# 《机器学习与大数据》课程大作业：M蛋白检测智能算法

## 1 预处理及图片识别

### 1.1数据预处理

将所有图片读入，因为颜色单一，所以将RGB数值合并为灰度数值(数值越小，代表颜色越深)。我们提供了两种将RGB图片转为灰度图片的方法，其一是利用cv.imread函数在读取图片时将其自动转为灰度图；其二是在查看了cv库源代码后发现，灰度和RGB之间存在以下关系：

$$Grey[i,j] = 0.299R[i,j] + 0.587[i,j] + 0.114[i,j]$$

由此我们实现了不调用任何库函数就将彩色图片转为了灰度图片。

具体步骤：

1） 去掉每张图片像素点的开头几行，那些事用于表示“K”、“G”、“A”等字母的像素值。

2） 观察到每张图片的M蛋白相关只出现在前125行之中，剩余的几行像素点只会出现白蛋白，所以裁剪掉125行之后的所有像素点。

3） 由于六列条带在条带内部十分对称，且彼此之间的白色间隔并没有提供信息。所以我们提取出六条条带，并且在每一行取均值代表这一条在该位置的灰度值。

4） 将样本中医生诊断不清晰的几组数据去除掉，不列入训练集或测试集之后。

5） 经过以上处理，将每个图片转化为了125乘6的灰度值矩阵，我们将用此进行分类学习

在神经网络算法中，我们需要将输入图片的尺寸归一化，我们提出了三种方法。其一是使用我自己写的函数进行暴力截断补零，由于该方法未进行任何优化，实验效果较差；其二是使用OpenCV的resize函数进行三次样条插值；其三是利用torchvision库中的transforms.Resize函数进行尺寸变换，该方法最为灵活，最终选用此方法。

### 1.2 M蛋白的位置识别

人工识别M蛋白位置，即对每一个阳性样本，在其“SP”条带上，选取灰度值最小（上文数据预处理中提到数值越小，越色越深）的一行作为M蛋白的位置，将对应的矩阵行数对应回图片的像素点，即可得到M蛋白在图片中的位置。

## 2 辅助诊断决策模型

### 2.1 问题描述

我们认为这是个9分类的问题，即一种阴性和八个阳性，并且由于阳性样本过少，我将“可疑”、“强阳性”、“阳性”、“弱阳性”等属性均划分为阳性，一定程度上减少了样本的不均衡性。并且用0-8对这九个类别进行编号。0代表阴性，1代表lgG-K阳性，2代表lgG-λ阳性，3代表lgA-K阳性，4代表lgA-λ阳性，5代表lgM-K阳性，6代表lgM-λ阳性，7代表K阳性，8代表λ阳性。在本部分，我们使用分类学习方法进行处理。

### 2.2 训练集、测试集的划分与PCA降维处理

直接从图片转化为灰度数据的样本数据维度过高，会导致后续训练过程中速度过慢，模型受噪音影响过大等缺陷。因此，我们对样本数据进行PCA降维处理。

将阳性样本中抽出20%加入测试集，在阴性中抽出同等数量的样本加入测试集，即测试集中阴性样本阳性样本比例为1：1，剩下的作为训练集。用训练集训练PCA降维模型，用该模型将训练集与测试集的数据维度降低到50维。

## 2.3 处理不均衡的数据

通过观察样本特征与初步的测试，我们发现训练集中阴性样本和阳性样本存在明显的不均衡现象，导致阴性和阳性的召回率有较大差距（分类效果见附录 1）。因此，我们尝试了一些方法处理不均衡的数据。

### 2.3.1 加权

我们尝试在 Loss Function 的计算中，依据各类别样本比例，赋予不同类别不同的权重。假设第 i 类样本的数量为$x_i$，则第 i 类样本的权重公式为

$$w_i = \frac{-x_i + \sum_{j=1}^{9} x_j}{\sum_{j=1}^{9} x_j}$$

应用这种方式后，我们发现，阳性召回率大幅度提高，但同时，阴性召回率有所下降，整体分类准确率只有略微的提高（分类效果见附录 2）。使用加权的方式处理不均衡的数据，分类效果的提升是有限的，因此，我们继续尝试使用重采样的方式处理不均衡的数据。

### 2.3.2 欠采样

#### 2.3.2.1 朴素随机欠采样

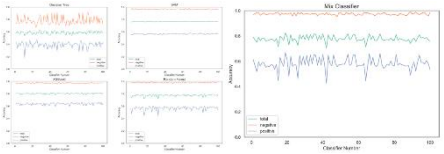
从阴性训练集中抽取（无放回）与阳性训练集相等数量的阴性样本作为阴性训练集，并训练分类模型。

应用这种方式后，我们发现，阳性召回率大幅度提高，且整体分类准确率也有比加权方法更大幅度的提高（分类效果见附录 3）。朴素随机欠采样的方式在该问题中比加权方式有更好的表现效果。但朴素随机欠采样的方式会导致大量阴性数据的浪费，因此，我们进一步尝试了 EasyEnsemble 的方法提高样本数据的利用率。

#### 2.3.2.2 EasyEnsemble

这一方法通过多次对样本进行朴素随机欠采样，训练多个分类器，最终按照各个分类器交叉验证集准确率的均值水平，将多个分类器对测试集的分类结果加权为一个分类器。

在这一部分，我们分别训练 100 个 Decision Tree，100 个 SVM，100 个 XGBoost，100 个 Random Forest，100 个 Decision Tree/SVM/XGBoos/Random Fores 随机混合，共 5 种加权分类器，并记录每种分类器内部分类器数量从 1 增加至 100 分类效果的变化。



（不同算法对应的加权分类器随着内部分类器数量的增多分类准确率的变化）

结果显示（大图见附录 4）随着 classifier 数量的增多，分类器的效果并无明显变化。尽管 EasyEnsemble 利用了更多的阴性样本数据，但它并不能使分类效果有较大的提高。推测可能是因为阴性样本中本身只有部分可以提供不重复的对分类有意义的信息。

### 2.3.3 过采样

欠采样中，被剔除的样本中可能包含着一些重要信息，无法充分利用已有的信息，所以我们进一步尝试过采样的处理方法。

#### 2.3.3.1 朴素随机过采样

在朴素随机过采样中，只是单纯地重复阳性样本，因此会过分强调训练集中的阳性样本，噪音和错误也容易被成倍地放大，出现对正例的过拟合。所以我们使用 SMOTE 方法。

#### 2.3.3.2 SMOTE

SMOTE 不再是单纯地重复阳性样本，而是在局部区域通过 K-means 生成了新的正例，相较于朴素过采样，优点是降低了过拟合的风险，对噪音的抵抗性更强，缺点是运算时间变长，同时也会生成一些“可疑的点”。

由于 SMOTE 对每个少数类样本均合成了相同数量的样本，没有利用少数样本数量的信息，所以尝试使用 ADASYN 处理。

#### 2.3.3.3 ADASYN

在 ADASYN 中，自动决定为每个少数类样本生成多少合成样本，相当于给少数类样本施加了一个权重，周围的多数类样本越多则权重较高，但是容易受 outlier 的影响，通过对比数据可发现，ADASYN 和 SMOTE 针对本数据集效果相差不大。

### 2.3.4 combination

由于普通 SMOTE 方法生成的少数类样本是通过线性差值得到的，在平衡类别分布的同时也扩张了少数类的样本空间，产生的问题是可能原本属于多数类样本的空间被少数类“入侵”，容易造成模型的过拟合。ENN/Tomek links 可以很好地解决“入侵”的问题。

我们使用了 SMOTE+ENN 和 SMOTE+Tomek links 两种方法进行处理，与前述方法对比，一方面增加了少数样本的数量，另一方面减少了不同类别样本之间的重叠，在项目中有较好的表现。

### 2.3.5 效果比对

以 SVM 算法为例，比对不同处理不均衡数据的方法的分类效果，结果如下表。

| 方法 | 不处理 | 加权 | 朴素随机欠采样 | EasyEnsemble |
|---|---|---|---|---|
| 正确率（整体/阳性） | 68.49%/38.54% | 73.57%/50% | 75.78%/55.73% | 77.47%/58.59% |
| 方法 | 朴素随机过采样 | SMOTE | ADASYN | SMOTE+Tomek |
| 正确率（整体/阳性） | 75.74%/76.30% | 75.13%/75.00% | 74.48%/75.00% | 74.35%/74.48% |

## 2.4 分类算法

在之前探究使用不同方法处理不均衡数据的过程中，我们发现，SMOTEENN 上采样方法对于我们的数据效果最好，因此，在后续探究不同分类算法的分类效果时，我们首先使用 SMOTEENN 上采样方法对不均衡的数据进行处理。

我们尝试了决策树、随机森林、SVM、XGBoost 四种分类算法对数据分类。效果如下：

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 45.44%/78.39% | 78.39%/66.15% | 70.57%/82.03% | 81.38%/75.78% |

结果显示，从总体分类效果上来看，XGBoost 算法分类效果最好，Random Forest 次之，但对于阳性样本的分类，SVM 分类更为准确。

XGBoost 和 Random Forest 都是基于 Decision Tree 衍生的模型，其中 XGBoost 是由 GDBT 算法改进，通过在代价函数里加入正则项，控制模型复杂度， 从而减少过拟合，而 GDBT 算法是决策树模型与 Boosting 算法的结合；而 Random Forest 模型是决策树模型与 Bagging 算法的结合。从算法机制上来说，XGBoost 通过减少模型偏差提高性能，其精度会更高，但同时拥有更高的过拟合风险；而 Random Forest 通过减少模型方差提高性能，因而拥有更强的泛化能力。我们的结果中，XGBoost 拥有最高的整体准确率，正是其高精度的体现，但可以看出，它在训练集上的精度为 1，说明模型存在过拟合的现象，且其阳性召回率远低于阴性召回率，说明其泛化能力较低。

SVM 算法的阳性召回率较高，这可能说明，对于阳性中内部的分类，基于 SVM 的超平面分类更能体现样本间的关系。而决策树的效果表现最差，是由于样本数据中存在较多的非线形关系，单纯的线形分类器难以捕捉样本中的非线形关系。

## 2.5 总结与展望

在这一部分，我们探究了不同处理不均衡数据的方法对分类效果的影响即 4 种分类算法对于我们当前问题的分类效果。我们发现，SMOTEENN 的过采样方法对于当前问题处理不均衡数据的效果最佳，XGBoost 算法具有最高的分类精度，而 SVM 算法具有最高的阳性召回率。同时，尽管 XGBoost 算法分类精度高，但其在训练集上存在过拟合的现象，导致它在测试集上对阳性样本的分类效果受到一定限制。

在未来，我们会继续探究如何减少 XGBoost 算法的过拟合现象。同时，我们猜测阴性阳性之间的区分能较好用基于 DT 衍生的算法实现，但对于阳性样本内部的分类，类似 SVM 的超平面分类关系能更好地对阳性样本进行细化分类。未来我们会探究我们的这一猜测是否正确，并考虑构造双层分类器，首先用基于 DT 的分类器将阴性阳性样本分开，之后用 SVM 对分类为阳性的样本进行细化分类。

# 3 聚类模型

## 3.1 问题描述

我们认为这是个 9 分类的问题，即一种阴性和八个阳性，并且由于阳性样本过少，我将“可疑”、“强阳性”、“阳性”、“弱阳性”等属性均划分为阳性，一定程度上减少了样本的不均衡性。并且用 0-8 对这九个类别进行编号。0 代表阴性，1 代表 lgG-K 阳性，2 代表 lgG-λ阳性，3 代表 lgA-K 阳性，4 代表 lgA-λ阳性，5 代表 lgM-K 阳性，6 代表 lgM-λ阳性，7 代表 K 阳性，8 代表λ阳性。

## 3.2 不均衡数据处理（抽样方式）

由于阴性样本过多，在处理时不方便。所以在考虑了其余八类阳性样本的个数后，我在每次的训练过程中，随机从所有的阴性样本中抽取 300 个作为参与训练和测试的总体阴性样本，这样便达到了样本的均衡性。由于每次抽样都是随机的，所以对结果正确率并没有较大的影响。在这之后的所有聚类分析操作都是基于均衡化处理的，后续不再赘述。

## 3.3 训练集与测试集的划分

将每个图片转化为 125×6=750 行的列向量，将这 10000+的列向量合并成为一个大地数据矩阵，在九个类别中均按照 7:3 的比例划分训练集和测试集，这样保证了即使样本不均衡也能在测试集中遍布九个类别的样本。

## 3.4 数据降维方式

### 3.4.1 LDA 分解

划分为训练集与测试集之后分别对其进行中心化，然后将类别信息添加到训练集之中，进行 LDA 分解，得到 LDA 的降维矩阵，在降维后训练集矩阵上进行聚类的学习。同时用训练集得到的降维矩阵对测试集进行降维，得到降维后的测试集矩阵。这样实现了维度从 750 降为 8 维(由于是 9 分类问题，所以 LDA 分解降至 8 维就可以对数据进行有效分类)。

### 3.4.2 kernel PCA 分解

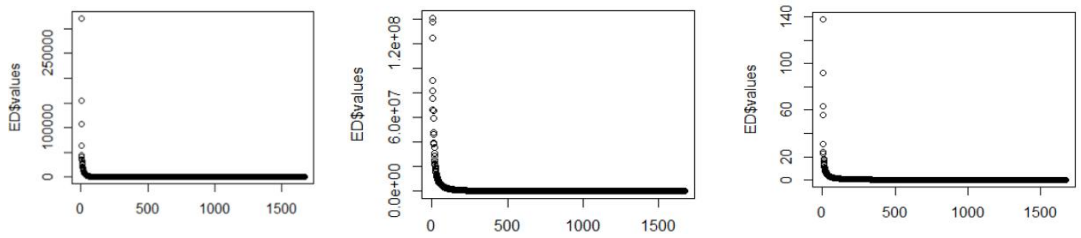
后续又采用了 kernePCA 对数据进行降维（由于 PCA 为 kernelPCA 的特例，所以就不单独赘述了）。

KernelPCA 降维中最重要的就是核函数的选择，我尝试了线性 kernel（也就是传统 PCA）、二次的 kernel 以及高斯 kernel，并且分别调整了各个核函数对应的内置参数，画出了三个崖底碎石土图（见附录），虽然三者的形状类似，但是尺度差别很大，高斯 kernel 的特征值尺度更为合理，可以在较少的维度提取较多的信息，所以以下均采用高斯 kernel，并且取前 30 个主成分降维。（注:二次核函数中的 a,c 高斯核函数中的σ都是需要调节的参数）

$$k(x,y) = x^T y \ (\textit{线性核函数})$$

$$k(x,y) = (ax^T y + c)^2 (\textit{二次核函数})$$

$$k(x,y) = e^{-\frac{||x-y||^2}{2\sigma^2}} (\textit{高斯核函数})$$




（自左至右分别是线性、二次、高斯核函数进行 KPCA 的崖底碎石土）

## 3.5 聚类算法

由于聚类的算法本身是无监督学习，会忽略掉标签信息，所以我对 K-means 聚类和 GMM 聚类方法进行了一定改进，让其能更合理地利用标签的信息。

### 3.5.1K-means 聚类

在降维后训练集样本上，计算每个类别样本里数据的均值向量（认为是该类别样本的中心点），以及该类别样本的协方差矩阵。

对于测试集中的每个点，计算其与这九个类别样本中心的的距离，看其与哪个类别的中心最近，则认为其分类到哪个类别当中。这里的距离我们尝试欧式距离与马氏距离，后来我们发现在高斯核下的 KPCA 降维数据中，应用马氏距离分类效果显著优秀于其他分类方法的效果。

$$\mu_i = \frac{1}{N_i}\sum_{j=1}^{N_i} xij \quad i = 0,1,2\ldots,8 \; j = 1,2,\ldots,N_i$$

$$\sigma_i = \frac{1}{N_i - 1}\sum_{j=1}^{N_i} (x_{ij} - \mu_i)(x_{ij} - \mu_i)^T \quad i = 0,1,2\ldots,8 \; j = 1,2,\ldots,N_i$$

$$d_{ik} = (x_k - \mu_i)^T \sigma_i^{-1}(x_k - \mu_i) \quad i = 0,1,2\ldots,8 \; k = 1,2,\ldots,N_{test}$$

（以上是计算训练集中心、协方差以及计算马氏距离的公式）

### 3.5.2 GMM 聚类

在降维后的训练集样本中，对每一类的样本进行高斯混合模型（GMM）的训练，即认为每一类样本均服从以下分布。

$$X|Y = i \sim p_{1i} * Z_{1i} + \ldots + p_{L_i i} * Z_{L_i i}, i = 0,1,2,\ldots 8$$

$$Z_{ji} \sim N(\mu_{ji}, \sigma_{ji} * I), i = 0,1,\ldots 8, j = 1,2,.L_i.$$

其中 i 表示训练集的类别编号，这里 $L_i$， $\mu_{ji}$， $\sigma_{ji}$ 都是需要估计的参数，再不断迭代至稳定后，将训练集的每一个点带入 GMM 模型中，可以得到 P（X|Y = i），即若当前样本属于第 i 类则出现该样本的概率为多少，比较九个条件概率，取令该概率最大的那个类别作为最后的分类结果。

## 3.6 分类结果分析（详细结果见附录）

我们采用了两种降维方式，两种聚类方法，并添加 KNN 作为对照组，得到最后的分类正确率结果如下。（注：这里 KNN 的 k 值是经过交叉验证后取得最优数值计算所得）

| 正确率 | KNN | GMM | K-means（马氏距离） | K-means（欧氏距离） |
|---|---|---|---|---|
| KPCA（高斯核） | 55.60% | 41.21% | 62.24% | 39.00% |
| LDA | 41.49% | 43.43% | 44.53% | 44.81% |

（分类正确率表格）

从表格中我们可以得出以下结论：

1) 大部分的聚类算法并没有 KNN 这一常见的分类算法效果好，由于聚类算法本就是无监督学习的方法，我们这里迁移过来应用到分类上，效果并没有胜过基础的分类算法。
2) 基于高斯核 PCA 降维后数据上进行马氏距离的 K-means 聚类效果，显著地胜过了其他方法，而同样的降维方式，只是采取了欧式距离后分类正确率就十分低，这说明了高斯核映射到高维的方式，与马氏距离十分契合，二者可以产生“1 + 1 > 2”的效果
3) LDA 分解得到的数据无论在那种分类方法上，分类结果都没较大的差别，相对于 KPCA 来说更加稳定。

总体来说，由于目标问题是九分类的问题，而且在观察了大量分类失败后的样本后发现，很多弱阳性和可疑的样本识别有很大的主观因素。而且通过附录中的各个分类的具体正确率可以看出，在阳性样本的正确率高于阴性样本，这样对临床来说是更易接受的，所以可以认为 62%的正确率是可以接受的。

### 3.7 方法的优劣性与展望

以上的方法还有很多不足的地方需要改进。比如有很多参数的选择并没有进行严格的交叉验证：随机抽取 300 个阴性样本来进行训练，高斯 kernel 函数中的 σ 大小，以及有没有比马氏距离更好的距离定义。等等这些问题的相关参数都没有严格的理论证明。

而且为了减少复杂度，对图片的列进行了压缩，压缩为 6 维，虽然这样来看并没有损失太多的信息，但是一定程度上还是有信息的损失。

不过应用传统的机器学习方法也有相对对深度学习很好的优势，最大的优势就是复杂度，以上所说的所有分类器的训练都是 2 分钟以内可以完成的。而且传统的机器学习具有很好的可解释性，所有的理论操作都是有意义的，而不是单纯的调参。

我觉得如果要继续提高算法效率，可以从高斯 kernelPCA 与马氏距离的契合度这个方面进行考虑，寻找比 k-means 方法更能提高二者契合度的方法比如层次聚类的方法，可能会有更好的分类正确率。以及还可采用 sparse kernel PCA 进行后续的尝试。

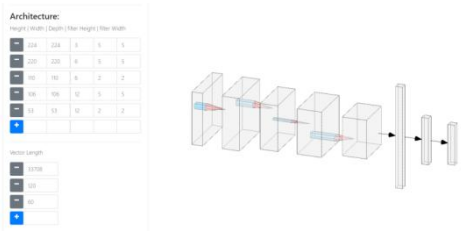
## 4 神经网络

### 4.1 前言

很明显这是一个图像分类问题，而在图像分类上卷积神经网络（CNN）有着巨大的优势。查阅文献得知，在 CNN 网络的发展历史上，有着两座伟大的里程碑，其一是首次在 ImageNet 上大放异彩的 AlexNet，其二是由何凯明团队发明的可以将网络拓展到几千层深度的 ResNet。我们根据前人的成果，尝试着做出这两种神经网络来解决问题。此外，GOOGLE 等团队提出了 AutoEncoder 模型（以下简称 AE），实现了利用神经网络对图片进行非线性压缩，其效果显著好于 PCA 等传统的线性压缩方法。我们尝试写出简单的 AE 模型，将压缩后的数据用传统机器学习方法进行分类，试图达到将传统方法与神经网络方法结合，既提高准确率又节省算力的效果，不过该部分结果不尽人意，展望部分会提出可能的解决方案。

### 4.2 AlexNet

我们仿照 AlexNet 的 8 层神经网络模型搭建出了 5 层 CNN 神经网络，网络结构如下：



输入为 $224 \times 224 \times 3$ 的图片，输入图片通过 torchvision 库的 transforms 函数进行尺寸归一化、中心化、转化为 tensor 等预处理。处理后的图片将分别进入两层卷积层和三层全连接层，卷积层之间有 Maxpool 池化层连接，可有效降低数据的维度；我在线性层之间添加了 Dropout 层，即每次训练时随机激活一半的神经元，这样可有效防止模型过拟合。最终训练得到的模型在测试集上可以得到 86.05%的准确率。

## 4.3 ResNet

ResNet 是 CNN 神经网络的另一座里程碑，我们深入了解其原理后自己动手搭建了 18 层和 34 层的 ResNet 网络。

ResNet 可以将网络层数达到一个特别恐怖的深度，其核心在于通过残差训练解决网络退化的问题。简单来说就是，随着神经网络层数加深，如果要求其至少不会时效果变差，应该有恒等映射 $H(X) = X$，但事实是神经网络很难训练恒等映射，这时不妨构造函数 $H(X) = F(X) + X$，这样通过训练更容易的残差 $F(X) = H(X) - X$ 即可达到目的。核心结构如下：

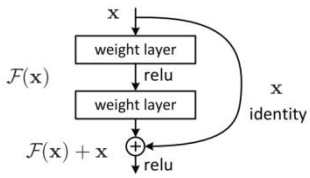



Figure 2. Residual learning: a building block.

用我们的 18 层 ResNet 网络最后可以得到 89.34%的准确率。用我们的 34 层 ResNet 网络最后可以得到 91.44%的准确率。准确率虽然与传统的机器学习方法相比有长足的提升，但是我们在训练模型的时候发现有一下问题。一、模型的参数巨大，在本地的笔记本自带的独显 GTX 1050（4G）上跑模型 ResNet34，常常出现显存不够的问题，只能降低 BatchSize，防止显存空间的溢出问题，但是这又会导致每个 epoch 的时间花费巨大，所以我们组最后在一台 RTX 2080Ti 上才勉强完成此次神经网络的训练任务。二、模型超参数众多，很多超参数之间相互耦合，会导致后期调参的时间花费巨大。三、由于 ResNet 理论上来说随着网络的深度增加其性能理论上应该更好，但是实践过程中我们发现由于显卡设备等的限制，我们无法继续堆叠网络的深度。

## 4.4 不足与改进

在尝试用神经网络解决本作业分类问题时，我们沿着前人走出的路，从 AlexNet 到 ResNet，在欣赏 CNN 波澜壮阔的发展历史的同时，我们也尝试复现了几种经典网络结构，有成功有失败，其中比较成功的网络得到了非常高的准确率，并且在结构的变化上有着较强的灵活性。

但是我们也遇到了一些问题，比如我在尝试设计 AutoEncoder 的时候，训练时的 loss 居高不下，颤来颤去，无论怎么调参都没法得到较好的结果，但是在 MNIST 数据集上就能得到很好的结果。我初步分析认为这是因为这种比较简单的只由卷积层和池化层组成的 AE 网络无法适应像素较高的图片，经过查阅文献我得到了如下可能的解决方法：

1. 在 2016 年提出 GDN 模型在神经网络压缩方面有奇效；

2. GOOGLE 在 2018 年提出实际上这种 AE 模型得到的隐变量之间有着空间依赖性，为了

使其相对独立，需要再加一个 hyperprior 层，简言之就是用两层网络来压缩图片。

还比如，由于时间有限，我们没办法与我们现有的 ResNet 做到更深，我觉得后续可以继续优化此网络，类似 AlexNet 的首篇文章一样，讲后续模型的训练采用双显卡训练的方式，从而解决训练过程中的显存瓶颈问题。还有考虑到此问题的应用的实际场景的限制，医用的嵌入式设备由于成本和可靠性相关方面的限制，先天没有我们训练网络的 GPU 工作站的性能好，即使我们训练出相关的模型，由于参数众多，对于设备的储存空间要求还是挺高的，后续我们可以考虑如何让此模型能够有更广的使用场景，可以采用现阶段比较流行的 MobilelNet，或者边缘计算等的理念继续优化我们的网络，使之有更加广阔的使用前景。最后 CV 领域还有着太多的知识，我觉得神经网络在 CV 领域的发展就像是一幅画卷，很开心做这次大作业，它让我窥见了这画卷的一角，并深深吸引了我。

# 5 项目总结与展望

我们通过神经网络，传统机器学习以及二者结合地方式对该分类问题进行了以上研究。不难看出神经网络与传统的机器学习相比，二者各有优劣。前者的不需要对图片进行过多的降维等预处理，而且具有更高的准确率，但是运行起来很耗费时间，复杂度过高。而后者则需要更多的降维处理，以及正确可能并不会很高，但是复杂度低，同时模型的解释性更强。

在传统的机器学习方法中，我们又采取了分类和聚类两大类方法来对数据进行处理，虽然后者原本是针对无监督学习的算法，但是经过改进后，我们对于算法本身有了更深层次的理解，而非简单的掉包，我觉得这一点让我们收获很多。

如果时间足够，我觉得我们可以尝试将神经网络与传统机器学习结合的方法做的更好，达到在较小的时间复杂度取得更高正确率的分类结果。或者尝试采用轻量级的神经网络，也可以达到上述目标。

最后感谢老师和助教这大半个学期的辛苦付出，我们在这门课上以及大作业的书写过程中收获了很多知识与技能，对机器学习有了更深入的理解。

# 6 附录

## 6.1 分类模型部分附录

未处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 64.06%/57.03% | 67.32%/34.90% | 68.49%/38.54% | 73.70%/52.86% |

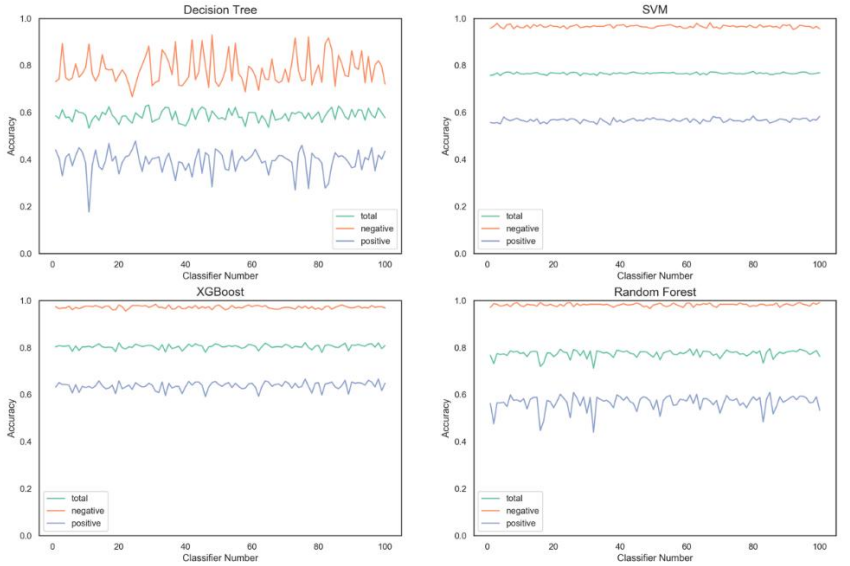
加权法处理不均衡的数据时分类准确率

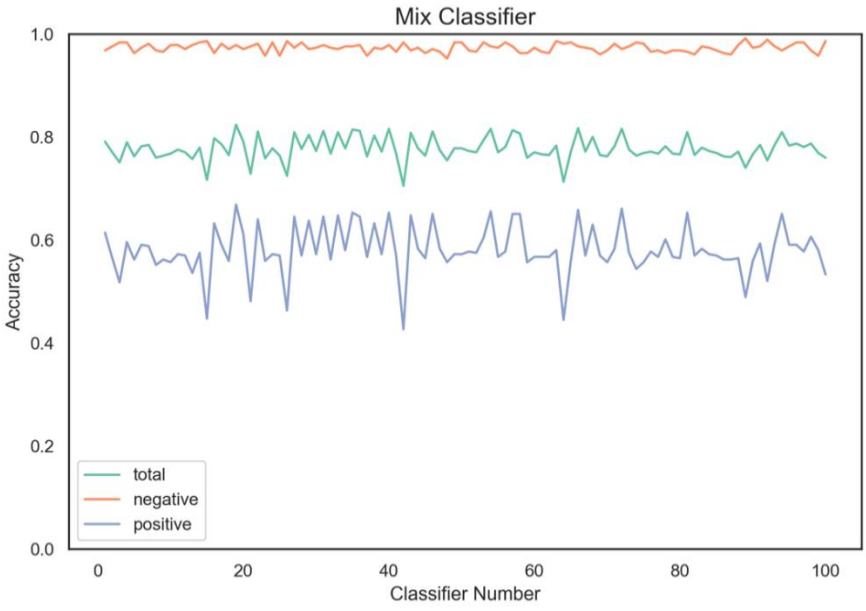
| 方法 | Decision Tree | Random Forest | SVM |
|---|---|---|---|
| 正确率（整体/阳性） | 61.33%/63.02% | 69.66%/39.84% | 73.57%/50% |

朴素随机欠采样法处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 58.60%/44.01% | 76.69%/56.25% | 75.78%/55.73% | 80.34%/63.28% |

不同算法对应的加权分类器随着内部分类器数量的增多分类准确率的变化



朴素随机过采样法处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 57.29%/44.01% | 72.53%/45.31% | 75.74%/76.30% | 78.91%/61.46% |

SMOTE 处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 55.47%/72.14% | 78.65%60.42% | 75.13%/75.00% | 81.38%/69.01% |

ADASYN 处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 53.65%/73.96% | 78.65%/59.90% | 74.48%/75.00% | 80.99%/68.49% |

SMOTE+ENN 处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 45.44%/78.39% | 78.39%/66.15% | 70.57%/82.03% | 81.38%/75.78% |

SMOTE+ Tomek 处理不均衡的数据时分类准确率

| 方法 | Decision Tree | Random Forest | SVM | XGBoost |
|---|---|---|---|---|
| 正确率（整体/阳性） | 56.38%/71.61% | 78.26%/58.59% | 74.35%/74.48% | 81.25%/68.23% |

## 6.2 聚类模型部分附录

LDA + KNN

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 22 | 20 | 30 | 9 | 29 | 2 | 1 | 0 | 6 | |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 12 | 70 | 11 | 10 | 11 | 6 | 2 | 2 | 4 | |
| 2 | 23 | 14 | 75 | 9 | 21 | 2 | 7 | 4 | 11 | |
| 3 | 3 | 2 | 6 | 28 | 9 | 2 | 1 | 3 | 1 | |
| 4 | 25 | 15 | 31 | 9 | 68 | 1 | 3 | 1 | 8 | |
| 5 | 1 | 1 | 1 | 0 | 0 | 18 | 3 | 0 | 0 | |
| 6 | 1 | 1 | 2 | 0 | 2 | 7 | 3 | 0 | 1 | |
| 7 | 1 | 0 | 3 | 1 | 4 | 1 | 0 | 3 | 1 | |
| 8 | 2 | 2 | 8 | 3 | 9 | 3 | 2 | 3 | 7 | |
| Total | 24.4% | 56.0% | 44.9% | 40.6% | 44.4% | 42.9% | 13.6% | 18.8% | 17.9% | 41.49% |

LDA + GMM

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 29 | 13 | 34 | 1 | 30 | 5 | 2 | 4 | 4 | |
| 1 | 12 | 62 | 11 | 6 | 7 | 1 | 0 | 2 | 2 | |
| 2 | 9 | 14 | 64 | 2 | 11 | 1 | 6 | 1 | 13 | |
| 3 | 9 | 7 | 8 | 46 | 17 | 3 | 0 | 1 | 4 | |
| 4 | 15 | 15 | 19 | 9 | 66 | 0 | 2 | 0 | 3 | |
| 5 | 1 | 3 | 4 | 0 | 3 | 26 | 7 | 0 | 0 | |
| 6 | 5 | 1 | 10 | 1 | 2 | 5 | 4 | 0 | 1 | |
| 7 | 2 | 6 | 4 | 1 | 1 | 1 | 0 | 4 | 4 | |
| 8 | 8 | 4 | 13 | 3 | 16 | 0 | 1 | 4 | 8 | |
| Total | 32.2% | 49.6% | 38.3% | 66.7% | 43.1% | 61.9% | 18.2% | 25.0% | 20.5% | 43.43% |

LDA + K-means（马氏距离）

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 27 | 15 | 29 | 2 | 28 | 5 | 3 | 4 | 3 | |
| 1 | 10 | 62 | 11 | 6 | 12 | 0 | 1 | 2 | 3 | |
| 2 | 13 | 14 | 69 | 2 | 9 | 1 | 3 | 1 | 10 | |
| 3 | 11 | 9 | 5 | 48 | 20 | 3 | 2 | 1 | 4 | |
| 4 | 13 | 12 | 17 | 8 | 65 | 0 | 1 | 0 | 3 | |
| 5 | 1 | 3 | 5 | 0 | 2 | 27 | 4 | 0 | 2 | |
| 6 | 6 | 3 | 9 | 1 | 1 | 5 | 7 | 0 | 3 | |
| 7 | 1 | 4 | 6 | 0 | 2 | 1 | 0 | 4 | 4 | |
| 8 | 8 | 3 | 16 | 2 | 14 | 0 | 1 | 4 | 7 | |
| Total | 30.0% | 49.6% | 41.3% | 69.6% | 42.5% | 64.3% | 31.8% | 25.0% | 17.9% | 44.53% |

LDA + K-means（欧氏距离）

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 33 | 23 | 31 | 12 | 29 | 1 | 3 | 0 | 7 | |
| 1 | 11 | 72 | 11 | 9 | 11 | 4 | 1 | 2 | 5 | |
| 2 | 15 | 7 | 75 | 6 | 17 | 3 | 5 | 4 | 7 | |
| 3 | 3 | 2 | 6 | 32 | 12 | 2 | 0 | 3 | 0 | |
| 4 | 15 | 7 | 24 | 5 | 66 | 0 | 1 | 1 | 5 | |
| 5 | 3 | 2 | 1 | 0 | 1 | 19 | 4 | 0 | 0 | |
| 6 | 1 | 3 | 2 | 0 | 2 | 10 | 7 | 0 | 2 | |
| 7 | 2 | 2 | 4 | 2 | 4 | 0 | 0 | 3 | 2 | |
| 8 | 7 | 7 | 13 | 3 | 11 | 3 | 1 | 3 | 11 | |
| Total | 36.7% | 57.6% | 44.9% | 46.4% | 43.1% | 45.2% | 31.8% | 18.8% | 28.2% | 44.81% |

KPCA + KNN

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 36 | 13 | 24 | 5 | 22 | 4 | 2 | 1 | 1 | |
| 1 | 7 | 74 | 4 | 2 | 4 | 1 | 0 | 3 | 0 | |
| 2 | 12 | 10 | 103 | 3 | 15 | 5 | 9 | 2 | 16 | |
| 3 | 1 | 1 | 1 | 37 | 1 | 0 | 0 | 0 | 2 | |
| 4 | 30 | 24 | 32 | 21 | 110 | 4 | 3 | 2 | 13 | |
| 5 | 4 | 1 | 0 | 1 | 0 | 27 | 5 | 0 | 0 | |
| 6 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | |
| 7 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 6 | 3 | |
| 8 | 0 | 1 | 2 | 0 | 1 | 1 | 0 | 2 | 4 | |
| Total | 40.0% | 59.2% | 61.7% | 53.6% | 71.9% | 64.3% | 4.5% | 37.5% | 10.3% | 55.60% |

KPCA + GMM

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 32 | 18 | 31 | 8 | 21 | 2 | 0 | 1 | 4 | |
| 1 | 7 | 64 | 10 | 2 | 8 | 4 | 0 | 1 | 1 | |
| 2 | 11 | 5 | 61 | 5 | 12 | 0 | 3 | 1 | 7 | |
| 3 | 4 | 3 | 6 | 32 | 13 | 1 | 0 | 1 | 1 | |
| 4 | 12 | 9 | 15 | 12 | 61 | 1 | 1 | 0 | 0 | |
| 5 | 4 | 7 | 8 | 2 | 8 | 24 | 12 | 1 | 2 | |
| 6 | 3 | 5 | 11 | 1 | 8 | 7 | 3 | 0 | 1 | |
| 7 | 7 | 9 | 7 | 7 | 5 | 1 | 2 | 8 | 13 | |
| 8 | 10 | 5 | 18 | 0 | 17 | 2 | 1 | 3 | 10 | |
| Total | 35.6% | 51.2% | 36.5% | 46.4% | 39.9% | 57.1% | 13.6% | 50.0% | 25.6% | 41.21% |

KPCA + K-means（马氏距离）

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 56 | 10 | 26 | 26 | 27 | 10 | 3 | 1 | 6 | |
| 1 | 10 | 94 | 2 | 2 | 5 | 5 | 0 | 4 | 1 | |
| 2 | 13 | 12 | 115 | 0 | 13 | 4 | 11 | 2 | 13 | |
| 3 | 1 | 12 | 1 | 46 | 0 | 2 | 0 | 3 | 3 | |
| 4 | 7 | 3 | 16 | 6 | 102 | 0 | 3 | 1 | 4 | |
| 5 | 1 | 3 | 0 | 0 | 0 | 19 | 1 | 0 | 0 | |
| 6 | 1 | 1 | 2 | 1 | 1 | 2 | 3 | 1 | 0 | |
| 7 | 1 | 1 | 0 | 0 | 2 | 1 | 0 | 10 | 0 | |
| 8 | 1 | 2 | 5 | 0 | 5 | 0 | 1 | 4 | 12 | |
| Total | 61.5% | 68.1% | 68.9% | 56.8% | 65.8% | 44.2% | 13.6% | 38.5% | 30.8% | 62.24% |

KPCA + K-means（欧氏距离）

| 真实分类 / 预测分类 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 41 | 24 | 29 | 11 | 35 | 7 | 1 | 1 | 5 | |
| 1 | 6 | 52 | 7 | 1 | 6 | 2 | 0 | 1 | 1 | |
| 2 | 3 | 1 | 39 | 0 | 1 | 0 | 2 | 0 | 2 | |
| 3 | 0 | 8 | 0 | 40 | 2 | 1 | 0 | 3 | 0 | |
| 4 | 3 | 1 | 6 | 4 | 58 | 0 | 0 | 0 | 2 | |
| 5 | 12 | 14 | 21 | 4 | 13 | 25 | 11 | 1 | 3 | |
| 6 | 10 | 8 | 27 | 3 | 14 | 4 | 4 | 0 | 1 | |
| 7 | 10 | 7 | 19 | 5 | 15 | 3 | 4 | 9 | 15 | |
| 8 | 5 | 10 | 19 | 1 | 9 | 0 | 0 | 1 | 10 | |
| Total | 45.6% | 41.6% | 23.4% | 58.0% | 37.9% | 59.5% | 18.2% | 56.3% | 25.6% | 39% |

## 6.3 神经网络部分附录

**参考文献：**

AlexNet:ImageNet Classification with Deep Convolutional Neural Networks
ResNet:Deep Residual Learning for Image Recognition
GDN：End-to-end Optimized Image Compression
AutoEncoder：Variational image compression with a scale hyperprior

**AlexNet 训练结果：**

[Train] Epoch 1: accuracy = 5991 out of 7297
[Train] Epoch 2: accuracy = 5991 out of 7297
[Train] Epoch 3: accuracy = 5991 out of 7297
[Train] Epoch 4: accuracy = 5991 out of 7297

[Train] Epoch 5: accuracy = 5991 out of 7297
[Train] Epoch 6: accuracy = 6026 out of 7297
[Train] Epoch 7: accuracy = 6026 out of 7297
[Train] Epoch 8: accuracy = 6044 out of 7297
[Train] Epoch 9: accuracy = 6044 out of 7297
[Train] Epoch 10: accuracy = 6062 out of 7297
[Train] Epoch 11: accuracy = 6062 out of 7297
[Train] Epoch 12: accuracy = 6062 out of 7297
[Train] Epoch 13: accuracy = 6077 out of 7297
[Train] Epoch 14: accuracy = 6095 out of 7297
[Train] Epoch 15: accuracy = 6095 out of 7297
[Train] Epoch 16: accuracy = 6095 out of 7297
[Train] Epoch 17: accuracy = 6113 out of 7297
[Train] Epoch 18: accuracy = 6133 out of 7297
[Train] Epoch 19: accuracy = 6148 out of 7297
[Train] Epoch 20: accuracy = 6148 out of 7297
[Train] Epoch 21: accuracy = 6148 out of 7297
[Train] Epoch 22: accuracy = 6164 out of 7297
[Train] Epoch 23: accuracy = 6180 out of 7297
[Train] Epoch 24: accuracy = 6180 out of 7297
[Train] Epoch 25: accuracy = 6180 out of 7297
[Train] Epoch 26: accuracy = 6200 out of 7297
[Train] Epoch 27: accuracy = 6216 out of 7297
[Train] Epoch 28: accuracy = 6216 out of 7297
[Train] Epoch 29: accuracy = 6235 out of 7297
[Train] Epoch 30: accuracy = 6235 out of 7297
[Train] Epoch 31: accuracy = 6252 out of 7297
[Train] Epoch 32: accuracy = 6270 out of 7297
[Train] Epoch 33: accuracy = 6270 out of 7297
[Train] Epoch 34: accuracy = 6290 out of 7297
[Train] Epoch 35: accuracy = 6290 out of 7297
[Train] Epoch 36: accuracy = 6290 out of 7297
[Train] Epoch 37: accuracy = 6306 out of 7297
[Train] Epoch 38: accuracy = 6306 out of 7297
[Train] Epoch 39: accuracy = 6306 out of 7297
[Train] Epoch 40: accuracy = 6306 out of 7297
[Train] Epoch 41: accuracy = 6322 out of 7297
[Train] Epoch 42: accuracy = 6340 out of 7297
[Train] Epoch 43: accuracy = 6340 out of 7297
[Train] Epoch 44: accuracy = 6340 out of 7297
[Train] Epoch 45: accuracy = 6340 out of 7297
[Train] Epoch 46: accuracy = 6355 out of 7297
[Train] Epoch 47: accuracy = 6355 out of 7297
[Train] Epoch 48: accuracy = 6374 out of 7297

[Train] Epoch 49: accuracy = 6374 out of 7297
[Train] Epoch 50: accuracy = 6374 out of 7297
[Train] Epoch 51: accuracy = 6374 out of 7297
[Train] Epoch 52: accuracy = 6392 out of 7297
[Train] Epoch 53: accuracy = 6392 out of 7297
[Train] Epoch 54: accuracy = 6392 out of 7297
[Train] Epoch 55: accuracy = 6410 out of 7297
[Train] Epoch 56: accuracy = 6428 out of 7297
[Train] Epoch 57: accuracy = 6428 out of 7297
[Train] Epoch 58: accuracy = 6446 out of 7297
[Train] Epoch 59: accuracy = 6446 out of 7297
[Train] Epoch 60: accuracy = 6446 out of 7297
[Train] Epoch 61: accuracy = 6446 out of 7297
[Train] Epoch 62: accuracy = 6446 out of 7297
[Train] Epoch 63: accuracy = 6446 out of 7297
[Train] Epoch 64: accuracy = 6446 out of 7297
[Train] Epoch 65: accuracy = 6446 out of 7297
[Train] Epoch 66: accuracy = 6446 out of 7297
[Test] accuracy = 878 out of 1015

**ResNet18 训练结果：**

Epoch 1/50
Batch 72, Test loss:0.0059, Test acc:0.8274, Time :49.7409461983994
Epoch 2/50
Batch 72, Test loss:0.0059, Test acc:0.8274, Time :50.02955351927077
Epoch 3/50
Batch 72, Test loss:0.0057, Test acc:0.8294, Time :44.64022463625481
Epoch 4/50
Batch 72, Test loss:0.0057, Test acc:0.8294, Time :44.45926332270611
Epoch 5/50
Batch 72, Test loss:0.0055, Test acc:0.8354, Time :46.338936789109766
Epoch 6/50
Batch 72, Test loss:0.0055, Test acc:0.8354, Time :50.72886502766587
Epoch 7/50
Batch 72, Test loss:0.0051, Test acc:0.8394, Time :48.80549518631009
Epoch 8/50
Batch 72, Test loss:0.0051, Test acc:0.8394, Time :43.21594635619103
Epoch 9/50
Batch 72, Test loss:0.0049, Test acc:0.8454, Time :45.21233039540178
Epoch 10/50
Batch 72, Test loss:0.0049, Test acc:0.8454, Time :49.9967231946869
Epoch 11/50
Batch 72, Test loss:0.0049, Test acc:0.8454, Time :45.06144423219545
Epoch 12/50

Batch 72, Test loss:0.0047, Test acc:0.8494, Time :45.58521664946801
Epoch 13/50
Batch 72, Test loss:0.0045, Test acc:0.8554, Time :44.98979841108145
Epoch 14/50
Batch 72, Test loss:0.0043, Test acc:0.8574, Time :50.47611372282514
Epoch 15/50
Batch 72, Test loss:0.0043, Test acc:0.8574, Time :47.47551947315474
Epoch 16/50
Batch 72, Test loss:0.0043, Test acc:0.8574, Time :49.879370645820664
Epoch 17/50
Batch 72, Test loss:0.0041, Test acc:0.8594, Time :50.434885573163605
Epoch 18/50
Batch 72, Test loss:0.0041, Test acc:0.8594, Time :46.9517793078346
Epoch 19/50
Batch 72, Test loss:0.0037, Test acc:0.8654, Time :44.02649721427392
Epoch 20/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :50.76351965302812
Epoch 21/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :45.3835283106042
Epoch 22/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :43.95406543855629
Epoch 23/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :44.64374653328912
Epoch 24/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :45.03364068041068
Epoch 25/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :46.255245752483965
Epoch 26/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :47.24689247455667
Epoch 27/50
Batch 72, Test loss:0.0033, Test acc:0.8714, Time :47.68910471042759
Epoch 28/50
Batch 72, Test loss:0.0029, Test acc:0.8754, Time :43.20934172310024
Epoch 29/50
Batch 72, Test loss:0.0029, Test acc:0.8754, Time :46.052836342421145
Epoch 30/50
Batch 72, Test loss:0.0029, Test acc:0.8754, Time :46.23309102752622
Epoch 31/50
Batch 72, Test loss:0.0029, Test acc:0.8754, Time :43.69430283374125
Epoch 32/50
Batch 72, Test loss:0.0029, Test acc:0.8754, Time :46.64662317548148
Epoch 33/50
Batch 72, Test loss:0.0029, Test acc:0.8754, Time :48.957867311148505
Epoch 34/50

Batch 72, Test loss:0.0027, Test acc:0.8814, Time :48.59072494988439
Epoch 35/50
Batch 72, Test loss:0.0027, Test acc:0.8814, Time :49.203805101823036
Epoch 36/50
Batch 72, Test loss:0.0027, Test acc:0.8814, Time :46.52812075289481
Epoch 37/50
Batch 72, Test loss:0.0025, Test acc:0.8834, Time :49.10649019561683
Epoch 38/50
Batch 72, Test loss:0.0025, Test acc:0.8834, Time :47.68277582540116
Epoch 39/50
Batch 72, Test loss:0.0021, Test acc:0.8854, Time :44.87976743901976
Epoch 40/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :48.61707963837438
Epoch 41/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :50.769190185809855
Epoch 42/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :50.756870791868735
Epoch 43/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :46.185546004008174
Epoch 44/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :49.82293945538806
Epoch 45/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :45.85651529298659
Epoch 46/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :46.144477458748845
Epoch 47/50
Batch 72, Test loss:0.0019, Test acc:0.8894, Time :47.918186295494706
Epoch 48/50
Batch 72, Test loss:0.0017, Test acc:0.8934, Time :48.5382330087145
Epoch 49/50
Batch 72, Test loss:0.0017, Test acc:0.8934, Time :50.02247552026672
Epoch 50/50
Batch 72, Test loss:0.0017, Test acc:0.8934, Time :48.62315276555704

**ResNet34 训练结果：**

Epoch 1/50
Batch 25, Test loss:0.0059, Test acc:0.8244, Time :70.52174002697711
Epoch 2/50
Batch 25, Test loss:0.0057, Test acc:0.8284, Time :74.65311990330581
Epoch 3/50
Batch 25, Test loss:0.0057, Test acc:0.8284, Time :72.49397571607676
Epoch 4/50
Batch 25, Test loss:0.0057, Test acc:0.8284, Time :71.51714274423279
Epoch 5/50

Batch 25, Test loss:0.0054, Test acc:0.8324, Time :73.44128971631875
Epoch 6/50
Batch 25, Test loss:0.0054, Test acc:0.8324, Time :73.70171488976027
Epoch 7/50
Batch 25, Test loss:0.0054, Test acc:0.8324, Time :71.64611599764953
Epoch 8/50
Batch 25, Test loss:0.0052, Test acc:0.8384, Time :74.12511847509946
Epoch 9/50
Batch 25, Test loss:0.0052, Test acc:0.8384, Time :72.73299236009377
Epoch 10/50
Batch 25, Test loss:0.0049, Test acc:0.8424, Time :71.16054282955565
Epoch 11/50
Batch 25, Test loss:0.0047, Test acc:0.8464, Time :74.20266948682179
Epoch 12/50
Batch 25, Test loss:0.0047, Test acc:0.8464, Time :71.75148786165019
Epoch 13/50
Batch 25, Test loss:0.0045, Test acc:0.8484, Time :71.050676769187
Epoch 14/50
Batch 25, Test loss:0.0043, Test acc:0.8524, Time :70.84042596365941
Epoch 15/50
Batch 25, Test loss:0.0042, Test acc:0.8584, Time :72.71825585682498
Epoch 16/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :74.95614575885627
Epoch 17/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :70.9213897444928
Epoch 18/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :73.22311584411682
Epoch 19/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :72.55219260616948
Epoch 20/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :70.527302769037
Epoch 21/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :74.90941412312921
Epoch 22/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :72.59208361390122
Epoch 23/50
Batch 25, Test loss:0.0038, Test acc:0.8644, Time :71.0851732644944
Epoch 24/50
Batch 25, Test loss:0.0035, Test acc:0.8684, Time :74.08208396438086
Epoch 25/50
Batch 25, Test loss:0.0031, Test acc:0.8704, Time :73.5560030174866
Epoch 26/50
Batch 25, Test loss:0.0028, Test acc:0.8724, Time :70.03767624182653
Epoch 27/50

Batch 25, Test loss:0.0024, Test acc:0.8764, Time :70.31465925479836
Epoch 28/50
Batch 25, Test loss:0.0022, Test acc:0.8804, Time :70.08824193931802
Epoch 29/50
Batch 25, Test loss:0.0022, Test acc:0.8804, Time :72.26342175292868
Epoch 30/50
Batch 25, Test loss:0.0022, Test acc:0.8804, Time :73.83700027451896
Epoch 31/50
Batch 25, Test loss:0.0022, Test acc:0.8804, Time :72.19071384260273
Epoch 32/50
Batch 25, Test loss:0.0022, Test acc:0.8804, Time :72.26401615239492
Epoch 33/50
Batch 25, Test loss:0.0019, Test acc:0.8824, Time :72.68511721754125
Epoch 34/50
Batch 25, Test loss:0.0017, Test acc:0.8864, Time :74.92418001042948
Epoch 35/50
Batch 25, Test loss:0.0014, Test acc:0.8924, Time :73.53880198118023
Epoch 36/50
Batch 25, Test loss:0.0012, Test acc:0.8984, Time :72.50732812686402
Epoch 37/50
Batch 25, Test loss:0.0012, Test acc:0.8984, Time :70.20537433599775
Epoch 38/50
Batch 25, Test loss:0.0012, Test acc:0.8984, Time :71.55028437475329
Epoch 39/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :71.19248577189477
Epoch 40/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :71.69288570886735
Epoch 41/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :72.5481898865203
Epoch 42/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :73.8244171084131
Epoch 43/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :73.08798087301835
Epoch 44/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :71.5638105747023
Epoch 45/50
Batch 25, Test loss:0.0008, Test acc:0.9044, Time :71.44910938264644
Epoch 46/50
Batch 25, Test loss:0.0005, Test acc:0.9104, Time :74.19775504313961
Epoch 47/50
Batch 25, Test loss:0.0005, Test acc:0.9104, Time :74.27261710513159
Epoch 48/50
Batch 25, Test loss:0.0005, Test acc:0.9104, Time :74.64874406259105
Epoch 49/50

```
Batch 25, Test loss:0.0005, Test acc:0.9104, Time :73.67531107988727
Epoch 50/50
Batch 25, Test loss:0.0003, Test acc:0.9144, Time :72.3881657053713
```