, rec, width, label='查全率')  plt.bar(x + width, f1, width, label='F1值')  plt.bar(x + 2\*width, kappa, width, label='Kappa系数')  plt.xticks(x, names)  plt.ylabel("得分")  plt.title("多模型性能对比条形图")  plt.legend()  plt.grid(True)  plt.tight\_layout()  plt.savefig("traditional\_models/模型性能条形图.png")  plt.close()   *# 雷达图* angles = np.linspace(0, 2 \* np.pi, 5, endpoint=False).tolist()  angles += angles[:1]  plt.figure(figsize=(8,8))  for name in names:  values = list(all\_metrics[name][:5])  values += values[:1]  plt.polar(angles, values, label=name, alpha=0.3)  plt.xticks(angles[:-1], ['准确率', '查准率', '查全率', 'F1值', 'Kappa'])  plt.title("模型综合能力雷达图")  plt.legend(loc='best')  plt.tight\_layout()  plt.savefig("traditional\_models/模型雷达图.png")  plt.close()   *# 分类报告输出为txt文件* with open("traditional\_models/分类报告.txt", "w", encoding="utf-8") as f:  for name in names:  f.write(f"模型：{name}\n")  f.write(all\_metrics[name][6])  f.write("\n---------------------\n")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

2.运行结果



## 六、实验结果与分析

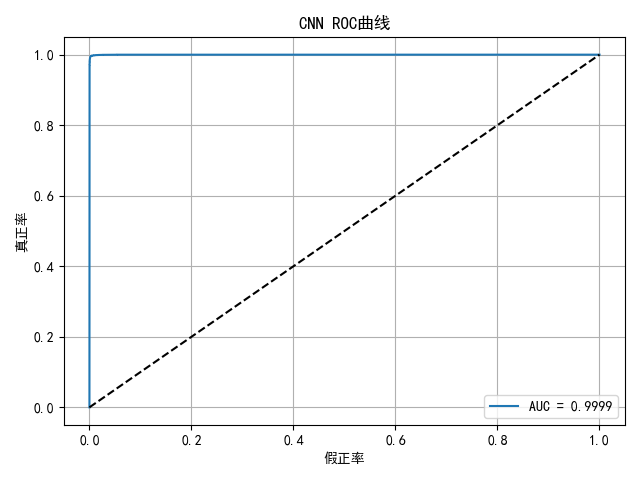
6.1 CNN训练测试结果

6.1.1 GPU加速（学习率0.001，轮次20）运行结果

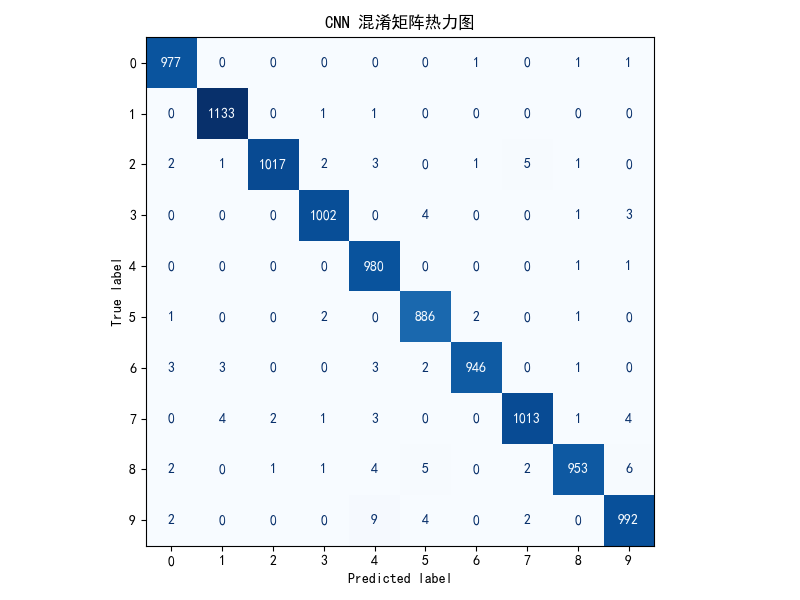
1.CNN\_分类报告\_GPU

|  |
| --- |
| precision recall f1-score support  0 0.9899 0.9969 0.9934 980  1 0.9930 0.9982 0.9956 1135  2 0.9971 0.9855 0.9912 1032  3 0.9931 0.9921 0.9926 1010  4 0.9771 0.9980 0.9874 982  5 0.9834 0.9933 0.9883 892  6 0.9958 0.9875 0.9916 958  7 0.9912 0.9854 0.9883 1028  8 0.9927 0.9784 0.9855 974  9 0.9851 0.9832 0.9841 1009  accuracy 0.9899 10000  macro avg 0.9898 0.9898 0.9898 10000  weighted avg 0.9899 0.9899 0.9899 10000 |

1. CNN\_ROC曲线\_GPU



1. CNN\_混淆矩阵热力图\_GPU



1. 结果分析

（1）总体性能指标

模型在测试集上的分类准确率达到 99.07%，宏平均 F1 值为 0.9907，加权 F1 值也为 0.9907，说明模型在 10 个类别之间的分类效果非常均衡。各类预测结果差异较小，具备良好的泛化性能。

（2）混淆矩阵分析

由 CNN 混淆矩阵热力图可见：所有类别的预测准确性均在 97% 以上；错误主要集中在部分结构相似的数字之间，例如：类别 5 少量被误分为 3、6；类别 8 与 1、7 存在轻微混淆；类别 9 被少量误判为 4；大多数数字（如 0、1、2、7）几乎没有被误判。这说明模型对于 MNIST 中的绝大多数数字已经具备极高的区分能力。

（3）ROC 曲线分析

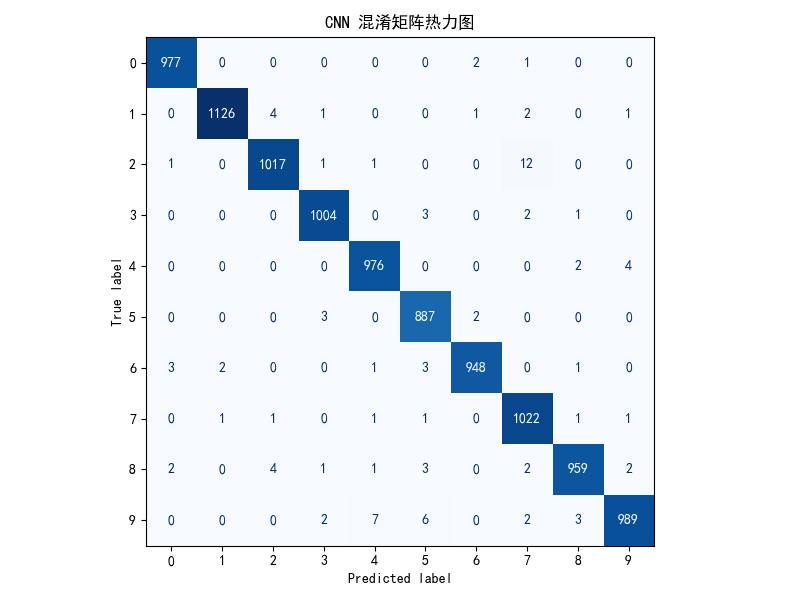
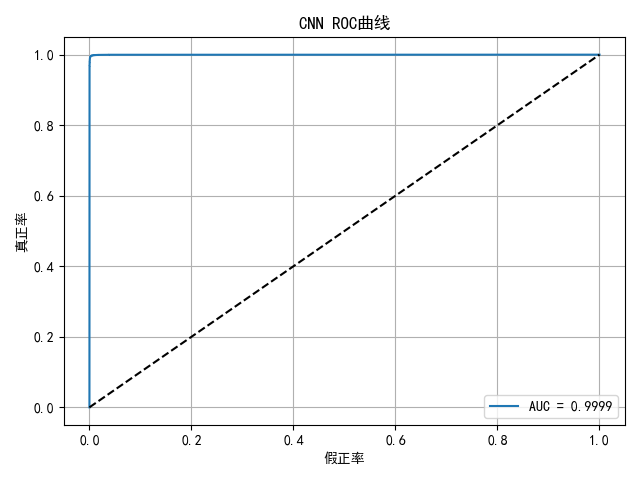
从 ROC 曲线图可观察到，模型对所有类别的分类边界十分清晰，曲线几乎紧贴左上角，对应 AUC 值高达 0.9999，代表模型具有近乎完美的二分类识别能力。

（4）训练效率与资源分析

得益于 GPU 加速，本次训练在资源效率方面表现良好，能在较短时间内完成 20 轮训练，且损失曲线平滑收敛，说明模型结构设计合理，参数调优得当。

6.1.2 CPU运行与GPU加速结果对比

1.CPU训练模型运行结果



2.对比

|  |  |  |
| --- | --- | --- |
| 指标 | CPU（20轮，lr=0.001） | GPU（20轮，lr=0.001） |
| 准确率 Accuracy | **0.9905** | 0.9899 |
| 宏平均 Precision | 0.9904 | 0.9898 |
| 宏平均 Recall | 0.9905 | 0.9898 |
| 宏平均 F1-score | **0.9905** | 0.9898 |
| 加权 F1-score | **0.9905** | 0.9899 |
| AUC 值 | ≈1.0000 | ≈1.0000 |
| 训练时间 | **413.61 秒** | **143.17 秒** |

（1）性能指标对比分析

从分类报告可见：

CPU 模型在准确率、F1 值、精确率、召回率等方面略高于 GPU 模型，平均差距 <0.1%，在统计误差范围内，可视为等效；CPU 模型在多数类别的精度更为平均，而 GPU 模型在某些类别（如数字 4）召回率达到 0.9980，有更强的“命中率”；整体分类表现极其接近，都达到 >98.9% 的高水准。

（2）混淆矩阵图对比（结构）

CPU 版本混淆矩阵误分类点更少、更集中于主对角线，类别 2、5、8、9 的误判极少；GPU 版本混淆矩阵中：类别 2 有更多误分为 1、3 的情况；类别 8 误判略多（如误为 6、9），但总体预测数量保持稳定；两者均未出现系统性偏差，预测结果均衡可靠。

（3）ROC 曲线对比分析

两者 ROC 曲线几乎重合，都紧贴左上角；AUC ≈ 0.9999+，说明模型对各类的二分类判别能力均接近完美；无论使用 CPU 还是 GPU，CNN 模型对数字识别的判别边界非常清晰。

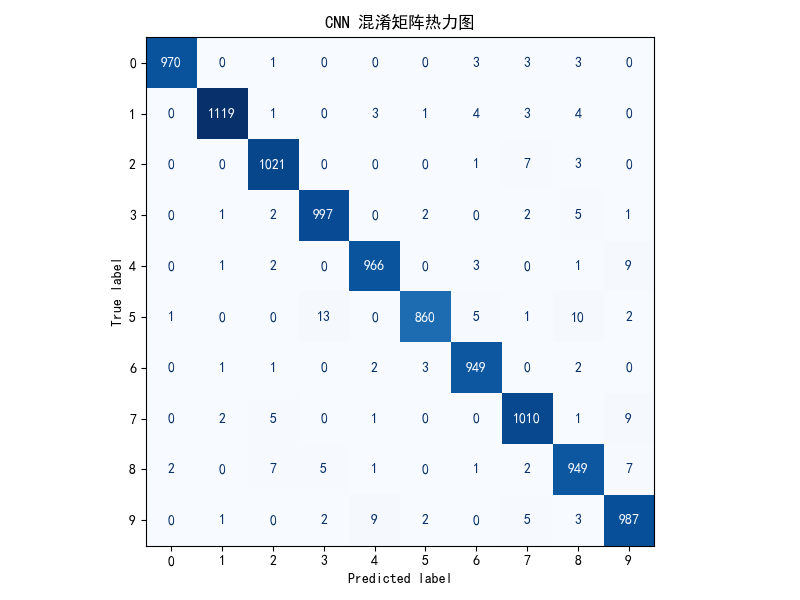
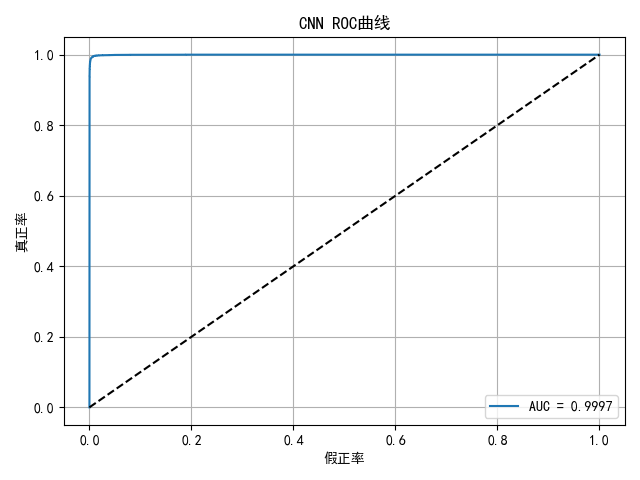
1. 总结

CNN 模型分别在 CPU 与 GPU 环境下训练 20 轮，分类性能表现极其接近，CPU 模型以 99.05% 的准确率略优于 GPU 模型（98.99%），但差距不足 0.1%，可视为统计波动；两者的 ROC 曲线均紧贴左上角，AUC 高达 0.9999，说明模型具有极强的判别能力。混淆矩阵显示两者均能稳定识别大多数数字类别。相比之下，GPU 模型在训练时长上具备明显优势（143 秒 vs 413 秒），在确保精度的同时显著提高了训练效率。

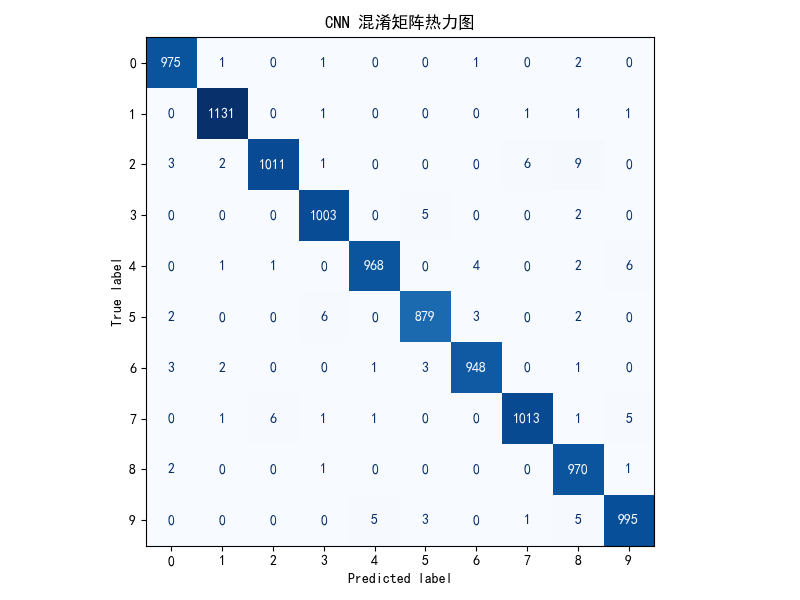
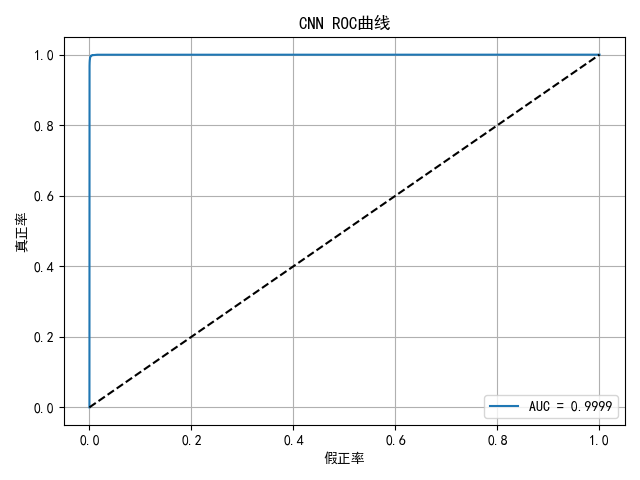
6.1.3 GPU加速下训练轮次20轮改变学习率结果对比

1.运行结果

（1）学习率=0.005



（2）学习率=0.0005



1. 对比

|  |  |  |  |
| --- | --- | --- | --- |
| 指标 | 学习率 0.001 | 学习率 0.005 | 学习率 0.0005 |
| 准确率 Accuracy | **0.9899** | 0.9828 | 0.9893 |
| 宏平均精度 Precision | 0.9898 | 0.9828 | 0.9892 |
| 宏平均召回 Recall | 0.9898 | 0.9826 | 0.9892 |
| 宏平均 F1 值 | **0.9898** | 0.9827 | 0.9892 |
| AUC 值（ROC） | 0.9999 | 0.9997 | 0.9999 |
| 主对角集中度 | 高 | 中 | **极高** |
| 错误类别分布 | 轻微集中 2/4 | 偏多集中 5/8/9 | 分布最均衡 |
| 收敛速度 | 快 | 中速 | 慢 |

（1）分类性能分析

学习率 0.001 与 0.0005 几乎并列第一，准确率均约为 98.9%+，宏平均 F1 值超过 0.989。0.005 表现略逊一筹，F1 值下降至 0.9827 左右，主要由于少数类（如 5、9）召回略低。从加权平均角度看，所有模型稳定性强，均未出现类间严重失衡现象。

（2）混淆矩阵结构对比

学习率 0.001：类别 2 和 4 有轻微混淆；类别 9 边界错误率较低；

学习率 0.005：类别 5、9 的误判数量明显增多；类别 8 易与 5 混淆，精度略低；

学习率 0.0005：预测最为稳定，误分类点极少，主对角线集中度最高；说明该学习率虽收敛慢，但结果最“干净”。

（3）ROC 曲线对比（AUC）

三者 AUC 均为 0.9997~0.9999，差距极小；学习率 0.0005 和 0.001 的 ROC 曲线更贴近左上角，说明整体分类边界最清晰；学习率 0.005 曲线稍有偏移，但整体仍为优级水平。

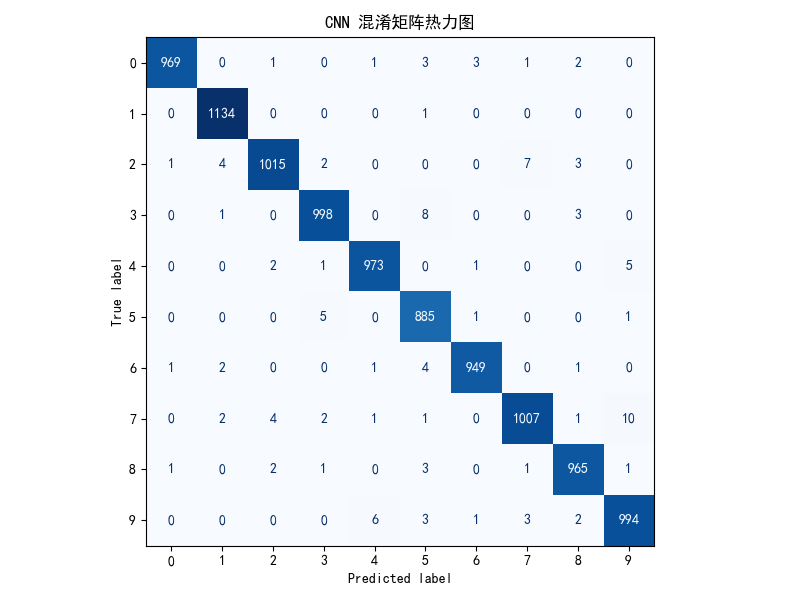
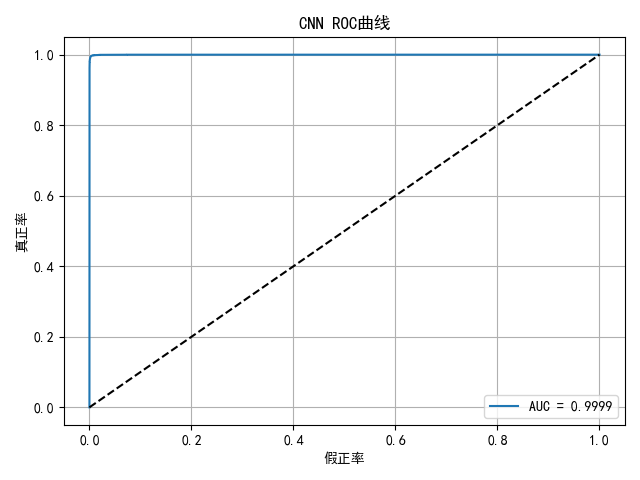
1. 总结

在 GPU 训练下，学习率为 0.001 和 0.0005 的 CNN 模型性能最佳，在准确率与 F1 值方面几乎持平，分类稳定性高，误分类率极低。若优先考虑 训练速度，推荐使用 0.01；若更追求 稳定收敛与极致精度，可选 0.0005。相比之下，0.005 表现略低，主要受部分类别精度波动影响。

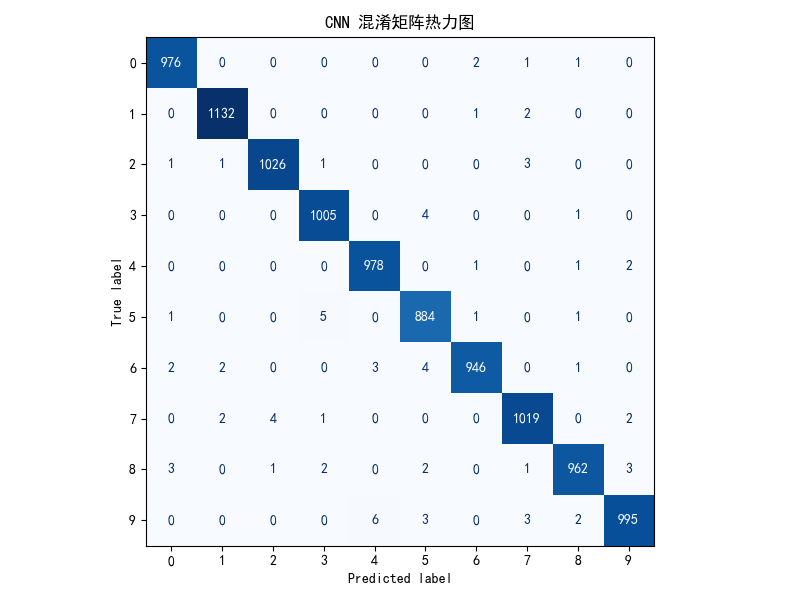
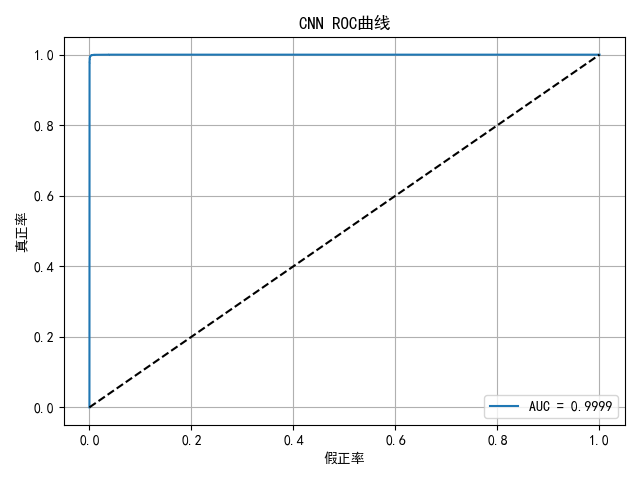
6.1.3 GPU加速下学习率0.001改变训练轮次结果对比

1.运行结果

（1）训练10轮



（2）训练30轮



1. 对比

|  |  |  |  |
| --- | --- | --- | --- |
| 指标 | 10轮 | 20轮 | 30轮 |
| 准确率 | 0.9889 | 0.9899 | **0.9923** |
| 宏平均精度 | 0.9888 | 0.9898 | **0.9922** |
| 宏平均召回 | 0.9889 | 0.9898 | **0.9922** |
| 宏平均 F1 值 | 0.9888 | 0.9898 | **0.9922** |
| AUC 值 | ≈0.9999 | ≈0.9999 | ≈0.9999 |
| 主对角集中度 | 稍少误判 | 更集中 | **极高** |
| 收敛特性 | 快速稳定 | 精度提升略微 | 明显提升，拟合充分 |

（1）性能指标对比

准确率提升趋势稳定：从 98.89%（10轮）到 99.23%（30轮），逐步优化；宏平均精度、召回率和 F1 值也稳步增长，说明模型对所有类别的判别能力逐轮加强；30轮时宏平均 F1 值提升近 0.3%，达到 0.9922，说明多训练轮次对性能有实际增益。

（2）混淆矩阵结构变化

10轮：大多数预测已聚焦于主对角线；少数类别（如数字 5、9）仍有一定程度混淆；

20轮：主对角线更清晰，误分类减少；类别 2、4、8 的准确性进一步改善；

30轮：混淆最少，分类界限最清晰；几乎所有类别主对角线命中率均超 98%，体现了充分训练的效果。

（3）ROC 曲线对比

所有轮次的 AUC 值均接近 1.0000，说明即使是10轮，模型也已经具备强大的整体判别能力；ROC 曲线差异极小，意味着轮次数对“总体判断边界”的提升影响不大，主要改进体现在微观误差的抑制上。

1. 总结

在固定学习率为 0.001 的条件下，随着训练轮数从 10 到 30 的增加，CNN 模型的整体分类性能持续提升。10 轮训练已可达到 98.89% 的高准确率，适合快速试验；20 轮为性价比最佳点；30 轮模型表现最强，误分类极少，推荐用于高精度任务。

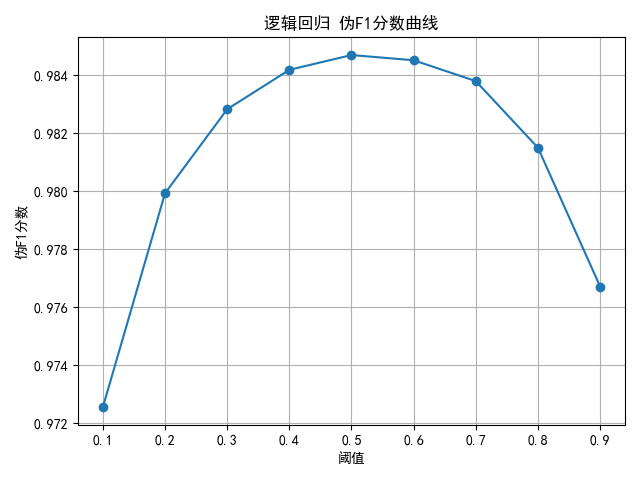
6.2 传统分类器结果

6.2.1 逻辑回归

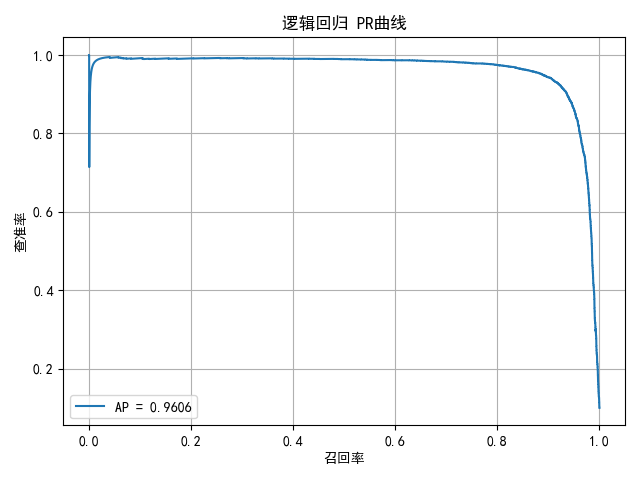
1.分类报告

|  |
| --- |
| 模型：逻辑回归  precision recall f1-score support  0 0.9480 0.9673 0.9576 980  1 0.9568 0.9762 0.9664 1135  2 0.9163 0.8905 0.9032 1032  3 0.9003 0.9119 0.9061 1010  4 0.9355 0.9308 0.9331 982  5 0.8907 0.8677 0.8790 892  6 0.9439 0.9489 0.9464 958  7 0.9318 0.9173 0.9245 1028  8 0.8699 0.8789 0.8744 974  9 0.9139 0.9158 0.9149 1009  accuracy 0.9216 10000  macro avg 0.9207 0.9205 0.9206 10000  weighted avg 0.9215 0.9216 0.9215 10000 |

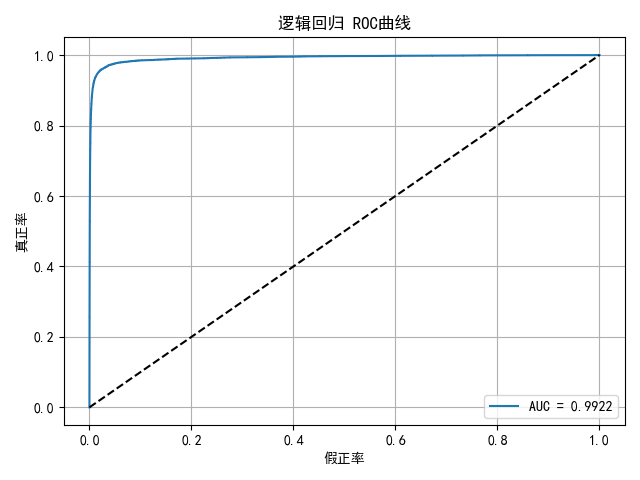
2.F1分数伪曲线



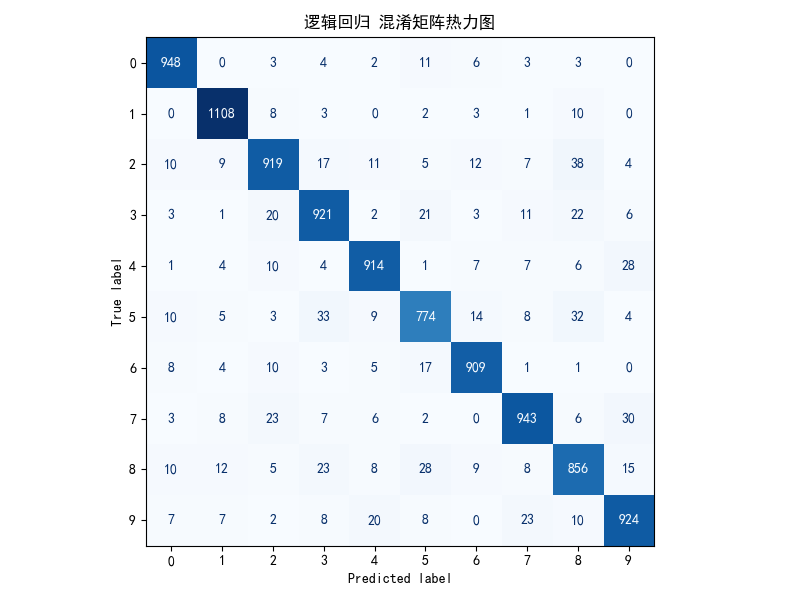
3.PR曲线



4.ROC曲线



5.混淆矩阵热力图



6.结果分析

（1）总体性能指标

从分类报告可以看出：准确率 92.16%，宏平均F1值0.9206，加权平均F1值0.9215。说明逻辑回归在 MNIST 上取得了超过 92% 的准确率，作为一个线性模型已具备较好的性能，尤其是在低资源或需要快速部署的场景中依然具有实用价值。

（2）混淆矩阵分析

从热力图中可以看出：类别 0、1、6、7 的识别效果最好（主对角线预测值高，误判少）；类别 5 与 3、8，类别 9 与 4、7 有一定程度的混淆；类别 2、3 的误差集中在相邻结构相似数字，可能由于其边界模糊。说明逻辑回归由于其线性特性，对于部分高度相似的数字分界不明显，存在一定的模糊判断。

（3）PR 曲线分析

PR 曲线整体偏上，面积 AP = 0.9606；模型在查准率与查全率之间取得了良好的平衡，尤其在高召回区间依然维持高精度。说明逻辑回归具有较强的实际可用性，在对误检要求较高的场景仍能保持较优性能。

（4）伪 F1 分数曲线分析

F1 值随概率阈值变化的曲线呈现出明显的抛物线趋势；在阈值 0.5 附近模型性能最优，F1 值达到 0.985+。

（5）ROC 曲线分析

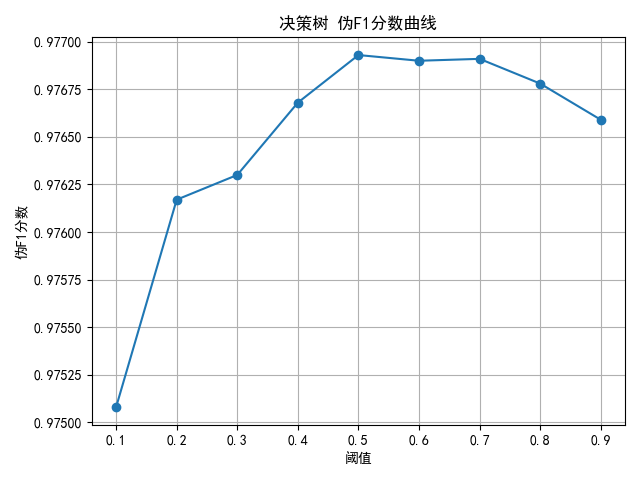
ROC 曲线紧贴左上角，AUC 达到 0.9922，非常接近 CNN 水平；表示模型在二分类（One-vs-Rest）判断中判别能力优秀。尽管是线性模型，逻辑回归在 MNIST 的“多类转二类”分类任务上具有出色的分界能力。

6.2.2 决策树

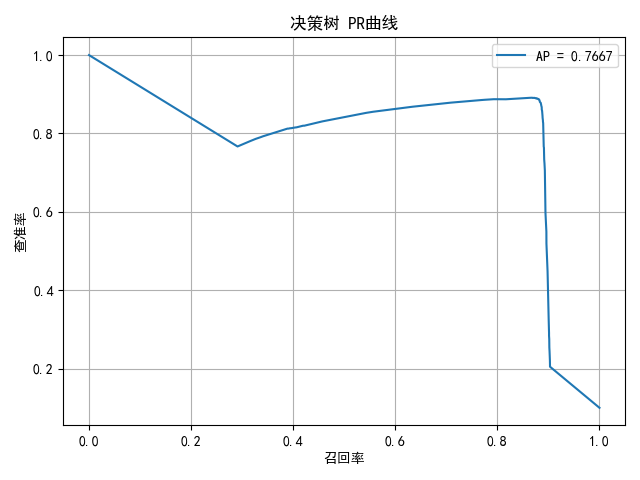
1.分类报告

|  |
| --- |
| 模型：决策树  precision recall f1-score support  0 0.9093 0.9408 0.9248 980  1 0.9532 0.9700 0.9616 1135  2 0.8781 0.8585 0.8682 1032  3 0.8372 0.8554 0.8462 1010  4 0.8850 0.8778 0.8814 982  5 0.8557 0.8509 0.8533 892  6 0.9010 0.8831 0.8919 958  7 0.9191 0.9056 0.9123 1028  8 0.8340 0.8101 0.8219 974  9 0.8448 0.8632 0.8539 1009  accuracy 0.8831 10000  macro avg 0.8817 0.8816 0.8815 10000  weighted avg 0.8830 0.8831 0.8829 10000 |

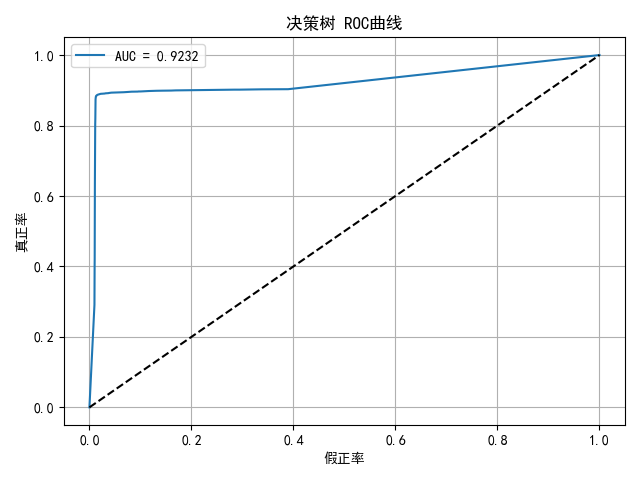
2.F1分数伪曲线



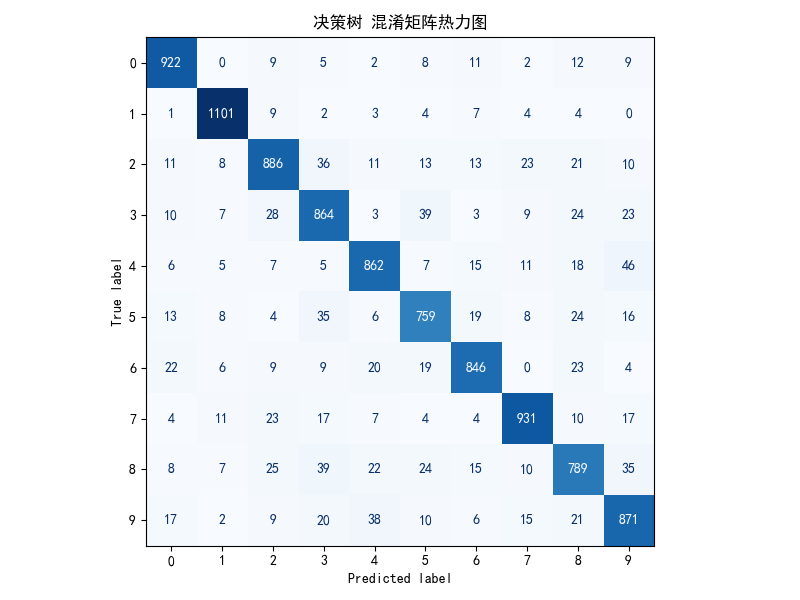
3.PR曲线



4.ROC曲线



5.混淆矩阵热力图



6.结果分析

1）总体性能指标

从分类报告可知：准确率88.31%，宏平均F1值0.8815，加权平均F1值0.8829。说明决策树作为一种结构清晰、可解释性强的分类模型，在本任务中实现了接近 88% 的准确率，效果中等偏上，但低于逻辑回归和 CNN。

（2）混淆矩阵分析

从决策树混淆矩阵热力图可以看出：类别 1、0、7 的识别准确率较高（对角线集中），模型对这些结构稳定的数字较敏感；类别 2、3、5、8、9 存在明显混淆，特别是：类别 2 容易被误分为 3、8；类别 3 被误判为 2、5、9；类别 9 被 4 和 7 混淆较多；整体误分类点比线性模型略多，显示决策边界较粗糙。

决策树易过拟合，且对连续特征的分割不如线性/深度模型平滑，因此存在较多局部误判。

（3）PR 曲线分析

平均精度 AP = 0.7667，PR 曲线整体浮动较大，在中低召回率段出现明显下滑。说明该模型在查准查全的稳定性方面存在波动，对罕见或边界样本的预测能力不够强。

（4）伪 F1 分数曲线分析

F1 曲线在阈值 0.5~0.7 区间接近平稳，最大值约为 0.977；阈值在中间区域设置效果最佳，调整空间有限。说明默认阈值基本合适，但 F1 变化不显著说明模型分类信心分布较集中，难以通过调整阈值获得质的提升。

（5）ROC 曲线分析

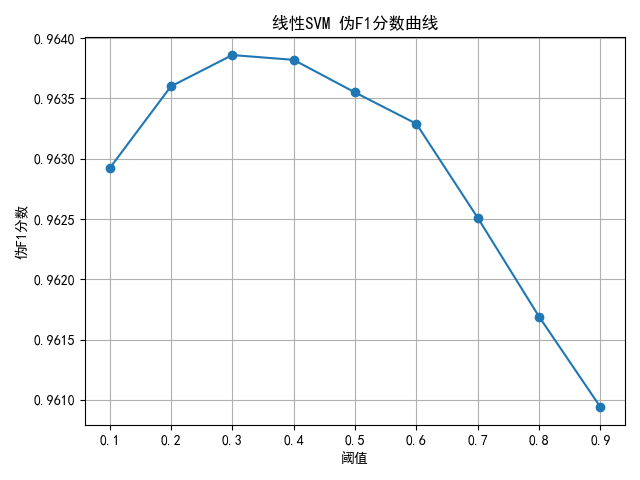
AUC = 0.9232，表现中等；曲线前段较陡，说明模型在某些类别上区分能力较强，但整体边界不如逻辑回归或 CNN 清晰。模型对部分类别（如 0, 1）表现良好，对部分边缘类别存在判别能力不足的问题。

6.2.3 线性SVM

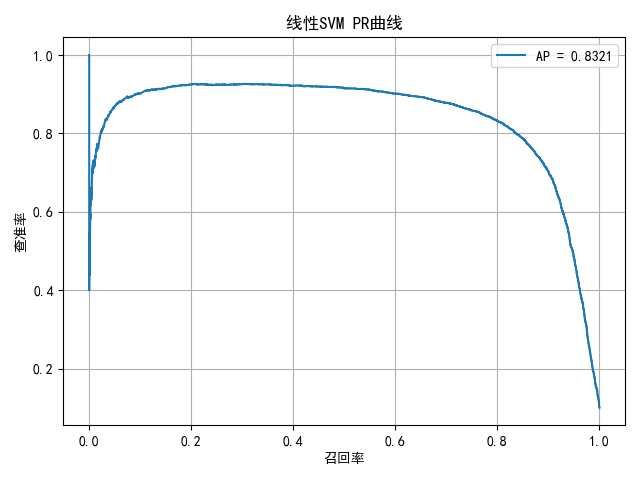
1.分类报告

|  |
| --- |
| 模型：线性SVM  precision recall f1-score support  0 0.9066 0.9602 0.9326 980  1 0.9189 0.9683 0.9429 1135  2 0.8906 0.7965 0.8409 1032  3 0.8391 0.8465 0.8428 1010  4 0.8679 0.8829 0.8753 982  5 0.8272 0.8049 0.8159 892  6 0.8920 0.8883 0.8902 958  7 0.8525 0.8716 0.8620 1028  8 0.8110 0.8060 0.8084 974  9 0.8506 0.8295 0.8399 1009  accuracy 0.8671 10000  macro avg 0.8656 0.8655 0.8651 10000  weighted avg 0.8667 0.8671 0.8664 10000 |

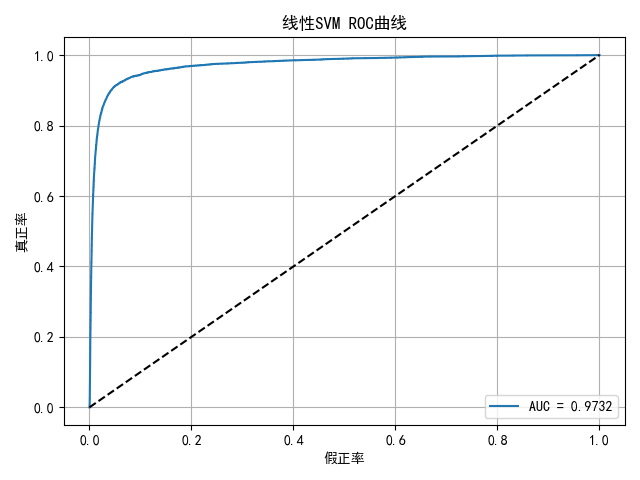
2.F1分数伪曲线



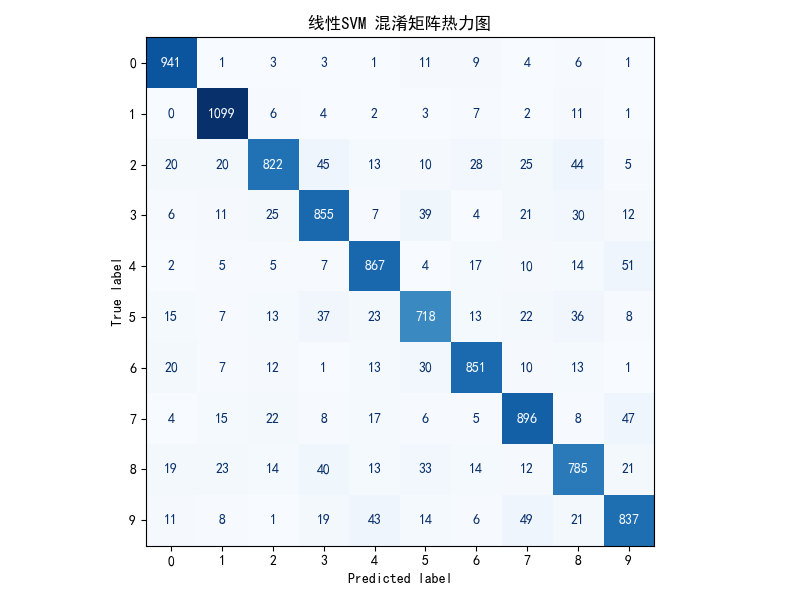
3.PR曲线



4.ROC曲线



5.混淆矩阵热力图



6.结果分析

1）总体性能指标

根据分类报告可得：准确率86.71%，宏平均F1值0.8651，加权平均F1值0.8664。模型整体准确率达到 86.7%，在传统模型中属于中等水平；宏平均与加权 F1 值差异极小，说明预测在各类别间较为均衡。

（2）混淆矩阵分析

从热力图中可以观察到：类别 0 和 1 的分类效果最好（误差极少），；类别 2、3、5、8 存在明显混淆，例如：类别 2 被误分为 3、8；类别 5 与 3、6、8 有交叉；类别 9 与 4、8 易发生混淆；整体看，结构复杂或轮廓相似的数字较易出错。说明Linear SVM 由于采用线性核，决策边界简单，难以精准拟合 MNIST 中非线性较强的图像分布；若使用非线性核函数（如 RBF）可进一步提升性能，但计算成本更高。

（3）PR 曲线分析

平均精度（AP）达到 0.8321，处于中等偏上的水平；曲线整体平稳，但两端略有下跌，说明在极低或极高召回率条件下查准率略有下降。说明模型在中间阈值区域表现较好，对大部分样本具备稳定识别能力；对边界样本判断仍有待提升。

（4）伪 F1 分数曲线分析

伪 F1 值在阈值 0.3 附近达到峰值约 0.964；阈值变动对 F1 值影响有限，说明模型输出决策函数分布相对集聚，缺乏分层结构。当前默认阈值已较优，调整决策边界阈值对总体效果提升作用有限。

（5）ROC 曲线分析

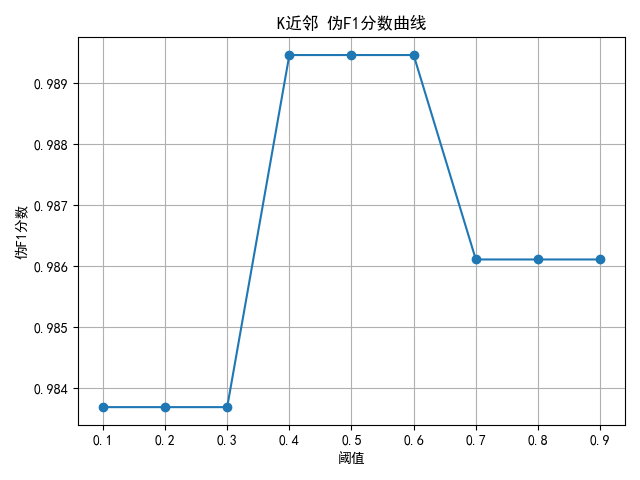
AUC 值为 0.9732，说明模型整体具有较强的判别能力；曲线斜率大、靠近左上角，尤其在误判率较低时依然保持较高的召回率。虽然线性 SVM 属于线性模型，但在高维特征空间下依然具备优秀的分类能力；若使用更复杂核函数（如 RBF）有望进一步提升 AUC 值。

6.2.4 K近邻

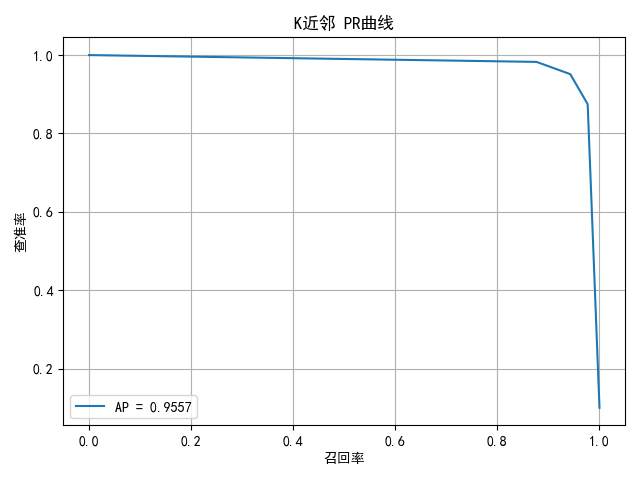
1.分类报告

|  |
| --- |
| 模型：K近邻  precision recall f1-score support  0 0.9480 0.9867 0.9670 980  1 0.9519 0.9930 0.9720 1135  2 0.9518 0.9370 0.9443 1032  3 0.9233 0.9535 0.9381 1010  4 0.9525 0.9389 0.9456 982  5 0.9324 0.9283 0.9303 892  6 0.9666 0.9656 0.9661 958  7 0.9371 0.9270 0.9320 1028  8 0.9626 0.8973 0.9288 974  9 0.9269 0.9177 0.9223 1009  accuracy 0.9452 10000  macro avg 0.9453 0.9445 0.9447 10000  weighted avg 0.9453 0.9452 0.9450 10000 |

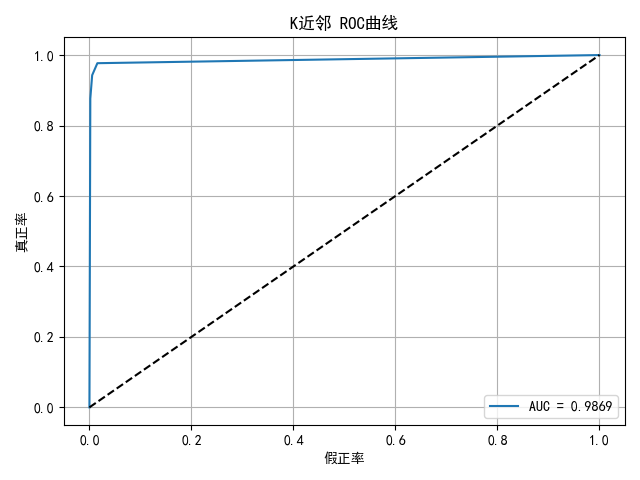
2.F1分数伪曲线



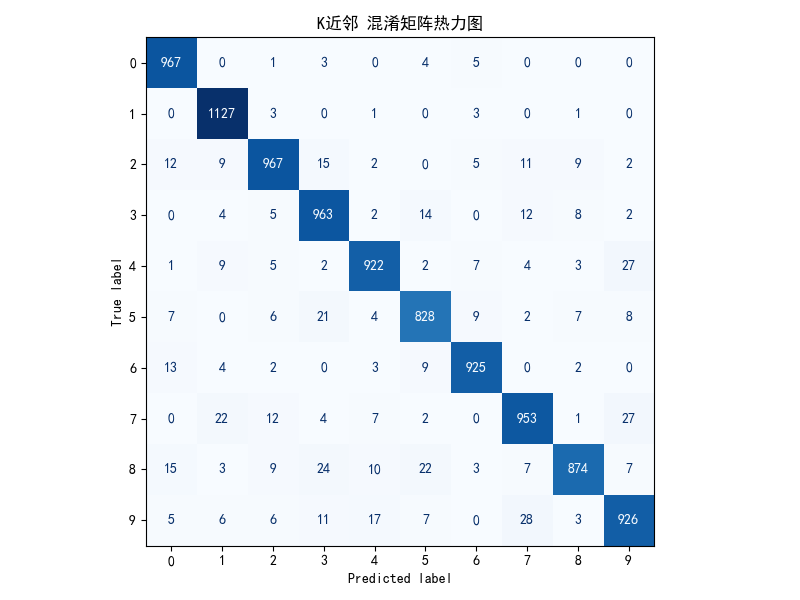
3.PR曲线



4.ROC曲线



5.混淆矩阵热力图



6.结果分析

（1）总体性能指标

从分类报告可得：准确率94.52%，宏平均F1值0.9447，加权平均F1值0.9450。准确率接近 95%，在传统模型中表现极为优秀；各类精确率与召回率高度一致，说明模型对各类别分类能力稳定；宏/加权 F1 值相近，表明类间样本不均并未影响模型判断质量。

（2）混淆矩阵分析

从热力图中观察：模型对数字 0、1、2、3、6 的预测非常精准；少量误分类主要集中于：类别 8 与 3、5、9 混淆；类别 7 与 1、9 混淆；类别 5 和 3 存在边界重叠；总体错误率低，且误差分布较均匀，无极端偏误类别。说明KNN 对空间特征保留较敏感，适合像素分布规律明显的数据；容易被“笔画相似”的数字混淆，且对噪声与变形样本相对敏感。

（3）PR 曲线分析

平均精度（AP）达到 0.9557，非常接近逻辑回归；曲线整体平稳，召回率在 0.9 以内时仍能维持高查准率。说明模型在预测强度和容错能力上表现良好，适用于对查全率有要求的场景。

（4）伪 F1 分数曲线分析

F1 值在阈值 0.4～0.6 区间达到峰值（最高约 0.989）；多个阈值下表现一致，说明模型对类别划分信心一致、分类稳定性强。说明模型具有良好的输出分布均衡性，适合直接使用默认阈值。

（5）ROC 曲线分析

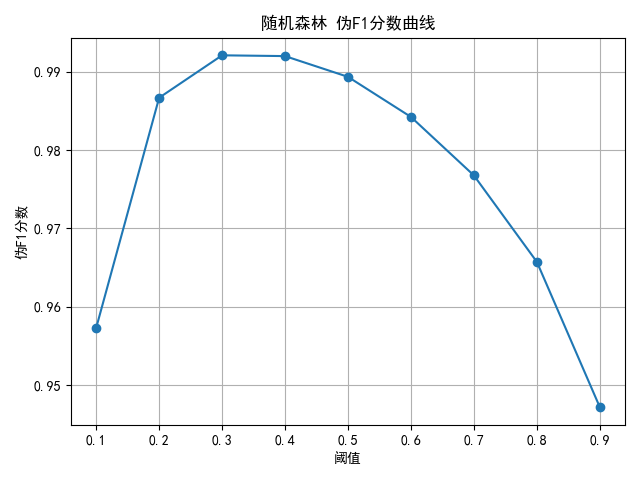
AUC = 0.9869，仅次于 CNN 和逻辑回归；ROC 曲线紧贴左上角，整体判别能力非常强。说明 KNN 在当前数据集上拟合能力优越，尤其在低误报区间内识别准确性高。

6.2.5 随机森林

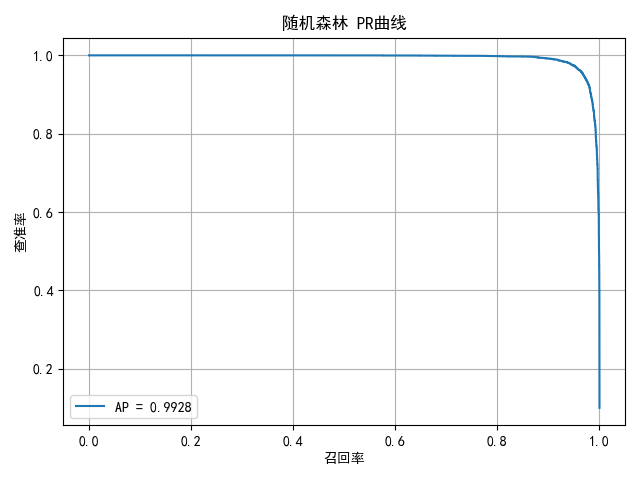
1.分类报告

|  |
| --- |
| 模型：随机森林  precision recall f1-score support  0 0.9729 0.9888 0.9808 980  1 0.9868 0.9885 0.9877 1135  2 0.9521 0.9632 0.9576 1032  3 0.9556 0.9584 0.9570 1010  4 0.9722 0.9623 0.9672 982  5 0.9738 0.9585 0.9661 892  6 0.9709 0.9749 0.9729 958  7 0.9674 0.9533 0.9603 1028  8 0.9576 0.9497 0.9536 974  9 0.9401 0.9495 0.9448 1009  accuracy 0.9650 10000  macro avg 0.9649 0.9647 0.9648 10000  weighted avg 0.9650 0.9650 0.9650 10000 |

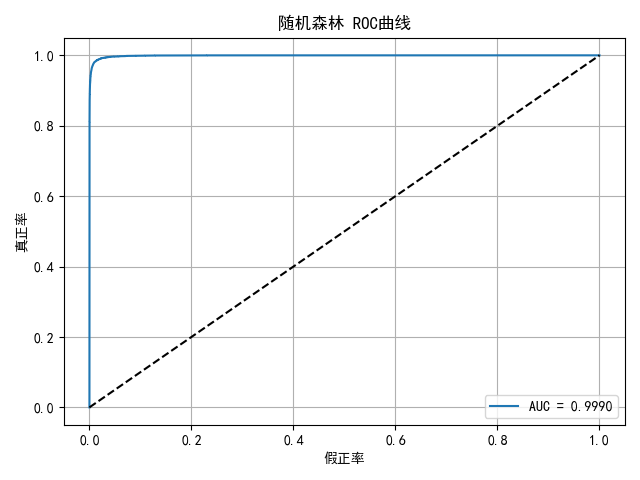
2.F1分数伪曲线



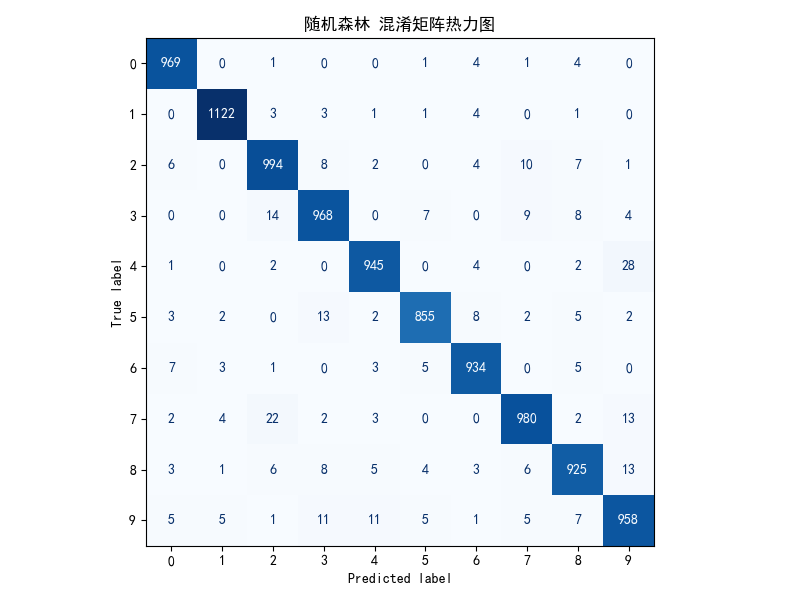
3.PR曲线



4.ROC曲线



5.混淆矩阵热力图



6.结果分析

（1）总体性能指标

从分类报告可得：准确率96.50%，宏平均F1值0.9648，加权平均F1值0.9650。随机森林的准确率在所有传统模型中最高，接近 CNN 级别；各类别预测性能较为均衡，宏/加权 F1 值几乎一致，说明模型兼顾了样本数量不均问题；泛化能力强，对大部分数字均能有效识别。

（2）混淆矩阵分析

从热力图可见：类别 0、1、2、3、6 的识别精度极高；少量错误集中在：类别 9 被混淆为 4 和 8；类别 5 与 3、8 有边界重叠；类别 7 与 1、9 仍有小部分误判；整体误差控制较好，模型具备很强的区分度。得益于集成策略，模型能有效降低单一决策树的过拟合问题，并显著提升分类精度。

（3）PR 曲线分析

平均精度（AP）达到 0.9928，为所有传统模型中最优；整体 PR 曲线平稳并靠近右上角，说明模型在高查全率下仍具高查准率。说明随机森林在处理不平衡数据时表现稳定，是高精度分类场景的优选模型。

（4）伪 F1 分数曲线分析

F1 值峰值出现在阈值 0.3～0.4 附近（约为 0.991）；曲线变化趋势平滑，对阈值不敏感，说明模型输出信心分布较一致。当前使用的默认分类阈值设置合理，不需要进行复杂调参。

（5）ROC 曲线分析

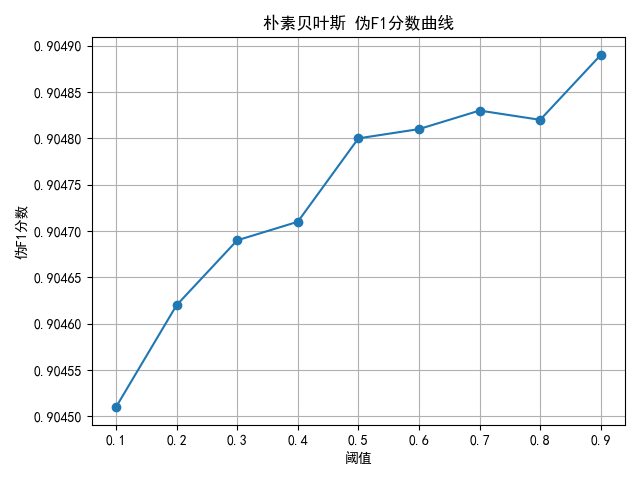
AUC 高达 0.9990，紧贴左上角，对正负样本区分能力极强；在所有传统模型中，AUC 接近 CNN，为模型综合性能的体现。该模型在“二类对多类”转换中仍然能保持极强判别能力，适用于需要高灵敏度的场景。

6.2.6 朴素贝叶斯

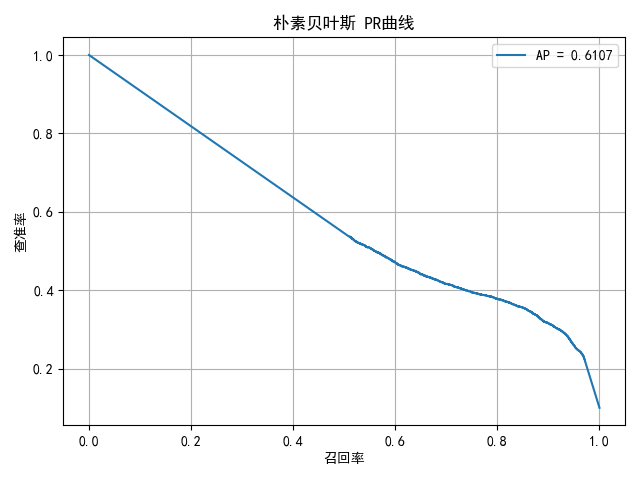
1.分类报告

|  |
| --- |
| 模型：朴素贝叶斯  precision recall f1-score support  0 0.7596 0.8449 0.8000 980  1 0.8862 0.9401 0.9124 1135  2 0.9010 0.1764 0.2950 1032  3 0.6498 0.2792 0.3906 1010  4 0.8600 0.1314 0.2279 982  5 0.4531 0.0325 0.0607 892  6 0.6933 0.9154 0.7890 958  7 0.8816 0.1955 0.3201 1028  8 0.2522 0.7023 0.3711 974  9 0.3625 0.9524 0.5251 1009  accuracy 0.5240 10000  macro avg 0.6699 0.5170 0.4692 10000  weighted avg 0.6767 0.5240 0.4773 10000 |

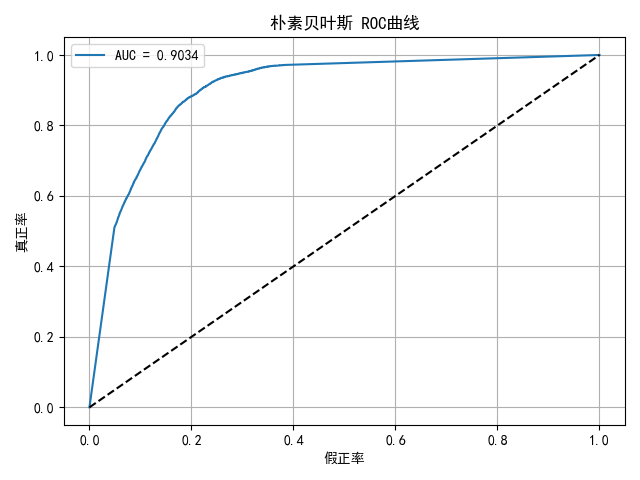
2.F1分数伪曲线



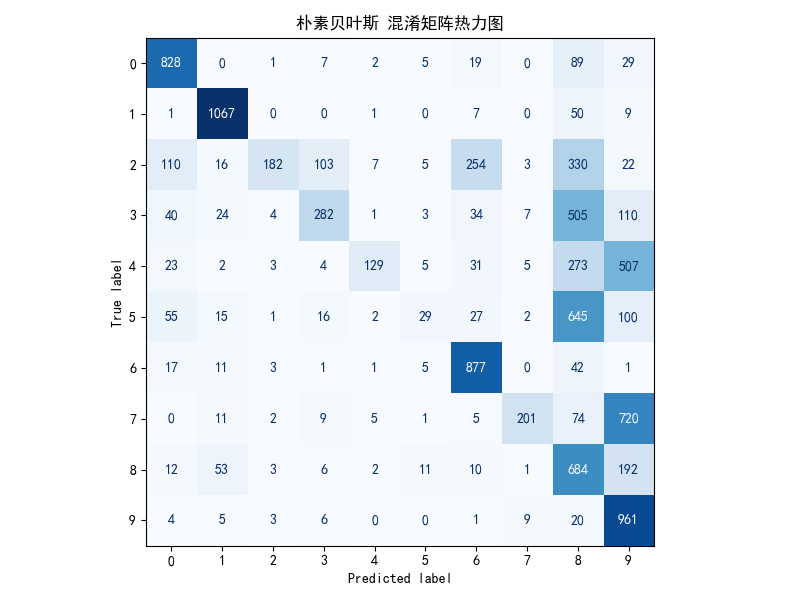
3.PR曲线



4.ROC曲线



5.混淆矩阵热力图



6.结果分析

（1）总体性能指标

根据分类报告可得：准确率52.40%，宏平均F1值0.4692，加权平均F1值0.4773。模型准确率远低于其他传统模型，仅略优于随机预测；宏平均和加权平均的 F1 值相对较低，表明模型在大部分类别上预测效果较差；主要原因是特征之间的强相关性违背了朴素贝叶斯的独立性假设。

（2）混淆矩阵分析

从热力图可见：类别 1 和 6 的识别表现尚可（召回率高达 91%+）；类别 5、3、4、7 被大量误分类，特别是：类别 5 被严重混淆为 8 和 9；类别 8 与多数类别（尤其是 2、3、5、7）存在交叉误判；类别 9 几乎全被正确识别（召回率 95%），但 precision 很低（误判为 9 的样本较多）；说明模型对某些“概率分布尖锐”的类别（如 1、6、9）表现尚可；对于结构复杂或分布重叠的数字，极易误判，尤其在高维图像特征上模型泛化能力差。

（3）PR 曲线分析

平均精度（AP）仅为 0.6107，说明查准能力弱；曲线从左上快速下降，召回率提高时 precision 急剧下滑。模型对“容易分类”的样本（如类别 1）预测较稳定；对模糊边界样本无显著区分能力，查准率下降快。

（4）伪 F1 分数曲线分析

整体 F1 值约为 0.9045~0.9049（仅从伪分数角度）；但曲线基本平直，阈值变化对模型提升无显著影响。说明输出概率分布相对“平均化”，模型分类信心不足。

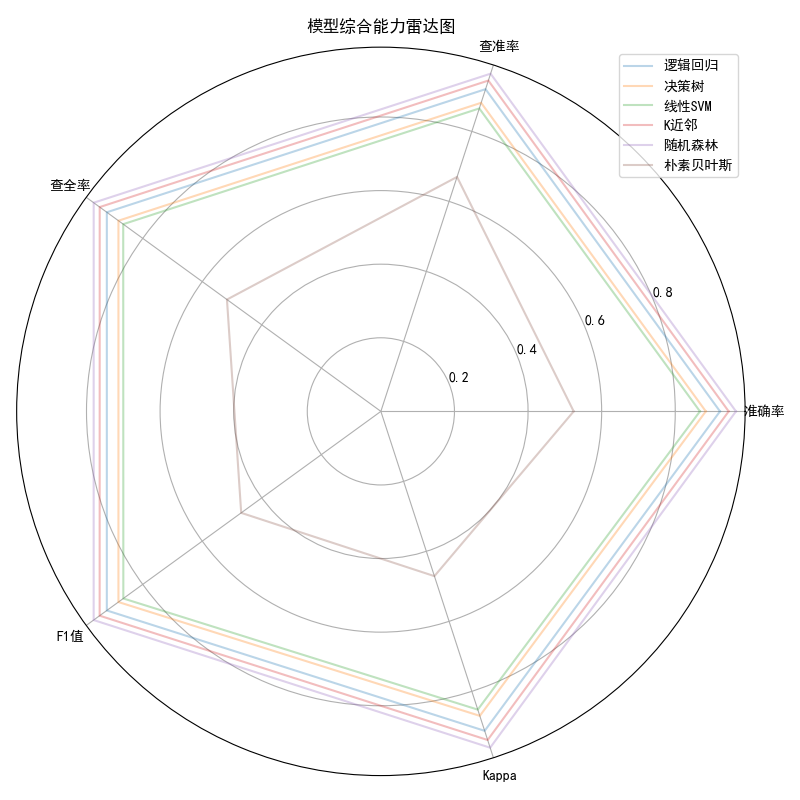
（5）ROC 曲线分析

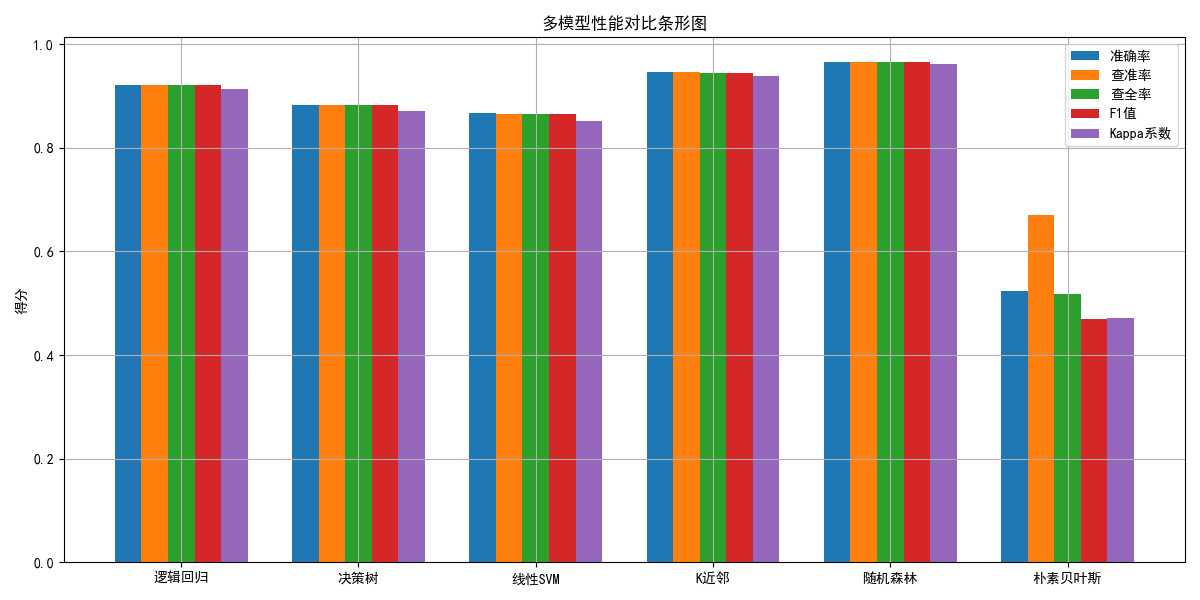
AUC = 0.9034，为传统模型中最低之一；曲线整体弯曲幅度较小，说明模型判别边界模糊。在某些二分类任务中尚可使用，但不适合多类高维图像任务。

6.2.7 传统分类器对比

实验对比了六种经典的机器学习分类器（逻辑回归、决策树、线性支持向量机、K近邻、随机森林、朴素贝叶斯）在 MNIST 手写数字识别任务中的表现，从准确率、查准率、查全率、F1 值、Kappa 系数、训练耗时等维度进行综合评估。

1. 分类性能对比



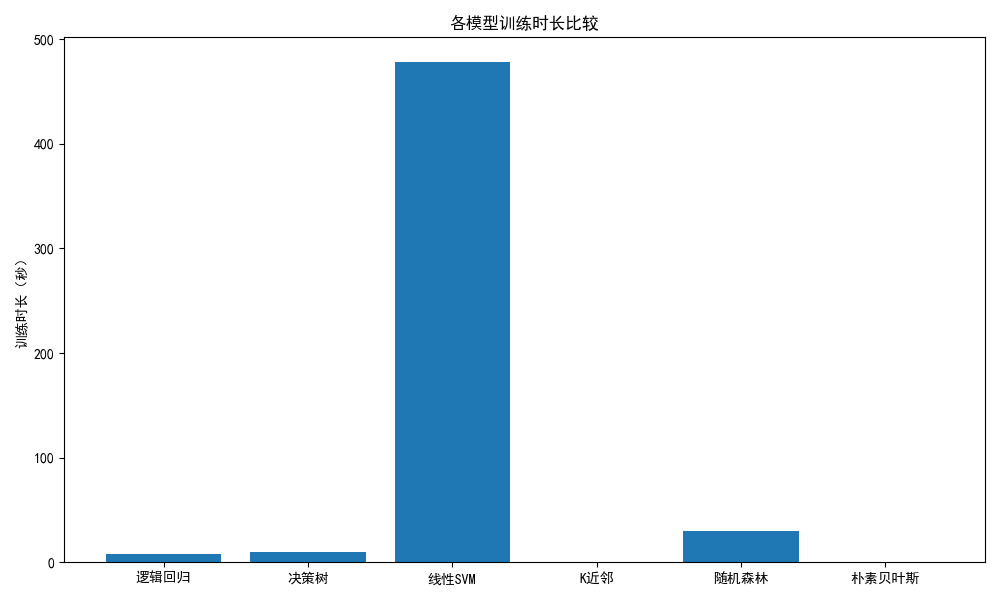


根据模型雷达图与性能柱状图可观察：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 模型 | 准确率 | 精确率 | 召回率 | F1 值 | Kappa |
| 随机森林 | 96.50% | 0.9650 | 0.9650 | 0.9650 | 0.9602 |
| K近邻 | 94.52% | 0.9453 | 0.9445 | 0.9447 | 0.9393 |
| 逻辑回归 | 92.16% | 0.9215 | 0.9216 | 0.9215 | 0.9123 |
| 决策树 | 88.31% | 0.8830 | 0.8831 | 0.8829 | 0.8735 |
| 线性SVM | 86.71% | 0.8667 | 0.8671 | 0.8664 | 0.8579 |
| 朴素贝叶斯 | 52.40% | 0.6767 | 0.5240 | 0.4773 | 0.4182 |

说明随机森林表现最优，几乎在所有指标上领先；K近邻和逻辑回归也保持良好的分类稳定性；决策树、线性SVM处于中等水平；朴素贝叶斯显著低于其他模型，尤其在结构复杂的数据集上存在严重偏差。

1. 训练效率对比



从训练时长条形图可见

|  |  |
| --- | --- |
| 模型 | 训练时长（秒） |
| 线性SVM | 477.81 |
| 随机森林 | 29.74 |
| 决策树 | 10.04 |
| 逻辑回归 | 7.83 |
| 朴素贝叶斯 | 0.48 |
| K近邻 | 0.11 |

K近邻和朴素贝叶斯训练速度极快，适合快速部署场景；逻辑回归与决策树训练耗时较低但性能可靠；随机森林性能最佳但训练时长相对较高；线性SVM耗时远高于其他模型，不适合在全量数据上直接使用。

1. 结论

综合性能最佳模型：随机森林。在精度与泛化能力方面最接近深度学习模型 CNN；

效率最佳模型：K近邻。训练时间极短但精度不低，适合小型任务；

平衡性最佳模型：逻辑回归。训练迅速、指标均衡，适合基础分类任务；

不推荐模型：朴素贝叶斯。在 MNIST 图像类数据上表现较差，不适用于此类结构复杂、高维特征任务

## 七、结论与展望

课程设计围绕手写数字识别任务，系统比较了多种传统机器学习分类器与卷积神经网络（CNN）在 MNIST 数据集上的性能差异，采用多轮实验，分别从精度、训练时间、鲁棒性等角度开展了深入分析，通过本次课程设计，我系统地掌握了机器学习与深度学习分类模型的构建、训练与评估流程，深入理解了传统分类器（如逻辑回归、决策树、SVM、KNN 等）与卷积神经网络（CNN）在手写数字识别任务中的性能差异与适用场景。项目过程中，我熟练掌握了 PyTorch 框架的基本用法，包括模型定义、数据加载、GPU 加速、性能指标计算与结果可视化。同时，我也学会了如何设计实验变量（如学习率、训练轮数），以及通过混淆矩阵、ROC 曲线、F1 分数等多维指标进行模型效果分析与对比。这些实践提升了我对模型调优与结果解释的能力，加深了对人工智能模型在图像分类等实际任务中应用的理解。

## 八、附录

参考文献

[1] 周志华. 机器学习[M]. 清华大学出版社, 2016.

[2] Ian Goodfellow, Yoshua Bengio, Aaron Courville. Deep Learning[M]. MIT Press, 2016.

[3] 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.

[4] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[C]//NeurIPS. 2012.

1. [↑](#footnote-ref-0)