模式识别与机器学习

实验报告

班级：10012003

姓名： 徐金海

学号：2020302703

目录

一、 实验目的

对从 Omniglot 数据集选择的 200 类（后发现有两类字符重复，更正为 198 类）手写字符进行划分，将其按照每类 3:1 的比例划分为训练集和测试集，通过对训练集进行学习构建卷积神经网络（CNN）模型和支持向量机（SVM）模型，对测试集的手写字符数据进行分类预测并说明预测效果。并尝试改进模型使预测效果提升。

二、 实验原理

2.1 数据来源：

助教老师提供的 NewDataset.mat 文件包含了 Omniglot 数据集中的 200 类手写字符数据，每次前 15 个样本作为训练集，后 5 个样本作为测试集。这是一个明显的监督式机器学习，我们可以根据测试集数据建立一个分类模型，再将测试集数据代入，并通过计算其分析准确率。

2.2 实验模型 CNN

本次实验主要使用卷积神经网络即 CNN 模型，其被广泛应用于图像处理领域。同时参考助教老师提供的代码示例，采用了 Python 的 PyTorch 框架，下面是一个基本的 CNN 模型的介绍。

卷积神经网络一般由卷积层、池化层和全连接层构成。

2.2.1 用卷积来代替全连接：

在传统的前馈神经网络中，如果第l层有M_l个神经元，第$l-1$层有个M_{l-1}个神经元，连接边有$M_l \times M_{l-1}$个，也就是权重矩阵有$M_l \times M_{l-1}$个参数。当M_l和M_{l-1}都很大时，权重矩阵的参数非常多，训练的效率会非常低。图 1 为全连接层的示意图。

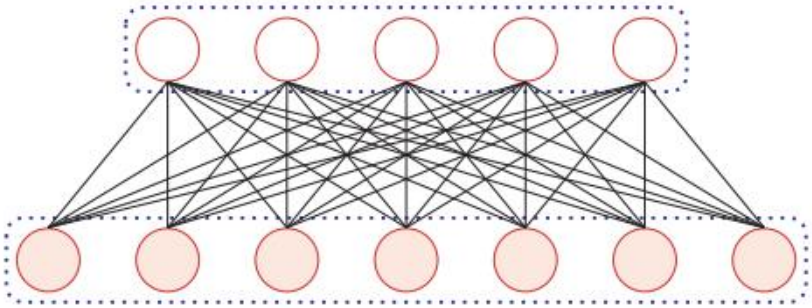


图 1：全连接层

在卷积神经网络中，我们采用卷积来代替全连接，第l层的净输入$\mathbf{z}^{(l)}$为第$l-1$层活性值$\boldsymbol{\alpha}^{(l-1)}$和卷积核$\boldsymbol{\omega}^{(l)} \in \mathbf{R}^K$的卷积，即

$$\mathbf{z}^{(l)} = \boldsymbol{\omega}^{(l)} \otimes \boldsymbol{\alpha}^{(l-1)} + b^{(l)} \tag{1}$$

其中卷积核 $\boldsymbol{\omega}^{(l)} \in \mathbf{R}^K$ 为可学习的权重向量，$b^{(l)} \in \mathbf{R}$ 为可学习的偏置。

根据卷积的定义，卷积层有两个很重要的性质：

①局部连接：

在卷积层（假设是第 l 层）中的每一个神经元都只和前一层（第 $l-1$ 层）中某个局部窗口内的神经元相连，构成一个局部连接网络。如图 2 所示，卷积层和前一层之间的连接数大大减少，由原来的 $M_l \times M_{l-1}$ 个连接变为 $M_l \times K$ 个连接，K 为卷积核大小。

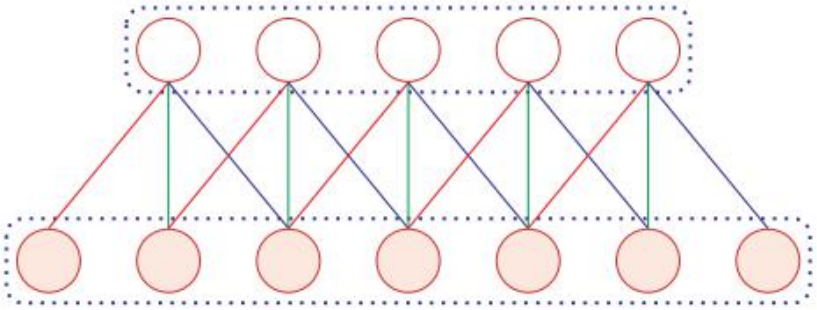


图 2：卷积层

②权重共享：

从公式(1)可以看出，作为参数的卷积核 $\boldsymbol{\omega}^{(l)}$ 对于第 l 层的所有的神经元都是相同的。如图 x 中，所有的同颜色连接上的权重是相同的。权重共享可以理解为一个卷积核只捕捉输入数据中的一种特定的局部特征。因此，如果要提取多种特征就需要使用多个不同的卷积核。

由于局部连接和权重共享，卷积层的参数只有一个 K 维的权重 $\boldsymbol{\omega}^{(l)}$ 和1维的偏置 $b^{(l)}$，共 $K+1$ 个参数。参数个数和神经元的数量无关．此外，第层的神经元个数不是任意选择的，而是满足 $M_l = M_{l-1} - K + 1$。

2.2.2 卷积层：

卷积层的作用是提取一个局部区域的特征，不同的卷积核相当于不同的特征提取器。上一节中描述的卷积层的神经元和全连接网络一样都是一维结构。由于卷积网络主要应用在图像处理上，而图像为二维结构，因此为了更充分地利用图像的局部信息，通常将神经元组织为三维结构的神经层，其大小为高度 $M \times$ 宽度 $N \times$ 深度 D，由 D 个 $M \times N$ 大小的特征映射构成。

特征映射（Feature Map）为一幅图像（或其他特征映射）在经过卷积提取到的特征，每个特征映射可以作为一类抽取的图像特征。为了提高卷积网络的表示能力，可以在每一层使用多个不同的特征映射，以更好地表示图像的特征。

在输入层，特征映射就是图像本身。如果是灰度图像，就是有一个特征映射，输入层的深度 $D=1$；如果是彩色图像，分别有 RGB 三个颜色通道的特征映射，输入层的深度 $D=3$。

不失一般性，一个卷积层的三维结构通常如图 3 所示：

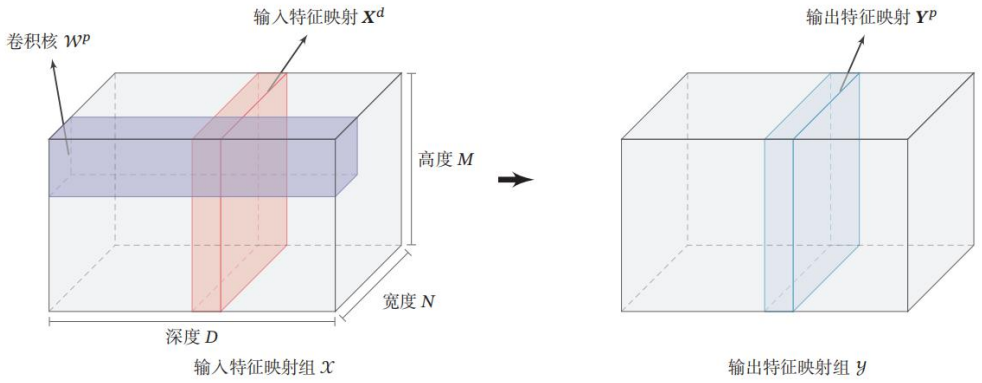


图 3：卷积层的三维结构表示

（1） 输入特征映射组：$x \in R^{M\times N\times D}$ 为三维张量（Tensor），其中每个切片（Slice）矩阵 $X^d \in R^{M\times N}$ 为一个输入特征映射，$1 \le d \le D$；

（2） 输出特征映射组：$y \in R^{M'\times N'\times D}$ 为三维张量，其中每个切片矩阵 $Y^p \in R^{M'\times N'}$ 为一个输出特征映射，$1 \le p \le P$；

（3） 卷积核：$\omega \in R^{U\times V\times P\times D}$ 为四维张量，其中每个切片矩阵 $W^{p,d} \in R^{U\times V}$ 为一个二维卷积核，$1 \le p \le P, 1 \le d \le D$。

为了计算输出特征映射 Y^p，用卷积核 $W^{p,1}, W^{p,2}, \cdots, W^{p,D}$ 分别对输入特征映射 $X^1, X^2, \cdots, X^D$ 进行卷积，然后将卷积结果相加，并加上一个标量偏置 b^p 到卷积层的净输入 Z^p，再经过非线性激活函数后得到输出特征映射 Y^p，如图 4 所示。

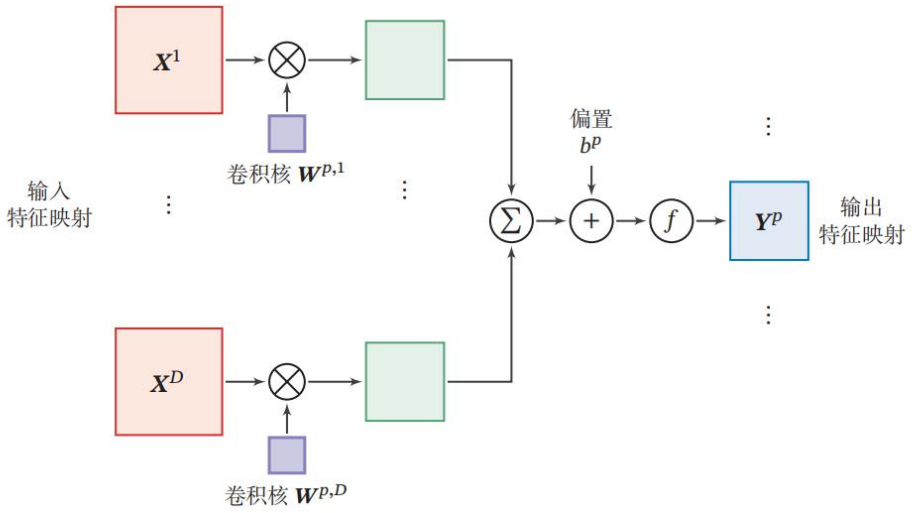


图 4：卷积层中从输入特征映射组 X 到输出特征映射 Y^p 的计算示例

$$Z^p = W^p \otimes X + b^p = \sum_{d=1}^{D} W^{p,d} \otimes X^d + b^p$$
$$Y^p = f(Z^p) \tag{2}$$

其中$W^p \in R^{U\times V\times D}$为三维卷积核，$f(\cdot)$为非线性激活函数，一般用 ReLU 函数。

整个计算过程如图 x 所示．如果希望卷积层输出P个特征映射，可以将上述计算过程重复P次，得到P个输出特征映射$Y^1, Y^2, \cdots, Y^P$。

在输入为$x \in R^{M\times N\times D}$，输出为$y \in R^{M'\times N'\times D}$的卷积层中，每一个输出特征映射都需要$D$个卷积核以及一个偏置．假设每个卷积核的大小为$U \times V$，那么共需要$P \times D \times (U \times V) + P$个参数。

2.2.3 池化层：

池化层（Pooling Layer）也叫子采样层（Subsampling Layer），其作用是进行特征选择，降低特征数量，从而减少参数数量。

卷积层虽然可以显著减少网络中连接的数量，但特征映射组中的神经元个数并没有显著减少．如果后面接一个分类器，分类器的输入维数依然很高，很容易出现过拟合。为了解决这个问题，可以在卷积层之后加上一个汇聚层，从而降低特征维数，避免过拟合。图 5 为池化层中汇聚过程示例。

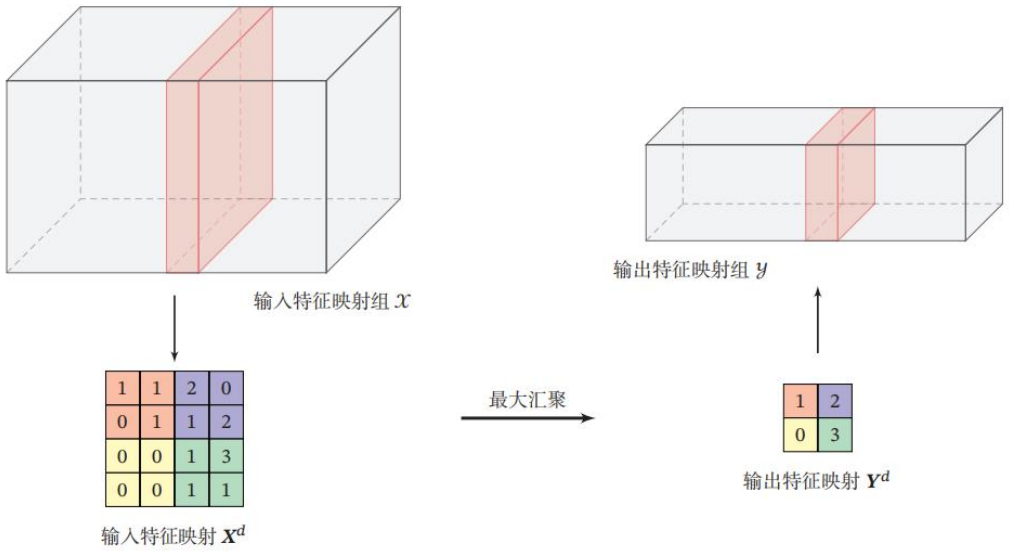


图 5：池化层中汇聚过程示例

典型的池化层是将每个特征映射划分为 2 × 2 大小的不重叠区域，然后使用最大汇聚的方式进行下采样。池化层也可以看作一个特殊的卷积层，卷积核大小为$K \times K$，步长为$S \times S$，卷积核为 max 函数或 mean 函数．过大的采样区域会急剧减少神经元的数量，也会造成过多的信息损失。

2.2.4 全连接层

全连接层的主要作用是将卷积层和池化层提取到的特征图转化为一个向量，然后通过一个全连接层对这个向量进行分类或回归等任务。这个全连接层中的每个神经元都与上一层中的所有神经元相连，因此它也被称为“密集连接层”。

在 CNN 中，全连接层通常出现在最后一层，用于分类或回归任务。在训练过程中，我们可以使用反向传播算法来调整全连接层中的权重和偏置，以优化模型的准确性和性能。

然而，由于全连接层的参数量较大，因此容易过拟合，导致模型的泛化能力不足。因此，在现代的 CNN 中，全连接层的使用已经被大量减少，取而代之的是使用全局平均池化层或卷积层来替代全连接层，以降低模型的复杂度和提高模型的泛化能力。

2.2.5 CNN 的整体结构：

一个典型的卷积网络是由卷积层、池化层、全连接层交叉堆叠而成．目前常用的卷积网络整体结构如图 6 所示。一个卷积块为连续 M 个卷积层和 b 个池化层（ M 通常设置为 $2\sim5$， b 为 0 或 1）。一个卷积网络中可以堆叠 K 个连续的卷积块，然后在后面接着 K 个全连接层（ N 的取值区间比较大，比如 $1\sim100$ 或者更大； K 一般为 $0\sim2$ ）。

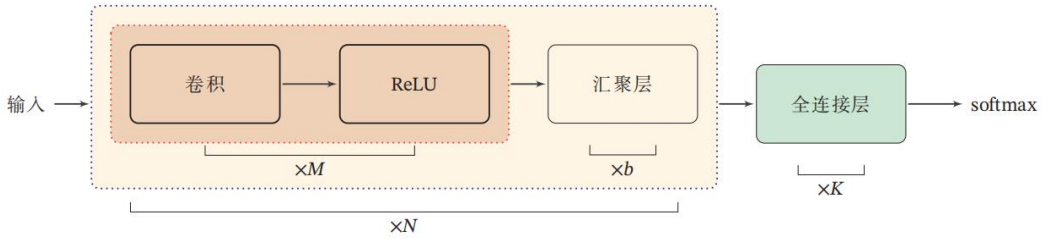


图 6：常用的卷积神经网络整体结构

目前，卷积网络的整体结构趋向于使用更小的卷积核（比如 1×1 和 3×3 ）以及更深的结构（比如层数大于 50）。此外，由于卷积的操作性越来越灵活（比如不同的步长），汇聚层的作用也变得越来越小，因此目前比较流行的卷积网络中，汇聚层的比例正在逐渐降低，趋向于全卷积网络。

2.3 实验模型 SVM

2.3.1 SVM 原理

SVM（Support Vector Machine）是一种用于二分类和多分类，线性和非线性分类的非概率统计机器学习算法。

下面是 SVM 多分类模型的原理：

（1）One-vs-One：给定一个分类问题，我们选择任意两类之间建立一个分类器。这样我们一共可以得到$C(C-1)/2$个分类器，其中C是所选用的类的个数。当每个分类器都训练好后，对于一个新的数据点，我们将它带入所有的分类器中，计算出每一个分类器的分类结果。我们选择得分最多的来决定测试点所属的类。

（2）One-vs-the-Rest：对于C个类别，我们分别能训练出C个分类器，将1类作为正例，将其余的$C-1$个类别合并起来作为反例，得到C个分类器。同样的，当预测新的数据点的分类时，我们将其带入所有的分类器中，每个分类器输出一个类的概率分数，我们选择最高得分对应的类别做为最终预测结果。

其中，One-vs-the-Rest 基于一种 "K vs. All" 的策略，而在实现时，通常是通过使用如下的损失函数(软间隔二分类)和相应的优化算法求出超平面：

$$\begin{aligned} &\min_{w,b,\{\xi_i\}} \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i, \\ &y_i\left(w^T x_i + b\right) \geq 1-\xi_i, \xi_i \geq 0, i=1,...,n \end{aligned} \tag{3}$$

其中，w是待求的系数向量，b是截距，ξ是误差或松弛变量，C为权衡分类间隔和误分类损失之间的重要性参数，y_i表示样本的真实标签，x_i样本特征向量。

最终，使用 One-vs-the-Rest 方法建立的 SVM 多分类器，可以对各类别之间的差异性进行有效地衡量，从而得到准确、可靠的预测效果。

2.3.2 间隔与支持向量

给定训练样本集$D=\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}, y_i \in \{-1,+1\}$，分类学习最基本的想法就是基于训练集$D$在样本空间中找到一个划分超平面,将不同类别的样本分开。但是往往能将将训练样本分开的划分超平面可能有很多，如图 7 所示，所以如何选择超平面成了一个问题。

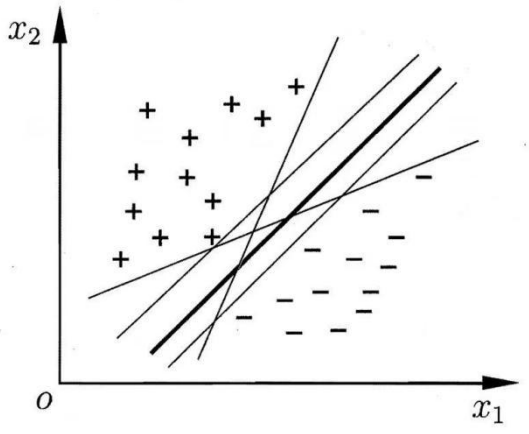


图 7： 多个可以将两类样本分开的超平面

直观上看，应该去找位于两类训练样本“正中间”的划分超平面，即图 7 中最粗的那个，因为该划分超平面对训练样本局部扰动的“容忍”性最好。例如，由于训练集的局限性或噪声的因素，训练集外的样本可能比图 7 中的训练样本更接近两个类的分隔界，这将使许多划分超平面出现错误，而红色的超平面受影响最小。换言之，这个划分超平面所产生的分类结果是最鲁棒的，对未见示例的泛化能力最强。

在样本空间中，划分超平面可通过如下线性方程来描述:

$$w^T x+b=0 \tag{4}$$

其中 $w=(w_1;w_2;...;w_d)$ 为法向量，决定了超平面的方向，b 为位移项，决定了超平面与原点之间的距离。显然，划分超平面可被法向量 w 和位移 b 确定，下面我们将其记为 (w,b)。则样本空间中任意点 x 到超平面 (w,b) 的距离可写为：

$$r=\frac{\left|w^T x+b\right|}{\|w\|} \tag{5}$$

假设超平面 (w,b) 能将训练样本正确分类，即对于 $(x_i,y_i)\in D$，若 $y_i=+1$，则有 $w^T x+b>0$；若 $y_i=-1$，则有 $w^T x+b<0$。令

$$\begin{cases} w^T x_i+b\geq +1, & y_i=+1 \\ w^T x_i+b\leq -1, & y_i=-1 \end{cases} \tag{6}$$

如图 8 所示，距离距离超平面最近的这几个训练样本点使式(6)的等号成立，它们被称为"支持向量" (support vector) ，两个异类支持向量到超平面的距离之和为：

$$\gamma=\frac{2}{\|w\|} \tag{7}$$

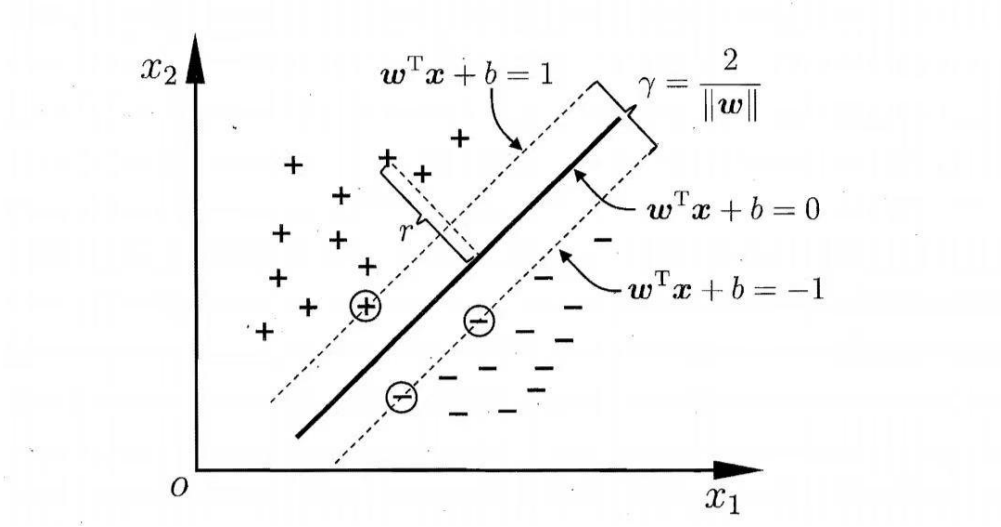


图 8： 支持向量与间隔

它被称为"间隔" (margin)。

欲找到具有“最大间隔” (maximum margin) 的划分超平面，也就是要找到能满足式(6)中约束的参数 w 和 b 使得 γ 最大，即

$$\max_{w,b} \frac{2}{\|w\|} \tag{8}$$

$$s.t.\ y_i(w^T x_i + b) \geq 1,\ i = 1,2,...,m$$

显然，为了最大化间隔，仅需最大化 $\|w\|^{-1}$，这等价于最小化 $\|w\|^2$。于是，式(x)可改写为：

$$\min_{w,b} \frac{1}{2}\|w\|^2 \tag{9}$$

$$s.t.\ y_i(w^T x_i + b) \geq 1,\ i = 1,2,...,m$$

这就是支持向量机(Support Vector Machine，简称 SVM) 的基本型。

2.3.3 核函数

在上面的讨论中，我们都是假设训练样本是线性可分的。即存在一个划分超平面能将训练样本正确分类。然而现实任务中，原始样本空间中也许并不存在一个能正确划分两类样本的超平面。如图 x 所示，对这一点问题，可将样本从原始空间映射到一个更高维的特征空间，使得样本在这个空间内线性可分。

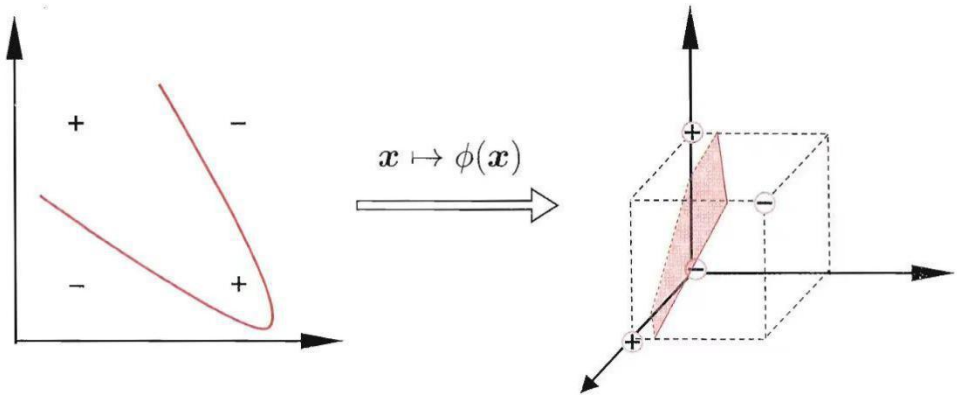


图 9： 异或问题与非线性映射

令 $\phi(x)$ 表示将样本 x 映射后的特征向量，于是，在特征空间中划分超平面所对应的模型可表示为 $f(x) = w^T\phi(x) + b$，然后对偶形式中也有数据向量的乘积， 于是便可以进行替换：$\langle x_i, x_j \rangle \to \langle \phi(x_i), \phi(x_j) \rangle$。

由于特征空间维数很高，甚至可能是无穷维，因此直接计算$\phi(x_i)\cdot\phi(x_j)$通常是 困难的。为了避开这个障碍，可以设想这样一个函数：$K(x_i,x_j)=\phi(x_i)\cdot\phi(x_j)$即$x_i$与$x_j$在特征空间的内积等于他们在原始样本空间中通过函数$K(\cdot)$计算的结果。有了这样的函数，我们就不必直接去计算高维甚至无穷维特征空间中的内积。

显然，如果知道合适映射$\phi(\cdot)$的具体形式，则可写出核函数$K(\cdot,\cdot)$，但在现实任务中我们通常不知道$\phi(\cdot)$是什么形式，什么样的函数能做核函数呢？这里有一个定理：只要一个对称函数所对应的核矩阵是半正定的，它就能作为核函数使用。表 1 列举了几种常见的核函数：

表 1：几种常见的核函数

名称	表达式	参数
线性核	$K(x_i,x_j)=x_i^Tx_j$	
多项式核	$K(x_i,x_j)=(x_i^Tx_j)^d$	$d\geq1$为多项式的次数
高斯核	$K(x_i,x_j)=\exp\left(-\frac{\|x_i-x_j\|^2}{2\sigma^2}\right)$	$\sigma>0$为高斯核的带宽
拉普拉斯核	$K(x_i,x_j)=\exp\left(-\frac{\|x_i-x_j\|}{\sigma}\right)$	$\sigma>0$
Sigmoid 核	$K(x_i,x_j)=\tanh(\beta x_i^Tx_j+\theta)$	tanh 为双曲正切函数，$\beta>0,\theta<0$

2.3.4 软间隔与正则化

在前面的讨论中，我们一直假定训练样本在样本空间或者特征空间中是线性可分的，即存在一个超平面能将不同类的样本完全划分开。然而，现实任务中往往很难确定合适的核函数使得训练样本在特征空间中线性可分。也有可能造成线性不可分的原因可能是存在一些噪声点的影响。

如果完完全全按照原始的约束条件$y_i(w^Tx_i+b)\geq1$，意味着所有数据点距离分割平面的距离都大于 1，可能会使 SVM 发生较大的误差，如图 10 所示。

缓解办法是允许支持向量机在一些样本上出错。为此，要引入“软间隔”的概念。 具体来说，前面介绍的支持向量机形式是要求所有样本均满足约束，即所有样本 都必须划分正确，这成为“硬间隔”，而软间隔则是允许某些样本不满足约束。 为了能让达到此目的，我们可以引入一个松弛变量$\xi_i\geq0$来允许错误的分类发生，也就是允许有间隔小于 1 的数据点出现，即$y_i(w^Tx_i+b)\geq1-\xi_i$。

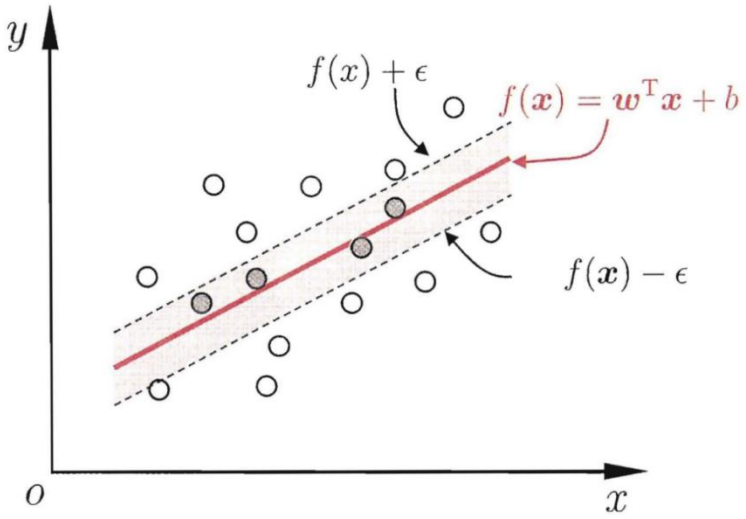


图 10： SVM 产生较大误差

三、 实验步骤和程序流程

3.1 实验步骤整体流程

3.1.1 CNN 模型流程图

为了完成本次实验的 CNN 分类模型的构建，我们需要进行数据预处理（可视化分析，重复类剔除，图片处理），CNN 模型的构建，损失函数的选择，优化器的定义，正则化技术的使用，超参数的初始化以及模型的评价与改进等。具体流程图如图 11 所示。

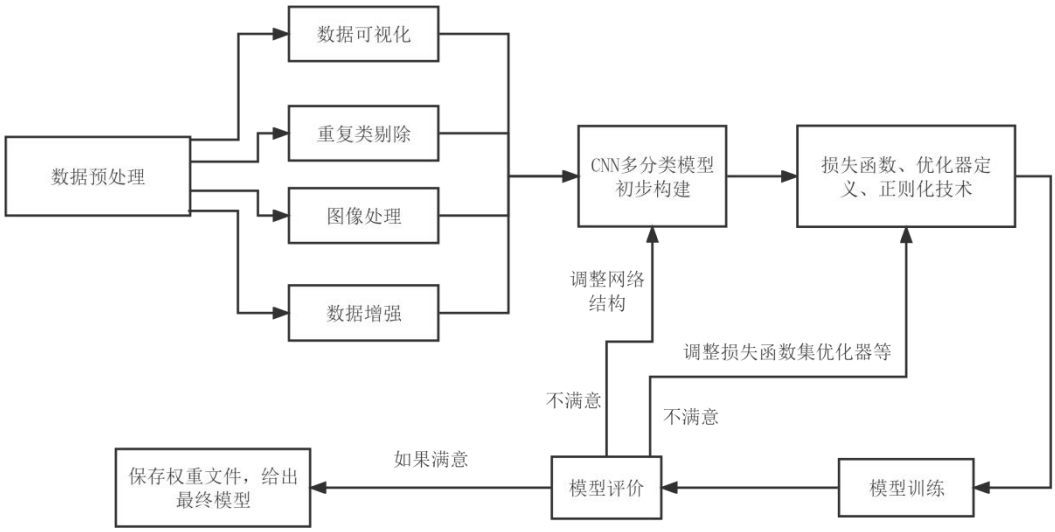


图 11：CNN 模型构建基本流程图

3.1.2 SVM 模型流程图

本实验中 SVM 多分类模型的构建我没有使用助教提供的框架，以下是我建立 SVM 模型的基本思路流程：

（1）从 CNN 模型建立前删除掉重复类数据的 data.mat 文件中导入训练集和测试集的图像数据和标签数据，注意此时应该仅有 198 类手写字符。再将将训练集和测试集的图像数据和标签数据分离，这里的训练集和测试集的图像数据是一个大小为[2970,28,28]的三维数组，标签数据是一个大小为 2970 的数组。

（2）定义 deskew（校正）函数，用于校正输入图像并去除其倾斜。

（3）定义 get_hog()函数，用于对输入的图像提取其对应的 HOG 描述符。

（4）定义 svm_init()函数，用于初始化 SVM 分类器，设置其所需要的核函数以及其他参数。在后续调试中可以换成其他的核函数，在本次实验中，我使用的 opencv 库提供的 SVM 模型，使用到的核函数有 SVM_CHI2, SVM_RBF, SVM_LINEAR, SVM_POLY, SVM_SIGMOID。

（5）定义 svm_train()函数和 svm_predict()函数，分别用于训练 SVM 分类器和对测试数据进行预测分类。

（6）定义 svm_evaluate()函数，用于评估 SVM 分类器的准确性。

（7）将训练集图像数据打乱，并提取 HOG 描述符。使用不同的 C 和 gamma 值训练 SVM 分类器，并计算分类精度。

（8）将结果保存至 Excel 文件中，并将最高分类精度输出。

（9）可视化结果。

具体流程图如图 12 所示：

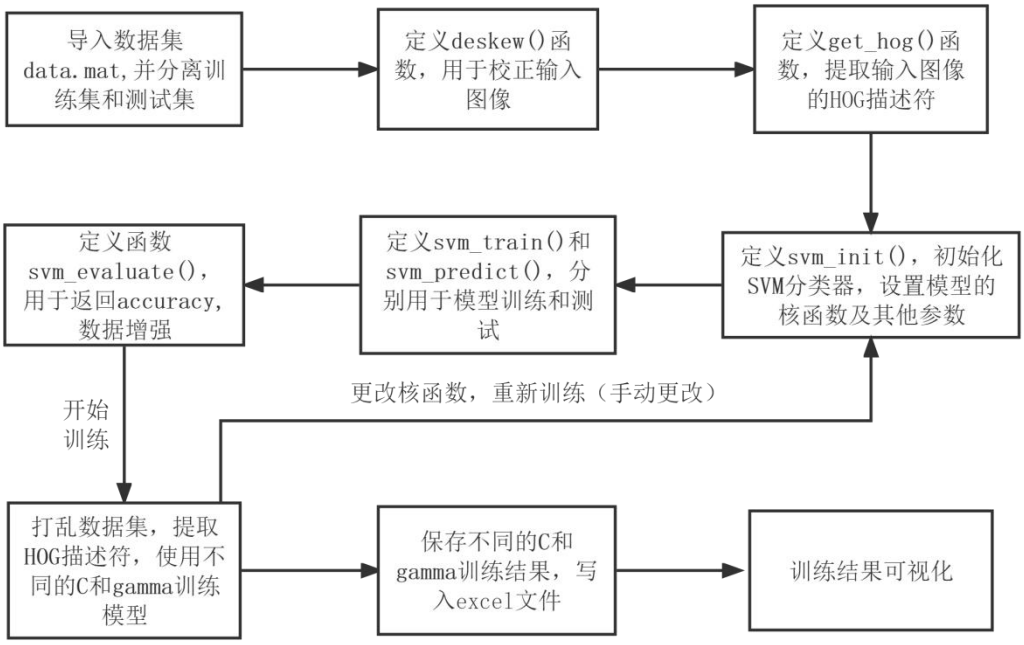


图 12：SVM 模型构建基本流程图

3.2CNN 模型数据预处理

3.2.1 数据可视化处理

给定的数据集中每类有 20 个样本，其中每个样本的大小为 28*28，我使用 python 根据类别将每次划分为 train 和 test 属性分别生成可视化图片（代码见附录 view.py）。值得注意的是，有同学发现，在本数据集中，存在 3 类（120，121，122）出现了字符重复的现象。经过可视化分析，发现相关类实际情况如图 13 所示，结合 Omniglot 原始数据集，发现其均为奥吉布瓦文字符中的同一个字符。

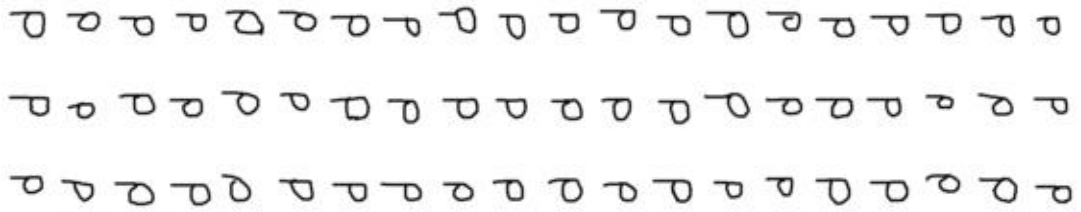


图 13：120、121、122 类手写字符数据（重复）

3.2.2 重复数据剔除：

鉴于以上数据集本身存在问题，索性我将原数据集 NewDataset.mat 文件中的 121 和 122 类数据删除，构建新的数据集 data.mat，故次数据集中只包含 198 类手写字符，与原数据集相同的是，每类字符仍有 20 张手写图片的样本，前 15 张为 train 类型，作为模型的训练集，后 5 张为 test 类型，作为模型的测试集。（注：删除异常数据的代码见附录 dataProc.py）。

3.2.3 数据处理：

在对不同类手写字符图片进行分类识别之前，我们需要先对图片数据做一些预处理工作，在本实验中，我们使用的模型在设计之初仅支持 RGB 类型图像的训练与预测，为了防止灰度图在预测时出错，需要将所有的图像转化为 RGB 类型（用过 utils.py 中的 cvtColor 函数实现）。同时，应该将输入图像调整为指定的大小，以满足模型的对图像尺寸的需求（通过 utils.py 中的 resize_image 函数实现)。

在图像输入神将网络模型前，我们还要对其进行标准化。在 utils.py 中定义了 preprocess_input 函数，该函数用于将输入图像标准化，以便将其输入神经网络进行训练或预测。该函数接受一个图像作为输入，将其所有像素值除以 255，从每个像素中减去 0.5，然后将结果乘以 2。这将得到一个像素值范围为 -1 到 1 的标准化图像。

3.2.4 数据增强：

数据增强技术可以通过对原始图片进行旋转、翻转、缩放、裁剪、平移、加

噪声等变换操作，生成多个不同的图像，从而扩大训练集的规模。这样做有助于缓解数据不足的问题，减少模型过拟合的风险，提高模型对于未知数据的适应能力。此外，数据增强还可以使得模型对于图像中的位置、尺度、光照等因素的变化更加鲁棒，从而提高模型在实际应用中的效果。

为了提高训练集的多样性和数量，从而提高模型的泛化能力和鲁棒性。我对训练集数据采用了一些常见的数据增强操作。在此次实验中，我采用了水平翻转、垂直翻转和随机裁剪这 3 种数据增强手段。分别在 dataloader_cl.py 中定义为 horizontal_flip()，vertical_flip()，random_crop()和 random_rotation()四个函数，并在训练集数据加载时进行了调用。其中 horizontal_flip()函数实现了对输入图片的水平翻转，通过随机生成一个 0 到 1 之间的随机数，如果小于 0.5，则对图片进行水平翻转操作，即将图片左右翻转。vertical_flip()函数实现了对输入图片的垂直翻转，与 horizontal_flip()类似，也是通过生成一个随机数来决定是否进行翻转。random_crop()函数实现了对输入图片的随机裁剪，首先计算出裁剪后图片的大小，然后随机生成裁剪区域的左上角坐标，最后从输入图片中截取对应区域的图像。random_rotation()函数使用 random.uniform 函数生成一个-10 到 10 的随机浮点数作为旋转角度，使用 skimage 库的 rotate 函数对输入的图像进行旋转，参数包括待旋转的图像、旋转的角度、是否重新调整图像尺寸、处理图像边缘的方式。

这些数据增强函数可以通过随机组合，生成更多样的图片数据，从而扩大训练集的规模，提高模型的泛化能力。

3.3CNN 模型构建：

3.3.1 CNN 神经网络模型构建

助教老师所提供的示例代码训练集数据 acc 表现良好，但测试集的 acc 仅有 0.69 左右，这一指标让我怀疑了模型可能存在一定的过拟合现象。但是，通过加深 CNN 网络，发现测试集的 acc 会有较为明显的提升。于是，我采取了一些折中策略，既对网络层数进行加深，又增加了一些正则化操作，例如通过 Dropout 随机删除神经网络中的神经元以解决可能存在的过拟合现象。

下面介绍我更改的网络结构（注：完整代码见附录中 model.py 文件）。

（1）输入层：一个大小为(B, 1, 28, 28)的张量，其中 B 是输入数据的批次大小，1 表示通道数，28x28 表示每个数字图像的大小。

（2）卷积层部分：由 4 个卷积块组成，每个卷积块包括一个卷积层和一个最大池化层。每个卷积层之后都有批次归一化和 LeakyReLU 激活函数。前几个卷积块的输出被展平以供后面的全连接层使用。

（3）全连接层部分：包括 4 个线性层，每个线性层之后跟随批次归一化和

LeakyReLU 激活函数，并在前三层使用了 Dropout 正则化技术。最后一个线性层的输出大小为 198，以匹配预测的类别数。

（4）输出层：一个大小为(B, 198)的张量，198 代表 198 类手写字符，每个元素表示相应类别的预测概率。

该模型的目标是将输入的数字图像通过卷积和池化过程特征提取，然后将这些特征输送到全连接层进行分类，最后输出每种类别的预测概率。图 14 为一个与之类似但仅有两个卷积块和两个全连接层的 CNN 网络结构。

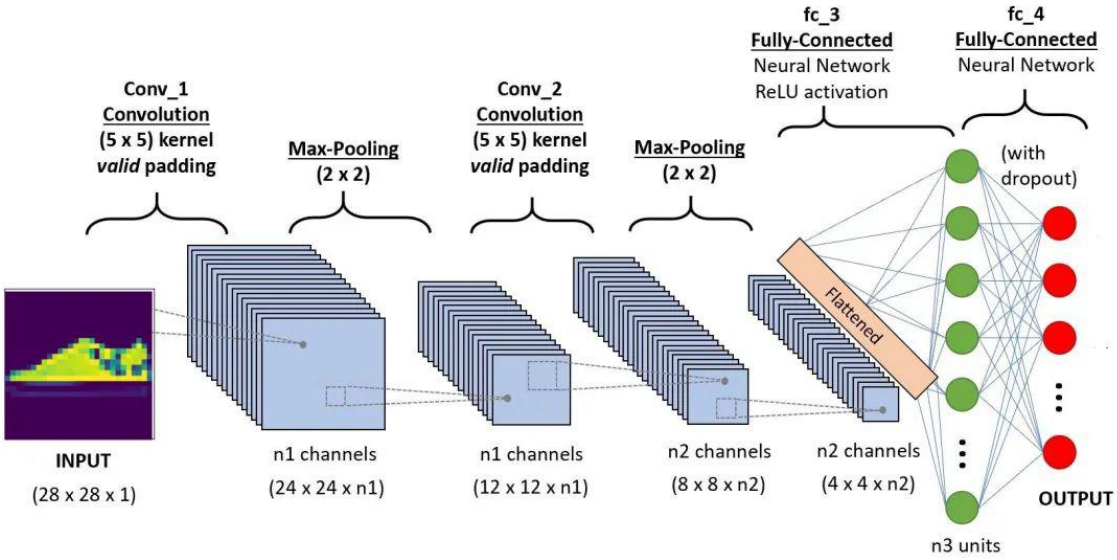


图 14：CNN 模型基本结构

卷积层、池化层、全连接层输入输出参数存在一定的关联关系。具体来说，卷积层输出的大小（即特征图大小）是通过多次卷积和池化所确定的，而全连接层的输出大小则是直接通过参数决定的。假设输入图像大小为 (batch_size, channel, height, width)，卷积层和池化层的参数为 kernel_size、stride、padding、pooling_size 等，全连接层的参数为输入特征向量的大小和输出特征向量的大小，则它们之间可以通过以下关系进行计算：

（1）对于卷积层和池化层：输出特征图的大小公式为：

$$H\_out = (H\_in + 2 * padding - kernel\_size) / stride + 1$$

$$W\_out = (W\_in + 2 * padding - kernel\_size) / stride + 1$$

其中，H_in 和 W_in 分别为输入特征图的高度和宽度。

（2）将卷积层输出的特征图扁平化后，特征向量的大小为：

$$feature\_size = channel\_number * H\_out * W\_out$$

（3）对于全连接层：输入特征向量的大小为 feature_size，输出特征向量的

大小为 output_size。

3.3.2 损失函数、优化器定义、激活函数、正则化技术

①Cross-Entropy 损失函数：

本实验是一个明显的 CNN 图像多分类问题，在模型中，使用了 criterion = nn.CrossEntropyLoss()来定义模型的损失函数。nn.CrossEntropyLoss()函数的作用是对于多分类问题，计算模型预测值与真实标签之间的差距，并将其作为损失函数的值。这个损失函数是一个负对数似然损失，用来衡量模型预测结果的正确性和置信度。CrossEntropyLoss 损失函数有以下下几个明显优点：

（1）适用于多分类问题：CNN 神经网络通常用于图像分类问题，其中图像可以属于多个类别。nn.CrossEntropyLoss() 可以处理多个类别的情况，因此非常适合用于 CNN 神经网络的训练。

（2）处理类别不平衡问题：在实际应用中，不同类别的样本数可能会不平衡，这可能会导致模型在少数类别上表现不佳。nn.CrossEntropyLoss() 可以通过加权损失来处理类别不平衡问题，使得每个类别对训练损失的贡献是平衡的。

（3）与 softmax 激活函数配合使用：nn.CrossEntropyLoss() 常常与 softmax 激活函数一起使用。softmax 函数可以将 CNN 神经网络的输出转换为各个类别的概率，而 nn.CrossEntropyLoss() 可以直接使用这些概率计算损失，简化了训练过程。

（4）数值稳定性好：nn.CrossEntropyLoss()的实现考虑了数值稳定性问题，可以防止数值溢出或下溢，从而保证了训练的稳定性和收敛性。

在此值得注意的是，在 CNN 神经网络模型中，通常会在最后一层全连接层中使用 softmax 函数。这个全连接层的输出神经元的数量应该与数据集的类别数量相同，softmax 函数会将神经网络的输出转换为每个类别的概率。这样，模型输出的每个元素都可以看作是该样本属于对应类别的概率值。但是，在使用 nn.CrossEntropyLoss() 损失函数时，softmax 操作已经在损失函数中进行了，因此在模型定义中不需要显式地使用 softmax。如果同时使用了 nn.CrossEntropyLoss() 损失函数和 softmax 操作，则会对同一输出进行两次 softmax 操作，从而可能会导致数值稳定性问题。

②Adam 优化器：

将 Adam 优化器用于卷积神经网络（CNN）处理图像的多分类问题中，可以带来以下几个好处：

（1）改善模型性能：Adam 优化器可以帮助模型更快地收敛到最优解，并且可以更好地避免训练过程中的过拟合现象。这使得模型在处理图像多分类问题时

具有更好的准确性和泛化能力。

（2）更高的效率：由于 Adam 优化器的收敛速度较快，因此在处理大规模图像多分类问题时，可以更快地完成训练，并且可以更快地调整模型参数以提高准确性。

（3）更好的处理稀疏梯度：CNN 在处理图像时，经常会遇到一些像素之间没有联系的情况，这会导致梯度变得稀疏。Adam 优化器可以更好地处理这种稀疏梯度，从而避免了优化过程中可能出现的问题。

同时，定义好优化器后，可以考虑对学习率进行退火。一种简单的学习率退火策略是 StepLR，它会在训练过程中按照预先设定的步数来调整学习率。可以创建一个学习率调度器：scheduler = StepLR(opt_model, step_size=1000, gamma=0.1)，并在每一个 epoch 结束后使用 scheduler.step()来调整学习率。在上面的代码中，学习率每训练 1000 个 batch 就会乘以 0.1，即以指数级别下降。在实验过程中可以根据实际情况调整相关参数。

③LeakyReLU 激活函数：

为了改善模型的鲁棒性和泛化能力，本次实验在 CNN 网络模型定义时引入了 LeakyReLU 激活函数。例如：

```
nn.Conv2d(1, 32, kernel_size=3, padding=1),
nn.BatchNorm2d(32),
nn.LeakyReLU(inplace=True),  # 添加 LeakyReLU 激活函数
nn.MaxPool2d(kernel_size=2)
```

通常情况下，在深度神经网络中，使用 sigmoid 或 tanh 之类的非线性函数作为激活函数，会出现梯度消失的问题。这是因为这些函数在其输入接近饱和（极端值）时，梯度接近于零，导致反向传播的梯度也接近于零，使得模型训练变得困难。

而 ReLU 函数具有简单、快速的计算方法，且不会出现梯度消失的问题。但是，在 ReLU 中，一旦输入值为负，该神经元输出值就为零，进而使得该神经元的梯度也为零，权重调整过程被抑制，这就是所谓的“死亡神经元”问题。而 LeakyReLU 通过引入一个小的斜率α，可以解决这个问题。下面是 LeakyReLU 的表达式：

$$f(x)=\begin{cases}x, & if\ x\geq 0\\ \alpha x, & if\ x<0\end{cases} \tag{10}$$

图 15 展示了 ReLU 与 LeakyReLU 的区别，其中左边为 ReLU，右边为 LeakyReLU。

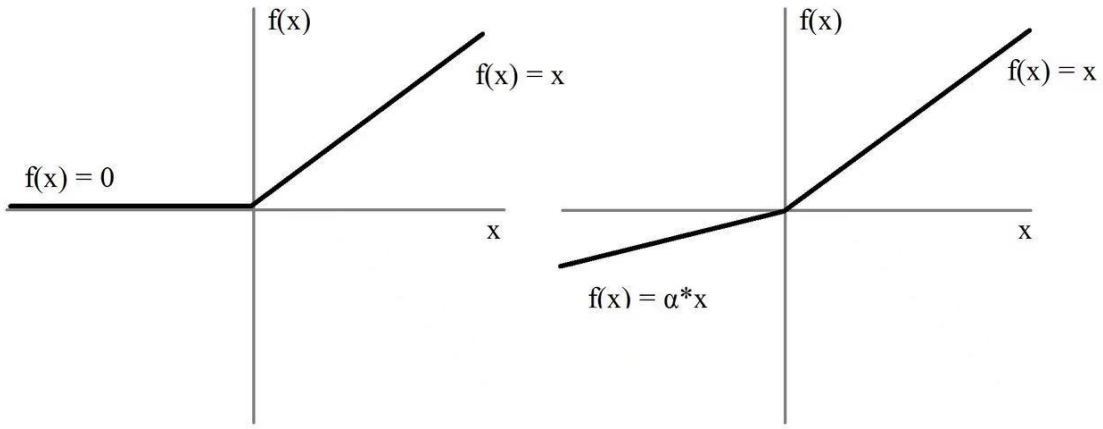


图 15：ReLU 与 LeakyReLU 图像对比图

因此，LeakyReLU 激活函数具有如下优点：

（1）在激活函数的输入 $x<0$ 时，不会出现梯度消失的问题，因此可以避免“死亡神经元”问题。

（2）对于正数 x，计算速度与 ReLU 相同。

（3）在激活函数的输入 $x>0$，相对于 ReLU，梯度仍然存在，可以更好的传播信息。

（4）改善模型的学习效果。LeakyReLU 可以使得模型更容易地学习非线性的特征，这在深度神经网络中特别有用。相对于 ReLU，LeakyReLU 在模型深度较大时，通常表现更好。

综上，LeakyReLU 激活函数可以有效地提高模型的鲁棒性和泛化能力，避免梯度消失和“死亡神经元”问题。

④正则化技术：

为了防止因网络层数过深，模型过于复杂，出现模型在训练数据上表现非常好，但在未曾见过的测试数据上表现很差的过拟合现象。我们需要引入正则化技术。正则化技术通过限制模型权重的大小或变化率，降低模型的自由度，从而降低模型的复杂度，并减少过拟合的风险。通过正则化技术可以使模型更加简洁、健壮、易于理解和调优。在本次实验中，我主要使用了以下 3 种正则化技术：

（1）批标准化（Batch Normalization）技术，这是一种常用的正则化技术，它能够提高神经网络的训练速度和泛化能力。Batch Normalization 是将数据进行标准化处理，即将每个特征的数值经过缩放和平移，从而使得输入数据服从均值为 0，方差为 1 的分布。在 CNN 模型定义的代码中，每一个卷积层（nn.Conv2d）都之后都添加了一个批标准化层（nn.BatchNorm2d），例如：

```
nn.Conv2d(1, 32, kernel_size=3, padding=1),
```

```
nn.BatchNorm2d(32),  # 添加批标准化层
nn.LeakyReLU(inplace=True),
nn.MaxPool2d(kernel_size=2)
```

同时对全连接层的输出也使用了 Batch Normalization 技术，例如：

```
nn.Linear(256, 512),
nn.BatchNorm1d(512) # 添加批标准化层
```

BatchNormalization 可以使得模型学习得更快，使其更稳定。BatchNormalization 实际上是通过对每一层神经元的输出进行归一化处理来达到标准化的效果，并且通过引入学习参数和偏置项来保证模型的表达能力，从而提高模型的鲁棒性和泛化能力。

（2）Dropout，这也是一种常见的正则化技术，其原理图如图 16 所示。通过在训练过程中随机失活一些神经元来减少过拟合的现象，从而提高模型的泛化性能和减少对神经元的依赖性，增加模型鲁棒性。

在 CNN 神经网络分类模型中，Dropout 通常被用在全连接层或者卷积层后的全连接层。它的原理是，在一个迭代的过程中，以一定的概率随机让神经元的输出设置为 0，从而减少过拟合的风险。

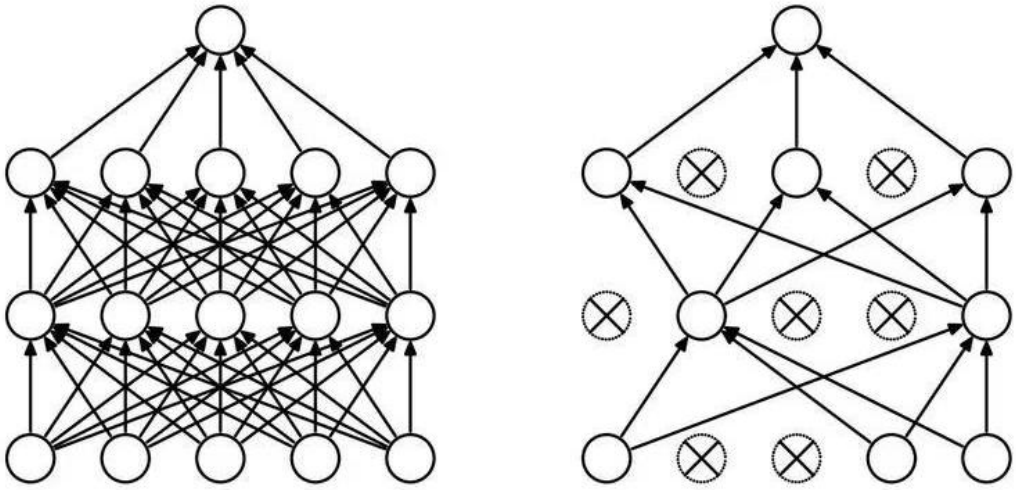


图 16：Dropout 原理图示

具体来说，对于 CNN 中的全连接层或者卷积层后的全连接层，首先需要将特征向量拉平成一维，然后再进行全连接层的处理，最后输出类别概率。在这个过程中，可以引入 Dropout 技术，以概率 p 将这一层或者某些神经元置零，使得部分神经元无法参与训练，降低过拟合的风险。

使用 Dropout 技术的 CNN 模型的训练过程中，每个神经元有一定（$1-p$）的概率参与训练，一定（p）的概率不参与训练。这样，相当于在每次的训练迭代中，重新确定了神经元之间的组合关系，从而达到一定程度的模型泛化效果。和其他正则化技术相比，Dropout 除了能够减少过拟合产生以外，还具备很好的并行性，算法实现简单等优点。

（3）L2 正则化，本实验中通过在 Adam 优化器定义的代码中通过设置 weight_decay 参数来实现 L2 正则化技术。在 Adam 优化器中，weight_decay 参数控制了 L2 正则化的强度，当 weight_decay 的值越大时，正则化项的作用就越明显。实际上，Adam 优化器的正则化项是通过在原始的损失函数上添加 L2 正则项得到的，通过向原始的损失函数中加上正则化项，可以实现在模型训练的过程中限制模型复杂度、缓解过拟合问题的效果。具体代码实现如下：

```
opt_model = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=0.0005)
```

3.3.3 相关超参数的初始化

①初始学习率 lr:

学习率（lr）是在反向传播时用来更新神经网络中参数的步长，它对于 CNN 神经网络多分类模型的训练十分重要。它对模型的训练有以下影响：

训练速度：当学习率较大时，训练速度往往较快，因为在以较大的步幅更新参数时，网络更快地找到或跳过局部最优解。但是如果学习率过大，网络可能无法正确拟合训练数据，并导致性能降低。

精度：学习率有时会影响模型的准确性。如果学习率太小，网络可能需要大量迭代才能收敛，尤其是在大型深度 CNN 模型的情况下。换句话说，学习率太小可能会导致训练较慢。但是，如果学习率太大，则可能难以找到合适的权重和偏差参数，在过多迭代次数后达到无穷大或无穷小的值，从而导致算法发散。

收敛性：如果学习率设置得过小，则可能需要较长时间才能收敛，因为正确的学习率对于快速收敛非常重要。在另一方面，如果学习率过大，则可能会导致优化算法不稳定，可能会偏离极小值并不断反弹，甚至可能导致无法收敛。

在本次实验中，不仅使用了 Adam 优化器在训练过程中动态调整学习率，还在优化器定义之后显式地定义了一个学习率调整策略，使用了等间隔调整学习率（StepLR）的策略，其定义如下：

```
scheduler = StepLR(opt_model, step_size=1000, gamma=0.1)
```

并在后续的迭代训练中，在每个 epoch 结束后使用 scheduler.step()语句进行调用实现。但经过后续调试，此策略不佳。后面也尝试了多步长衰减 multiStepLR 方法。

综上所述，学习率对于 CNN 神经网络多分类模型的影响十分重要。我们需要根据模型的复杂性、数据集大小以及需求的计算资源等情况动态调整，从而找到合适的学习率，既可以保证模型快速收敛，又可以保证模型的泛化能力和准确性。

②batch_size：

CNN 神经网络模型的 batch_size 指的是每次迭代（epoch）在训练中使用的样本数。在神经网络的训练过程中，将数据划分为一个个 batch，每个 batch 包含了一部分训练样本，通过对这一部分训练样本进行反向传播，进行参数的更新。batch_size 的取值大小对模型训练有比较重要的影响：

(1)训练效率： batch_size 的数值越大，每次迭代计算的样本数也就越多，训练速度也就越快。但同时，也会导致可用内存变得更大，而且需要更大的网络才能适应更大的 batch_size，因此 batch_size 也不宜设置得太大。

(2)模型的泛化能力：在训练过程中，较小的 batch_size 有助于避免过拟合现象。因为较小的 batch_size 能更充分地反映训练数据的短期统计特征，这样能使得神经网络更多地“看到”真实数据的一些细节特征，从而提高泛化能力。

(3)确保收敛性：batch_size 也影响到是否能正确收敛和训练时间。如果 batch_size 比较小，那么每一步更新的方向就可能过于随机化，容易造成收敛性较差的问题。而如果 batch_size 比较大，那么每一步的参数更新就较为合理，从而避免了收敛性较差的问题。

(4)对梯度更新的影响：batch_size 的不同也对梯度更新起到了很大的不同影响。batch_size 越大，那么更新梯度使用的信息也就越具有代表性，让我们能够更准确地推知其他样本的梯度情况，但同时，也可能远离了梯度最优解的位置，无法使得模型达到局部最优解。

因此，CNN 神经网络模型选择合适的 batch_size 是很重要的，需要均衡效率和模型的泛化能力，同时要确保模型能够有效地收敛训练。这就需要我们在模型训练的过程中不断调整，直至寻找到一个合适的 batch_size。

3.4 CNN 模型训练

这个过程中，我们只需要运行整个代码，观察训练集和测试集的 acc 以及损失函数 loss。需要注意的是这 3 项指标的数据大小和变化趋势，并据此反映模型的拟合效果。同时，我们需要根据指标反映出的效果动态地调整上面所提到的与模型有关的结构，优化器，超参数等。

3.5 SVM 模型数据预处理

SVM 的数据集可视化与重复类删除操作与 CNN 的完全一致，此处不再赘述。

下面介绍 SVM 的其他数据预处理操作。

3.5.1 数据划分

先对 data.mat（去除重复类的数据）的数据进行划分，按照数据集的特征，每行包含代表一类手写字符的 20 张大小为 28*28 的图片数据，其中前 15 张作为训练集，后 5 张作为测试集。我们需要分别对训练集和测试集的数据进行图像和标签的分离，具体实现见代码中 split_train()及 split_test()两个函数。

3.5.2 图像去斜

对于 Omniglot 数据集的手写字符图片数据，考虑到可能存在多种因素导致图像倾斜，例如手写习惯、拍照角度等等，而倾斜的图像可能会严重影响模型的分类效果。因此，在建立 Omniglot 数据集的多分类模型中，可以使用这段代码对数据集中的手写字符图片进行去斜处理，从而减小倾斜对分类结果的影响，提高模型的分类准确性。为此，定义了一个 deskew()函数，其代码如下：

```
def deskew(img):
    m = cv2.moments(img)    # 计算输入图像的矩
    if abs(m['mu02']) < 1e-2:   # 判断是否需要进行校正
        return img.copy()
    skew = m['mu11'] / m['mu02']    # 计算图像的倾斜角
    M = np.float32([[1, skew, -0.5 * SIZE_IMAGE * skew], [0, 1, 0]])  # 创建变换
    img = cv2.warpAffine(img, M, (SIZE_IMAGE, SIZE_IMAGE), flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
    return img
```

其原理是输入一张需要去斜处理的灰度图像，利用图像的矩计算出图像的倾斜角，然后创建一个变换矩阵，将图像进行校正，最后返回处理后的图像。根据输入图像的倾斜情况，可以减小或消除由于图像倾斜而带来的误差，提高图像识别、测量等应用的准确性。

3.5.3 数据增强

在 CNN 模型中我们也定义了一些数据增强方法，但是可能由于网络层数较多，其对模型的准确性并没有太大的影响。在后续 SVM 模型的建立中中，同样尝试了一些数据增强操作，发现其对于使用某些核函数的 SVM 模型的准确率提升具有显著效果。下面介绍对 SVM 模型的数据增强操作：

①旋转增强：

```
def rotate(image, angle):
    # 对图像进行旋转增强
```

```
    rows, cols = image.shape
    # 构造旋转矩阵
    M = cv2.getRotationMatrix2D((cols/2, rows/2), angle, 1)
    # 进行仿射变换
    dst = cv2.warpAffine(image, M, (cols, rows))
    return dst
```

②平移增强：

```
def shift(image, x, y):
    # 对图像进行平移增强
    rows, cols = image.shape
    M = np.float32([[1, 0, x], [0, 1, y]])
    dst = cv2.warpAffine(image, M, (cols, rows))
    return dst
```

③缩放增强

```
def zoom(image, scale):
    '''
    对图像进行缩放增强
    '''
    rows, cols = image.shape
    M = cv2.getRotationMatrix2D((cols/2, rows/2), 0, scale)
    dst = cv2.warpAffine(image, M, (cols, rows))
    return dst
```

④增强函数

```
def enhance(image):
    # 对图像进行一系列增强操作，生成新的增强图像
    images = []
    angles = [-10, -5, 5, 10]        # 旋转角度列表
    shifts = [(5, 5), (-5, -5), (5, -5), (-5, 5)]   # 平移距离列表
    scales = [0.9, 1.1]              # 缩放比例列表
    # 旋转增强
    for angle in angles:
        img = rotate(image, angle)
        images.append(img)
    # 平移增强
    for shift_x, shift_y in shifts:
        img = shift(image, shift_x, shift_y)
```

```
        images.append(img)
    # 缩放增强
    for scale in scales:
        img = zoom(image, scale)
        images.append(img)
    return images
```

完成上面一系列的定义之后，在训练模型前，可以先使用 enhance()函数对训练集图像进行增强，生成更多的训练数据，增加模型鲁棒性和泛化能力。

3.6 SVM 模型构建

3.6.1 SVM 模型初始化

在初始化模型时，我使用了 python-opencv 提供的 SVM 框架，定义模型的函数为 svm_init()，函数实现了对 SVM 模型的初始化。其中，创建了一个初始的 SVM 模型；选择支持向量机核函数；设置核函数的系数 gamma 值，一般为样本空间维数的倒数；设置 SVM 模型参数中的惩罚系数 C，C 值越大，则分类精度越高，但是模型更容易过度拟合；设置 SVM 的类型为 C-SVC，即使用惩罚项的分类支持向量机；设置终止准则，当达到最大迭代次数，或者目标函数的变化小于一个指定的阈值，就可以认为已找到了最优解，即达到模型的准确性。最后，该函数返回初始化好的 SVM 模型。

在后续调试中，可以尝试更改 SVM 的核函数来观察结果，以此来选择一个最优的核函数。核函数的定义方法为：

```
model.setKernel(cv2.ml.SVM_CHI2)
```

以上示例中使用的是卡方核 CHI2，实验过程中，我也尝试了其它核函数，如径向基函数核 RBF，线性核函数 LINEAR，多项式核函数 ROLY，S 型核函数 SIGMOID。

3.6.2 高级描述符 HOG 的定义及使用

HOG 是一种用于图像识别中的特征描述符，主要用于提取图像中的线条、边缘、纹理等纹理特征，以用于后续分类任务。使用 HOG 描述符可以通过计算图像的梯度方向直方图，提取出图像中边缘、角点、纹理等重要信息。这些特征可以用于分类、定位、检测等各种图像处理和分析应用。在机器学习任务中，可以将提取出的 HOG 特征作为输入，使用支持向量机等分类器完成分类任务。下面是 HOG 的定义:

```
def get_hog():
    hog = cv2.HOGDescriptor((SIZE_IMAGE, SIZE_IMAGE), (8, 8), (4, 4), (8,
8), 9, 1, -1, 0, 0.2, 1, 64, True)
```

```
# 初始化 HOG 描述符
print("hog descriptor size: {}".format(hog.getDescriptorSize()))
return hog
```

定义完成后，训练集图像在训练前需要使用 HOG 描述符，对数据集中的图像进行特征提取，提取后的特征可用于分类器的训练与测试。

3.6.3 改变参数 C 和 gamma 并输出结果

对于每个 SVM 模型，我们需要设置其参数 C 和 gamma。其中，C 是惩罚系数，控制着分类间隔的大小和是否允许出现异常点。C 值越大，参数向量 w 的影响力就越大，这意味着大的 C 值会导致更严格的决策边界，从而可能导致过拟合。反之，C 值越小，就会导致更宽松的决策边界，从而可能导致欠拟合。而 gamma 表示核函数的带宽，它决定了数据映射到高维空间后的点之间的相似度。gamma 值越大，支持向量越少，决策边界就越窄，可能导致过拟合。反之，gamma 值越小，支持向量越多，决策边界就越宽松，可能导致欠拟合。

实际模型中，合适的 C 和 gamma 对模型准确率的影响非常大，因此需要通过不断调参确定 C 和 gamma 的最优值，以使得模型既能够得到较高的分类准确率，同时又不出现过拟合或欠拟合的情况。

在实验中，我设置了多组 C 和 gamma 的组合，并将不同输出准确率保存到指定的 Excel 文件中，同时可视化结果，便于后续分析。

3.7 SVM 模型训练

此模型不需要迭代训练，但我在实验中会手动切换不同的核函数并观察它们对模型准确率的影响。值得注意的是，由于添加了很多数据增强操作，在训练过程中这些操作会耗费绝大多数时间，同时由于我设置了 10 个 C 的值，9 个 gamma 的值，这些都会导致模型训练很慢。不同核函数的训练速度也会有很大不同。

另外值得注意的是，每次运行代码训练的结果会稍有差异。但总体趋势是比较平稳。

四、 实验结果

4.1 CNN 模型结果

在仅改进网络结构和修改一些超参数的初始化值（如学习率 lr 和批处理大小 batch_size），同时添加了 dropout 正则化技术后，经过多轮迭代训练并保留模型权重文件，训练集数据的 acc 接近 1，测试集 acc 在 0.89 左右，对于有的迭代轮次，测试集的 acc 可达 0.9。刚开始 300 轮次迭代训练的模型指标如图 17 所示：

从图 x 可以看出，前 300 轮训练过程中，train_acc 和 test_acc 的值经历快速增长后增速下降，300 轮时还有缓慢上升趋势。

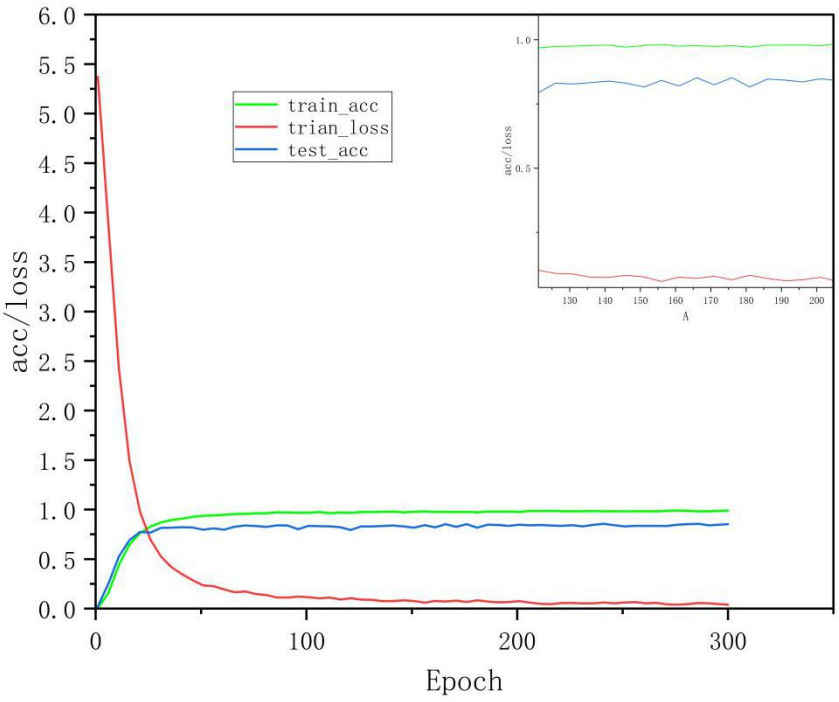


图 17：前 300 轮迭代得到的预训练模型指标

然后我将上述预训练模型的权重文件导入，反复训练多次得到了一个稳定的模型，相关指标如图 18 所示：

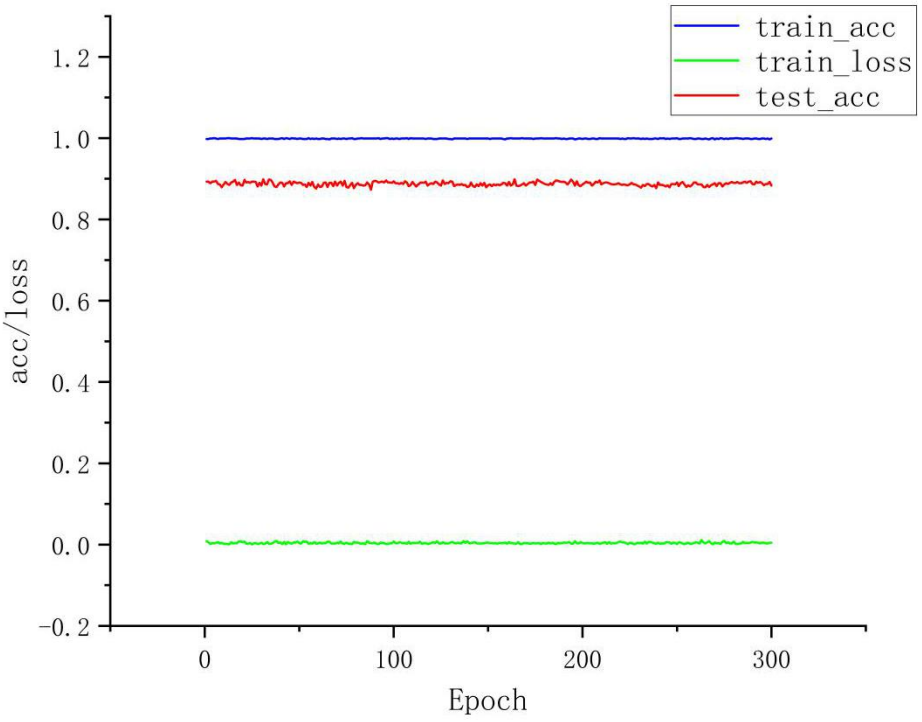


图 18：经过多轮迭代稳定的模型指标

但是，得到上述结果需要迭代较多次，300 轮 epoch 通常达不到上面那个稳定的结果，反映出我修改的模型收敛速度较慢。

下面展示一些多轮迭代训练好的模型运行过程中的进度条，train_acc，train_loss，test_acc 的运行截图，如图 19 所示：

图 19：稳定的模型训练过程部分截图

上述模型在测试集中表现过于良好，但在测试集中表现虽然不错但是较难提升，因此我怀疑模型存在一定的过拟合现象，同时可迁移性较差。于是，我在原模型的基础上增加了数据增强操作，同时使用了更多的正则化技术来预防可能存在的过拟合现象，同时增加了一些学习率衰退策略。更改后的模型各项指标变化如图 20 所示，

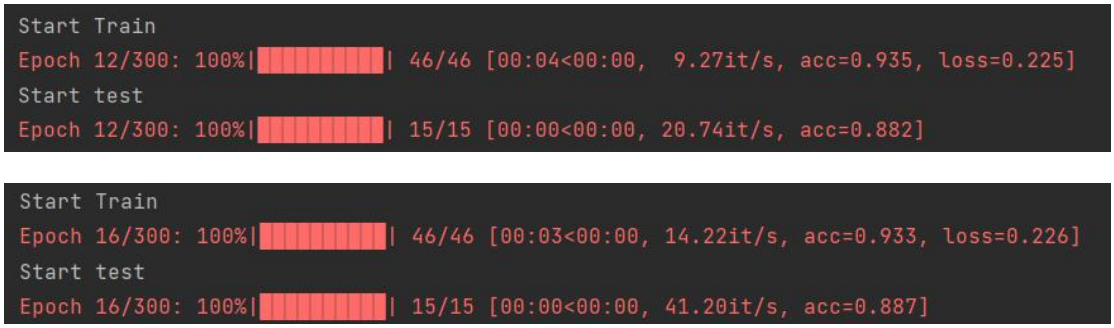


图 20：增加多正则化技术的模型训练过程部分截图

根据图 20，可见此时模型个过拟合现象有了较大的改善。模型在训练集的准确率仅有 0.935 左右时，测试集的准确率已经达到 0.885 左右。训练稳定后，准确率可达 0.9 以上。

4.2 SVM 模型结果

在调试时，由于在 CNN 模型训练过程中，使用数据增强的效果不是很好，

开始时只使用了图像去斜操作，切换不同的 SVM 和函数得到的最优准确率只能达到 0.71，图 21 展示了使用了效果最好的 RBF 核函数的结果：

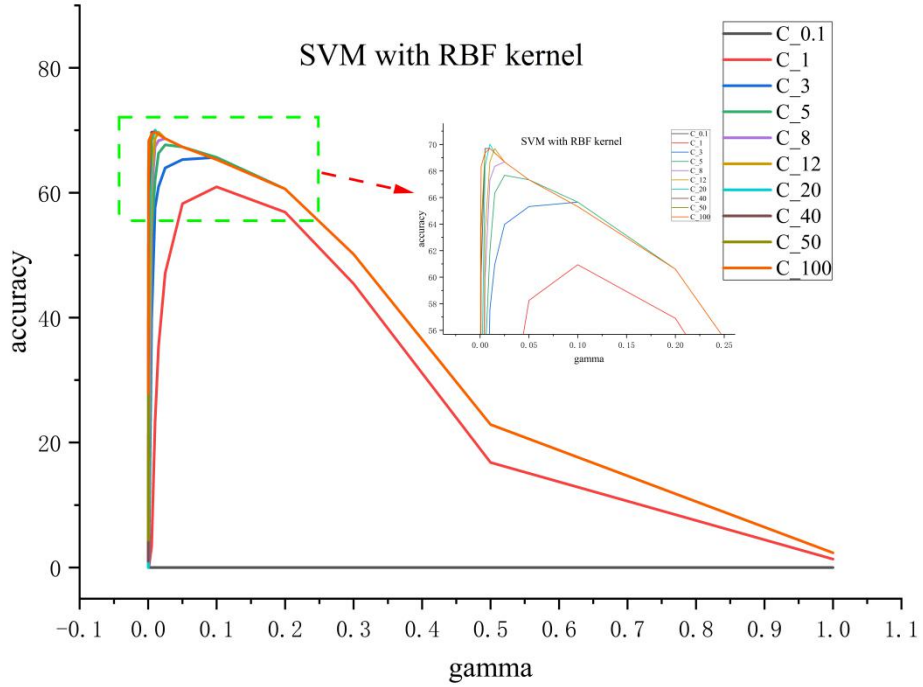


图 21：使用 RBF 核函数的 SVM（不含数据增强）

后面尝试使用了数据增强技术，注意此时应停止使用图像去斜技术，因为二者在实现原理上操作是相矛盾的。下面展示一些结果。

使用 RBF 核函数（含数据增强）：

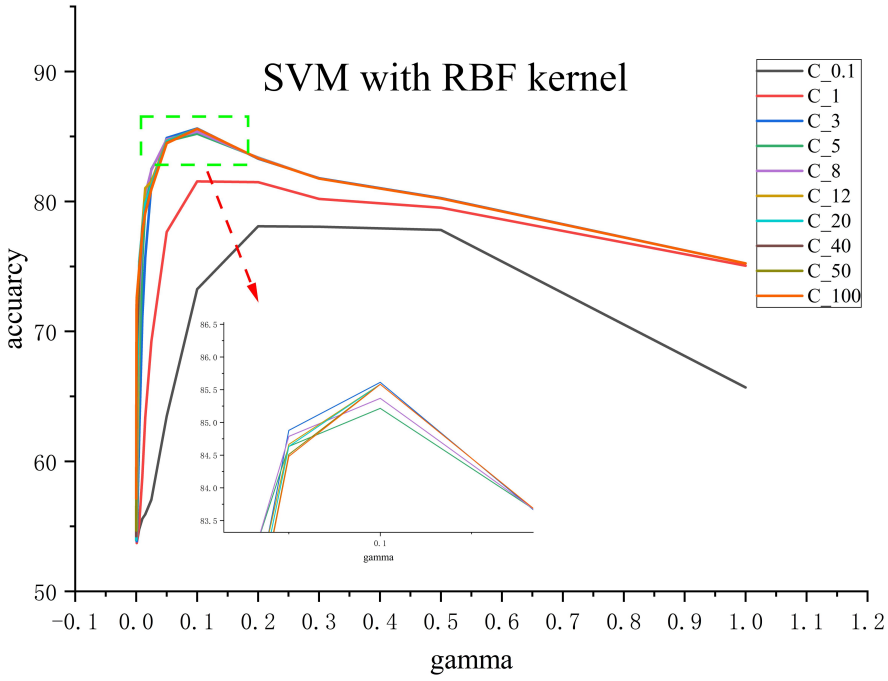


图 22：使用 RBF 核函数的 SVM（含数据增强）

不难看出，对比图 21，图 22 展示的使用 RBF 核函数的 SVM 模型的准确率有了明显提升。准确率最高可达 85.61371289，此时 C 取 3，gamma 取 0.1。

使用 CHI2 核函数（含数据增强）：

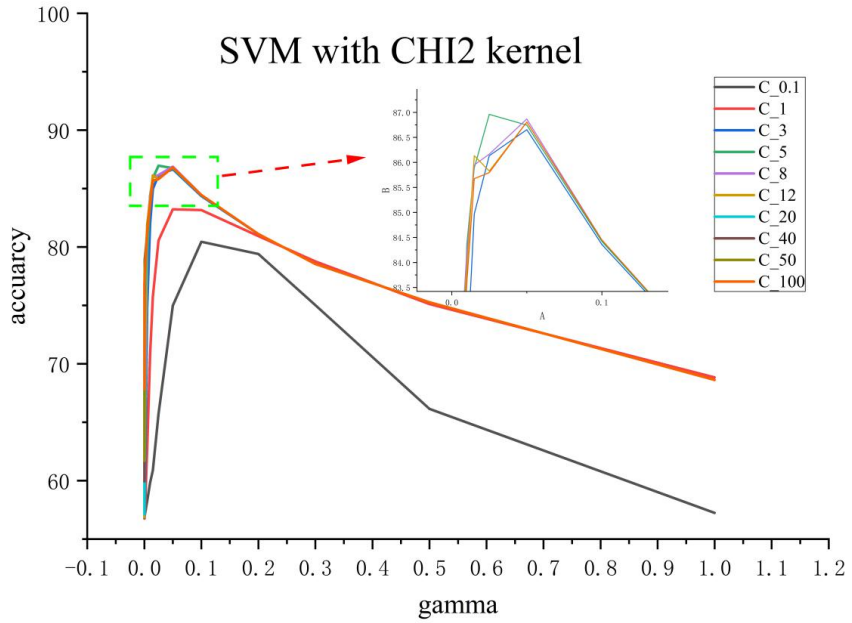


图 23：使用 CHI2 核函数的 SVM（含数据增强）

使用 LINEAR 核函数（含数据增强）：

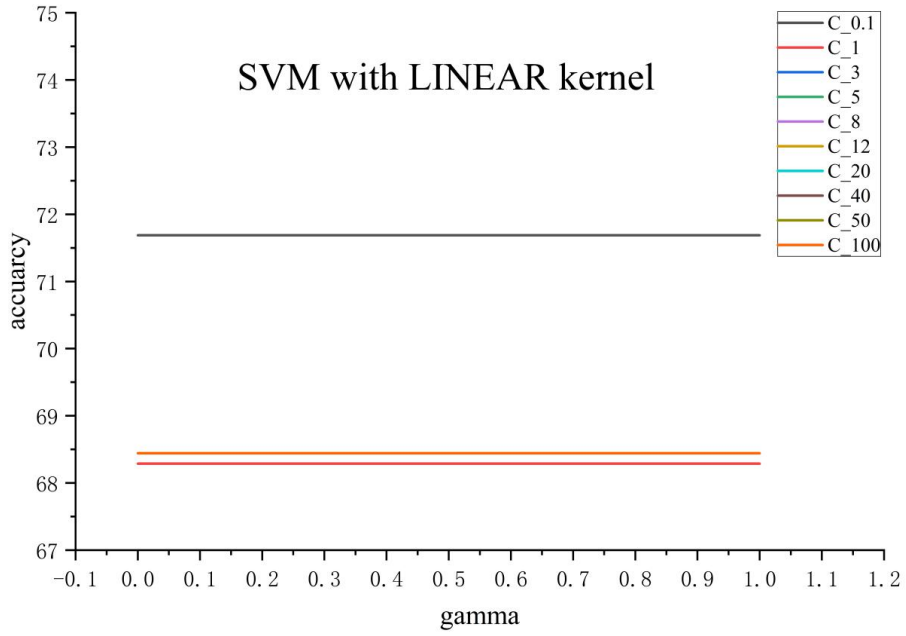


图 24：使用 LINEAR 核函数的 SVM（含数据增强）

使用 POLY 核函数（含数据增强）：

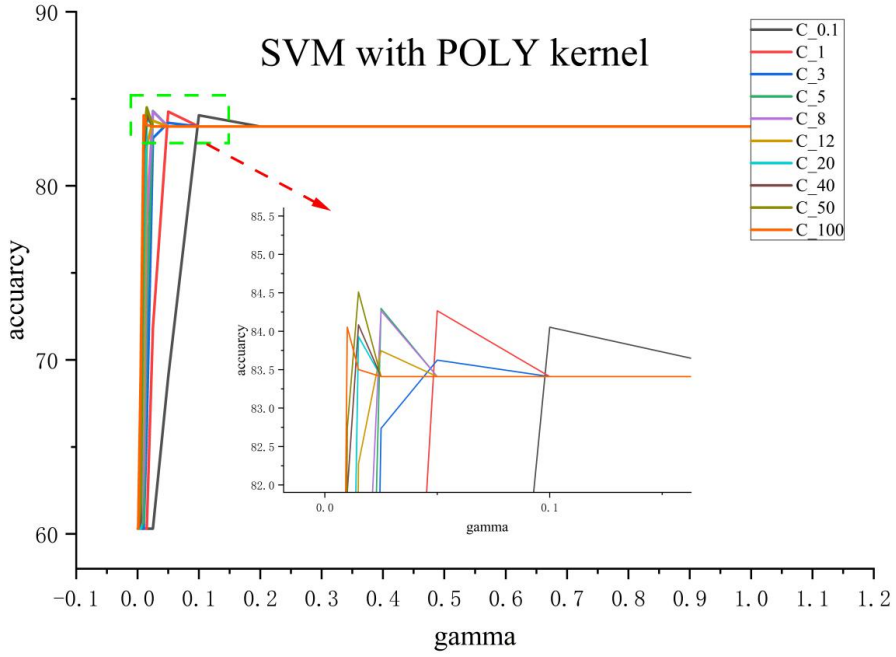


图 25：使用 POLY 核函数的 SVM（含数据增强）

使用 SIGMOID 核函数（含数据增强）：

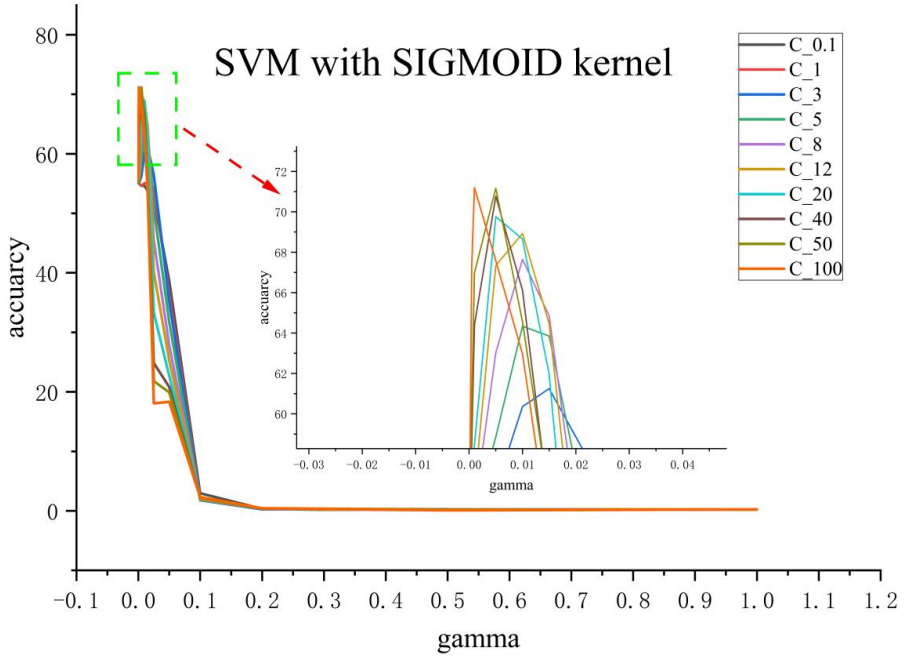


图 26：使用 SIGMOID 核函数的 SVM（含数据增强）

根据图 22-26，使用不同核函数的最优参数 C 和 gamma 的组合如下：

（1）RBF 核：C=3，gamma=0.1，acc=85.61371289。

（2）CHI2 核：C=8，gamma=0.05，acc=86.86868687。

（3）LINEAR 核：C 取 0.1 即可，acc=71.6865626。

（4）POLY 核：C=5，gamma=0.025，acc=84.29752066。

（5）SIGMOID 核：C=12，gamma=0.01，acc=68.93174166。

根据准确率来考虑，在本次实验中，建议使用 RBF 核函数和 CHI2 核函数。但是，CHI2 核函数运算太慢，综合考虑，建议使用 RBF 核函数。

五、 评价分析

5.1 关于 CNN 模型：

通过本次实验，我对 CNN 神经网络多分类模型的构建与调整有了初步的了解，学习到了不少的知识。

针对本次实验的内容，我在将模型训练到在测试集上能达到 0.89 的准确率后，通过查阅资料尝试模型调整时，我也学习了一些损失函数的定义，优化器的定义以及正则化技术，同时也了解了一些数据增强技术。但是，在实际优化过程中，不知是我的技术问题还是数据集本身特征的影响，上面的一些手段对准确率提升的作用并不是很大，甚至会引起收敛速度变慢，分类准确率不增反减等问题，这令我十分困惑，需要进一步学习。

由于本人之前没有做过图片数据集的分类模型，故本次实验对我来说难度还是相对较大，尤其是对于图片数据的处理方法并不是很了解，在这个方面消耗了大量的时间。同时，对于 Pytorch 框架下的 CNN 网络的结构也是一无所知，对于模型的参数也是从头学起。同时，需要进一步了解适用于 CNN 网络的优化器和损失函数的选择，以及一些相关参数比如学习率 lr 和 batch_size 等对收敛速度、模型准确率的影响，综合这些因素一点点尝试以提升模型的综合性能。

5.2 关于 SVM 模型：

此模型的训练过程相对简单，比较容易的点是 opencv 库提供了较为成熟的 SVM 模型，方便调用。难点是数据增强中对于图片数据的处理工作，不仅需要考虑图片的类型，还要考虑图片的大小，以便图片数据能够顺利的代入 SVM 模型进行训练。

值得注意的一个问题是，代码的计算性能有待提升，未加入数据增强操作前，SVM 模型的准确率欠佳，但是计算速度挺快。但是，由于增加了数据增强操作，模型的计算量大大提升，计算时间明显延长。使用 CHI2 核函数时，对于我的代码，SVM 模型训练需要十几个小时。但是，经过观察，这个代码在运行时 CPU

的占用率仅有 13%左右，之前的 CNN 模型训练是 CPU 占有率可达 90%多。显然，这与算法存在一定的关系，这是一个值得优化的点。

5.3 关于实验：

实验过程中比较痛苦的是，在 CNN 模型的训练时，由于我的 PC 机是 AMD 集显，该版本不支持通过 Pytorch 使用 A 卡的 GPU 加速，故无法把损失函数的计算搬到 GPU 上实现，计算速度过于感人，300 轮迭代训练下来，需要花费几十分钟，网络层数较深的话，学习率较小的情况下，速度更慢，有点令人恼火。调个参数看到效果需要花费太多时间！

通过本次实验，我对 CNN 核 SVM 有了一定的了解，一点一点把准确率提高的过程也让我十分有成就感，收获颇丰。

注：①实验报告中卷积神经网络原理示意图来自复旦大学邱锡鹏教授所著的《神经网络与深度学习》，资源下载地址为：https://github.com/nndl/nndl.github.io/blob/master/nndl-book.pdf

②实验报告中支持向量机原理示意图来自周志华所著的《机器学习》。

③实验报告中所有关于代码内具体函数的说明在附录中均加有注释。

附 1:参考文献

[1] 高晗. 基于 SVM 的图像分类算法[D].吉林大学,2019.

[2] 张来君.基于 SVM 的手写数字识别[J].电子技术与软件工程,2021,No.217(23):166-167.

[3] 陈虹州.基于 SVM 的手写数字识别[J].电子制作,2019,No.382(18):49-51+53.DOI:10.16589/j.cnki.cn11-3571/tn.2019.18.019.

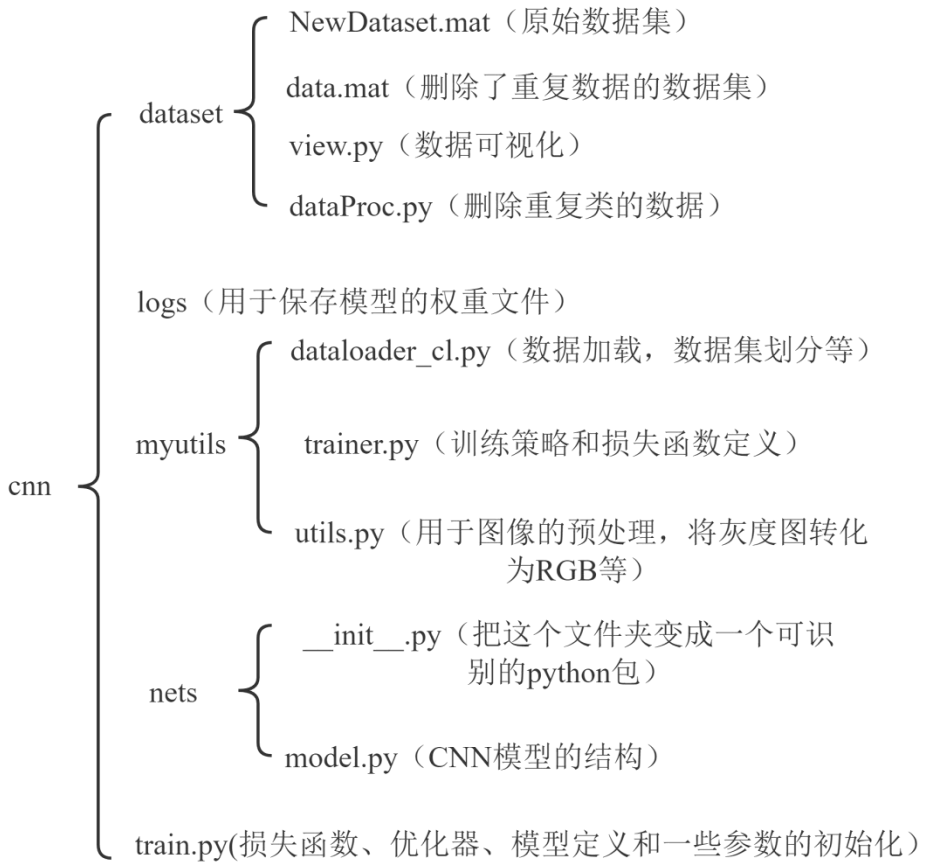
[4] 宗春梅,张月琴,石丁.PyTorch 下基于 CNN 的手写数字识别及应用研究[J].计算机与数字工程,2021,49(06):1107-1112.

[5] 黄一天,陈芝彤.Pytorch 框架下基于卷积神经网络实现手写数字识别[J].电子技术与软件工程,2018,No.141(19):147.

[6] 郝辉. 基于 CNN 的印刷体维吾尔文字符识别[D].新疆大学,2018.

附 2：代码

附 2.1CNN 模型整个项目结构：



view.py（可视化训练集与测试集）：

```
import os
import scipy.io
import numpy as np
import matplotlib.pyplot as plt

# 加载.mat 文件
data = scipy.io.loadmat('NewDataset.mat')

# 获取 train 和 test 数据
train_images = np.zeros((200, 15, 28, 28))
test_images = np.zeros((200, 5, 28, 28))
```

```
for i in range(200):
    train_images[i] = data['train'][i][:15].reshape(15, 28, 28)
    test_images[i] = data['test'][i][:5].reshape(5, 28, 28)

# 创建 train 和 test 文件夹
if not os.path.exists('train'):
    os.makedirs('train')
if not os.path.exists('test'):
    os.makedirs('test')

# 保存 train 图像
for i in range(200):
    combined_image = np.concatenate(train_images[i], axis=1)
    filename = f'train/{i:03d}_train.png' # 序号命名
    plt.imsave(filename, combined_image, cmap='gray')

# 保存 test 图像
for i in range(200):
    combined_image = np.concatenate(test_images[i], axis=1)
    filename = f'test/{i:03d}_test.png' # 序号命名
    plt.imsave(filename, combined_image, cmap='gray')
```

dataProc.py（删除原数据集中 121，122 的重复类）：

```
import scipy.io as sio
import numpy as np

# 加载原始数据文件
raw_data = sio.loadmat('NewDataset.mat')

# 将原始数据中的所有行复制到新的变量中
new_train_data = np.copy(raw_data['train'])
new_test_data = np.copy(raw_data['test'])

# 删除第 121 和 122 行
new_train_data = np.delete(new_train_data, [120, 121], axis=0)
```

```
new_test_data = np.delete(new_test_data, [120, 121], axis=0)

# 保存处理后的数据到新的 MATLAB 格式文件
sio.savemat('data.mat', {'train': new_train_data, 'test': new_test_data})
```

dataloader_cl.py（数据加载，数据集划分，数据增强）：

```
import numpy as np
import torch
from PIL import Image
from torch.utils.data.dataset import Dataset
import scipy.io as scio
from torchvision import transforms
import random
from myutils.utils import preprocess_input
from scipy.ndimage import rotate

def horizontal_flip(img):
    if random.random() < 0.5:
        img = np.fliplr(img)
    return img

def vertical_flip(img):
    if random.random() < 0.5:
        img = np.flipud(img)
    return img

def random_crop(img):
    h, w = img.shape[:2]
    new_h, new_w = int(h * 0.9), int(w * 0.9)
    top = np.random.randint(0, h - new_h)
    left = np.random.randint(0, w - new_w)
    img = img[top: top + new_h, left: left + new_w]
    return img

def random_rotation(img):
```

```
        angle = random.uniform(-10, 10)
        img = rotate(img, angle, reshape=False, mode='nearest')
        return img

class Dataset(Dataset):
        def __init__(self, path, input_shape, epoch_length, is_train):
                super(Dataset, self).__init__()
                self.path = path
                self.input_shape = input_shape
                self.epoch_length = epoch_length

                self.epoch_now = -1
                self.data = scio.loadmat(path)
                self.is_train = is_train

                if is_train:
                        self.length = np.shape(self.data['train'])[0] * 15
                        self.data = self.data['train']
                else:
                        self.length = np.shape(self.data['test'])[0] * 5
                        self.data = self.data['test']

        def __len__(self):
                return self.length

        def __getitem__(self, index):
                if self.is_train:
                        index = index % self.length
                        label = index // 15
                        index = index % 15
                else:
                        index = index % self.length
                        label = index // 5
                        index = index % 5
```

```
            # print(label, index)
            image = self.data[label][index]
            image = np.expand_dims(image, axis=2)

            if self.is_train:
                image = horizontal_flip(image)
                image = vertical_flip(image)
                image = random_crop(image)
                # image = color_jitter(image)
                image = random_rotation(image)
            image = np.transpose(preprocess_input(np.array(image, dtype=np.float32)),
(2, 0, 1))

            return image, label

# DataLoader 中 collate_fn 使用
def dataset_collate(batch):
    images    = []
    Label = []
    for img, label in batch:
        images.append(img)
        Label.append(label)

    images    = torch.from_numpy(np.array(images)).type(torch.FloatTensor)
    Label     = torch.from_numpy(np.array(Label)).type(torch.LongTensor)
    return images, Label
```

trainer.py（训练策略及损失函数定义）：

```
import os
import cv2
import kornia
import numpy
from torch import Tensor
import torch.nn as nn
import torch
```

```
from tqdm import tqdm

def fit_one_epoch(model_train, model, opt_model, epoch, epoch_step,
epoch_step_val, train_gen, val_gen, Epoch,
                          cuda, save_period, save_dir):
    loss = 0
    train_set = set()
    print('Start Train')
    criterion = nn.CrossEntropyLoss()
    if cuda:
        criterion = criterion.cuda()
    pbar = tqdm(total=epoch_step, desc=f'Epoch {epoch + 1}/{Epoch}', postfix=dict,
mininterval=0.3)
    acc = 0
    for iteration, batch in enumerate(train_gen):
        if iteration >= epoch_step:
            break
        images, label = batch[0], batch[1]    # image (B,C,H,W)      label (B)
        with torch.no_grad():
            if cuda:
                images = images.cuda()
                label = label.cuda()

        model_train.train()

        prob_tensor = model_train(images)
        class_index = torch.argmax(prob_tensor, dim=1)
        acc = acc + (label == class_index).sum().item()
        loss_value = criterion(prob_tensor, label)
        opt_model.zero_grad()
        loss_value.backward()
        opt_model.step()
        loss += loss_value.item()
        pbar.set_postfix(**{'loss': loss / (iteration + 1),
```

```
                                    'acc': acc / ((iteration + 1) * label.shape[0])
                                    })
            pbar.update(1)
        print('Start test')
        pbar.close()
        pbar = tqdm(total=epoch_step_val, desc=f'Epoch {epoch + 1}/{Epoch}',
postfix=dict, mininterval=0.3)
        acc = 0
        for iteration, batch in enumerate(val_gen):
            if iteration >= epoch_step_val:
                break
            model_train.eval()
            images, label = batch[0], batch[1]
            for i in range(label.shape[0]):
                train_set.add(int(label[i]))
            with torch.no_grad():
                if cuda:
                    images = images.cuda()
                    label = label.cuda()
            prob_tensor = model_train(images)
            class_index = torch.argmax(prob_tensor, dim=1)
            acc = acc + (label == class_index).sum().item()
            pbar.set_postfix(**{'acc': acc / ((iteration + 1) * label.shape[0]),
                                    })
            pbar.update(1)
        pbar.close()


        save_state_dict = model.state_dict()
        # save_state_dict_gen = Generator.state_dict()
        if (epoch + 1) % save_period == 0 or epoch + 1 == Epoch:
            torch.save(save_state_dict, os.path.join(save_dir, "ep%03d-loss%.3f.pth" %
(epoch + 1, loss / epoch_step)))

        torch.save(save_state_dict, os.path.join(save_dir, "last_epoch_weights.pth"))
```

utils.py（用于图像的初步处理）：

```
import numpy as np
from PIL import Image

#---------------------------------------------------------#
#   将图像转换成 RGB 图像，防止灰度图在预测时报错。
#   代码仅仅支持 RGB 图像的预测，所有其它类型的图像都会转化成 RGB
#---------------------------------------------------------#
def cvtColor(image):
    if len(np.shape(image)) == 3 and np.shape(image)[2] == 3:
        return image
    else:
        image = image.convert('RGB')
        return image
#---------------------------------------------------#
#   对输入图像进行 resize
#---------------------------------------------------#
def resize_image(image, size, letterbox_image):
    iw, ih  = image.size
    w, h    = size
    if letterbox_image:
        scale   = min(w/iw, h/ih)
        nw      = int(iw*scale)
        nh      = int(ih*scale)
        image   = image.resize((nw,nh), Image.BICUBIC)
        new_image = Image.new('RGB', size, (128,128,128))
        new_image.paste(image, ((w-nw)//2, (h-nh)//2))
    else:
        new_image = image.resize((w, h), Image.BICUBIC)
    return new_image

#---------------------------------------------------#
#   获得学习率
#---------------------------------------------------#
def get_lr(optimizer):
```

```
    for param_group in optimizer.param_groups:
        return param_group['lr']

def preprocess_input(image):
    image /= 255.0
    return image

def show_config(**kwargs):
    print('Configurations:')
    print('-' * 70)
    print('|%25s | %40s|' % ('keys', 'values'))
    print('-' * 70)
    for key, value in kwargs.items():
        print('|%25s | %40s|' % (str(key), str(value)))
    print('-' * 70)

def preprocess_input(x):
    x = x.astype('float32')
    x /= 255.
    x -= 0.5
    x *= 2.
    return x
```

model.py（CNN 结构）：

```
import cv2
import kornia
import numpy
from matplotlib import pyplot as plt
import numpy as np
import torch
import torch.nn as nn
from torch import Tensor
import torch.nn.functional as F

class Baseline(nn.Module):
```

```
def __init__(self):
        super(Baseline, self).__init__()
        self.conv_features = nn.Sequential(
                nn.Conv2d(1, 32, kernel_size=3, padding=1),
                nn.BatchNorm2d(32),
                nn.LeakyReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2),
                nn.Conv2d(32, 64, kernel_size=3, padding=1),
                nn.BatchNorm2d(64),
                nn.LeakyReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2),
                nn.Conv2d(64, 128, kernel_size=3, padding=1),
                nn.BatchNorm2d(128),
                nn.LeakyReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2),
                nn.Conv2d(128, 256, kernel_size=3, padding=1),
                nn.BatchNorm2d(256),
                nn.LeakyReLU(inplace=True),
                nn.MaxPool2d(kernel_size=2)
        )

        self.classifier = nn.Sequential(
                nn.Dropout(p=0.5),
                nn.Linear(256, 512),
                nn.BatchNorm1d(512),
                nn.LeakyReLU(inplace=True),
                nn.Dropout(p=0.5),
                nn.Linear(512, 256),
                nn.BatchNorm1d(256),
                nn.LeakyReLU(inplace=True),
                nn.Dropout(p=0.5),
                nn.Linear(256, 128),
                nn.BatchNorm1d(128),
                nn.LeakyReLU(inplace=True),
```

```
            nn.Linear(128, 198)
        )

    def forward(self, x):    # size(x) == (B,1,28,28)
        x = self.conv_features(x)
        x = x.view(x.size(0), -1)
        # print(x.shape)
        x = self.classifier(x)
        # x = F.softmax(x, dim=1)
        return x
```

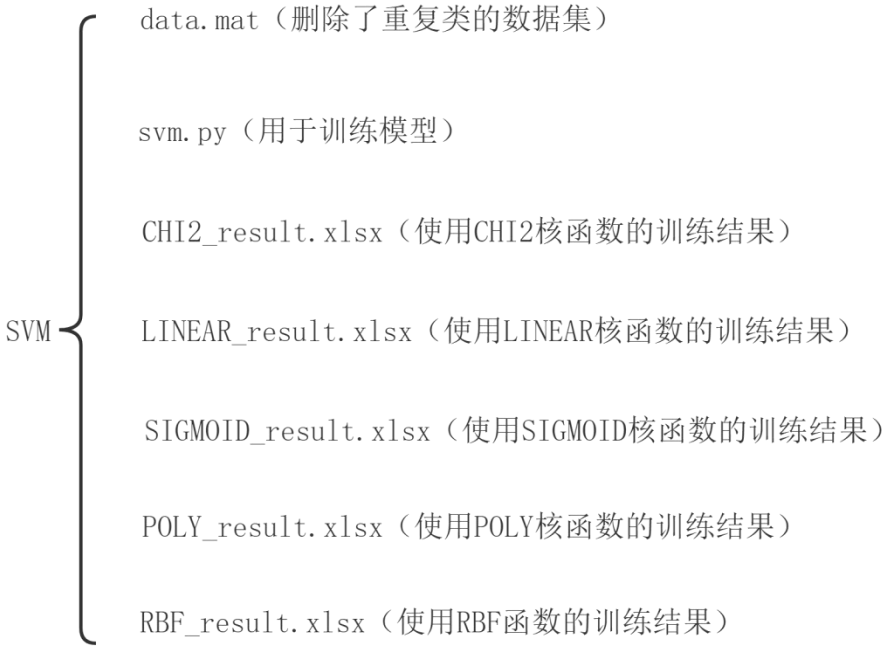
train.py（模型训练）：

```
import os.path     # 导入 os.path 模块，用于操作文件路径
from torch.optim.lr_scheduler import StepLR
from torch.optim.lr_scheduler import MultiStepLR
import torch       # 导入 PyTorch 库
import torch.backends.cudnn as cudnn       # 导入 cudnn 模块，用于提升 PyTorch 的计算速度
from torch.utils.data import DataLoader # 导入 DataLoader 类，用于加载数据集
from myutils.dataloader_cl import Dataset, dataset_collate       # 导入自定义数据集和数据集处理函数
from myutils.trainer import fit_one_epoch        # 导入自定义训练函数
from nets.model import Baseline        # 导入自定义神经网络模型

if __name__ == "__main__":
    Cuda = False       # 是否使用 GPU 进行训练，默认为 False
    pretrained_model_path    = 'logs/last_epoch_weights.pth'# 预训练模型的路径，默认为空 logs/last_epoch_weights.pth
    input_shape = [28, 28]    # 输入图像的大小，默认为 28*28
    batch_size = 64 # 每个 batch 的大小，默认为 32
    Init_Epoch = 0          # 起始迭代次数，默认为 0
    Epoch = 300         # 起始迭代次数，默认为 0
    lr = 0.0003
    save_period = 50             # 每隔多少个 epoch 保存一次模型权重，默认为 50
    save_dir = 'logs/'     # 权重和日志文件保存的文件夹名称，默认为'logs/'
```

```
    # 如果指定文件夹不存在，则创建文件夹
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)
    num_workers = 0        # 数据集加载器使用的进程数，默认为 0
    train_val_dataset_path = 'dataset/data.mat'
    ngpus_per_node = torch.cuda.device_count()  # 获取可用的 GPU 数量
    # 设置设备为 GPU，如果没有 GPU 则使用 CPU
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    model = Baseline()  # 实例化模型对象
    if pretrained_model_path != '':  # 如果预训练模型路径不为空，则加载预训练模型的权重
        print('Load weights {}.'.format(pretrained_model_path))
        pretrained_dict = torch.load(pretrained_model_path, map_location= device)
        model.load_state_dict(pretrained_dict)
    model_train = model.train()        # 将模型设置为训练模式

    # 如果使用 GPU 进行训练，则使用 DataParallel 将模型转换为可以在多个 GPU 上运行的模型，并设置 cudnn.benchmark 以加速模型的计算
    if Cuda:
        Generator_train = torch.nn.DataParallel(model)
        cudnn.benchmark = True
        Generator_train = Generator_train.cuda()

    opt_model = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=0.0005)
    # 构建学习率衰减，以下是两种，在动态调整模型参数时使用
    # scheduler = StepLR(opt_model, step_size=1000, gamma=0.1)
    # lr_scheduler = MultiStepLR(opt_model, milestones= [20, 80, 160, 240], gamma=0.1, last_epoch=-1, verbose=False)
    train_dataset = Dataset(train_val_dataset_path, input_shape, epoch_length=Epoch, is_train=True)
    val_dataset = Dataset(train_val_dataset_path, input_shape, epoch_length=Epoch, is_train=False)
    shuffle = True  # 表示是否打乱数据集
```

```
    # 创建训练集数据加载器，使用 PyTorch 内置的 DataLoader 类，并将训练集数据集、是否打乱数据集、batch_size、线程数、是否将数据加载到 GPU 上、是否丢弃最后一批数据、数据集合并方式、采样器等作为参数
    train_gen = DataLoader(train_dataset, shuffle=shuffle, batch_size=batch_size, num_workers=num_workers,
                           pin_memory=True, drop_last=True, collate_fn=dataset_collate, sampler=None)
    # 创建测试集数据加载器
    val_gen = DataLoader(val_dataset, shuffle=shuffle, batch_size=batch_size, num_workers=num_workers,
                           pin_memory=True, drop_last=True, collate_fn=dataset_collate, sampler=None)

    for epoch in range(Init_Epoch, Epoch):
        # 计算每个 epoch 中的步数
        epoch_step = train_dataset.length // batch_size
        epoch_step_val = val_dataset.length // batch_size
        # 更新数据集的当前 epoch 值
        train_gen.dataset.epoch_now = epoch
        val_gen.dataset.epoch_now = epoch
        ## 使用 fit_one_epoch 函数进行模型训练
        fit_one_epoch(model_train, model, opt_model, epoch, epoch_step, epoch_step_val, train_gen, val_gen, Epoch, Cuda, save_period, save_dir)
        # scheduler.step()
        # lr_scheduler.step()
        opt_model.step()
```

附 2.2 SVM 模型整个项目结构：



svm.py（SVM 模型）：

```
# @Author: Jinhai Xu
# @Time: 2023/4/1 2:33
# @IDE: PyCharm
# @Project Name: svm_1 -> svm
import cv2                                  #导入 opencv 库
import numpy as np                          #导入 numpy 库
import matplotlib.pyplot as plt         #导入 matplotlib 库
from collections import defaultdict #导入 defaultdict 类，它是字典的子类
import scipy.io as scio                     #导入 scipy.io 库
import keras                                #导入 keras 库
import pandas as pd

# 分离训练集的图像和标签
def split_data(data):
    X = np.resize(data, [2970, 28, 28])   # 将输入数据 data 重新变形为大小为[2970, 28, 28]的数组
    label = np.zeros([2970])                  # 初始化大小为 2970 的 0 数组作为标
```

```
签
    label_info = np.array([0 for i in range(15)])   # 初始化一个大小为 15 的数组作为标签信息
    for i in range(198):
        label[i * 15:15 + i * 15] = label_info     # 将每 15 个图像的标签设为相同的数字
        label_info += 1
    return X, label

# 分离测试集的图像和标签
def split_test(data):
    X = np.resize(data, [990, 28, 28])     # 将输入数据 data 重新变形为大小为[990, 28, 28]的数组
    label = np.zeros([990])                    # 初始化大小为 990 的 0 数组作为标签
    label_info = np.array([0 for i in range(5)])   # 初始化一个大小为 5 的数组作为标签信息
    for i in range(198):
        label[i * 5:5 + i * 5] = label_info         # 将每 5 个图像的标签设为相同的数字
        label_info += 1
    return X, label

# 导入数据集
dataFile = 'data.mat'
data = scio.loadmat(dataFile)

# train_dataset 和 train_labels 代表训练集的图像与标签, test_dataset 与 test_labels 代表测试集的图像与标签
(train_dataset, train_labels) = split_data(data['train'])     # 将训练集的数据分离为图像和标签
(test_dataset, test_labels) = split_test(data['test'])          # 将测试集的数据分离为图像和标签

SIZE_IMAGE = train_dataset.shape[1]     # 图像的大小为 28x28
```

```
train_labels = np.array(train_labels, dtype=np.int32)    # 将标签转换为 int32 类型的 numpy 数组

# 预处理函数
def deskew(img):
    m = cv2.moments(img)    # 计算输入图像的矩
    if abs(m['mu02']) < 1e-2:    # 判断是否需要进行校正
        return img.copy()
    skew = m['mu11'] / m['mu02']    # 计算图像的倾斜角
    M = np.float32([[1, skew, -0.5 * SIZE_IMAGE * skew], [0, 1, 0]])    # 创建变换
    img = cv2.warpAffine(img, M, (SIZE_IMAGE, SIZE_IMAGE), flags=cv2.WARP_INVERSE_MAP | cv2.INTER_LINEAR)
    return img
#
# # HOG 高级描述符
# hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,
#
histogramNormType,L2HysThreshold,gammaCorrection,nlevels)
def get_hog():
    hog = cv2.HOGDescriptor((SIZE_IMAGE, SIZE_IMAGE), (8, 8), (4, 4), (8, 8), 9, 1, -1, 0, 0.2, 1, 64, True)
    # 初始化 HOG 描述符
    print("hog descriptor size: {}".format(hog.getDescriptorSize()))
    return hog
'''
# 模型初始化函数 SVM_CHI2, SVM_RBF, SVM_LINEAR, SVM_POLY, SVM_SIGMOID
修改核函数时，需要手动修改 model.setKernel(cv2.ml.SVM_CHI2)中的核函数，如上图所示
'''
```

```
def svm_init(C=12.5, gamma=0.50625):
    model = cv2.ml.SVM_create()  # 创建 SVM 模型
    model.setKernel(cv2.ml.SVM_CHI2)
    model.setGamma(gamma)  # 设置 SVM 模型参数
    model.setC(C)
    # model.setCoef0(0.1)
    # 在使用 POLY 核函数时需要下面的语句设置多项式的次数
    # model.setDegree(3)
    # 在使用 NOVA 核函数时需要下面的语句设置 degree 参数
    # model.setDegree(2)
    model.setType(cv2.ml.SVM_C_SVC)
    model.setTermCriteria((cv2.TERM_CRITERIA_MAX_ITER, 100, 1e-6))
    # 设置 SVM 模型的训练终止条件
    return model

# 模型训练函数，用于训练 SVM 模型
def svm_train(model, samples, responses):
    model.train(samples, cv2.ml.ROW_SAMPLE, responses)
    # 对 SVM 模型进行训练
    return model

# 模型预测函数，用于对新样本进行预测
def svm_predict(model, samples):
    return model.predict(samples)[1].ravel()

# 模型评估函数，用于评估 SVM 模型的分类准确率
def svm_evaluate(model, samples, labels):
    predictions = svm_predict(model, samples)  # 对样本进行预测
    acc = (labels == predictions).mean()  # 计算分类准确率
    return acc * 100

def rotate(image, angle):
    '''
    对图像进行旋转增强
```

```
    '''
    rows, cols = image.shape
    # 构造旋转矩阵
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
    # 进行仿射变换
    dst = cv2.warpAffine(image, M, (cols, rows))
    return dst

def shift(image, x, y):
    '''
    对图像进行平移增强
    '''
    rows, cols = image.shape
    M = np.float32([[1, 0, x], [0, 1, y]])
    dst = cv2.warpAffine(image, M, (cols, rows))
    return dst

def zoom(image, scale):
    '''
    对图像进行缩放增强
    '''
    rows, cols = image.shape
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 0, scale)
    dst = cv2.warpAffine(image, M, (cols, rows))
    return dst

def enhance(image):
    '''
    对图像进行一系列增强操作，生成新的增强图像
    '''
    images = []
    angles = [-10, -5, 5, 10]  # 旋转角度列表
    shifts = [(5, 5), (-5, -5), (5, -5), (-5, 5)]  # 平移距离列表
    scales = [0.9, 1.1]  # 缩放比例列表
```

```
        # 旋转增强
        for angle in angles:
            img = rotate(image, angle)
            images.append(img)
        # 平移增强
        for shift_x, shift_y in shifts:
            img = shift(image, shift_x, shift_y)
            images.append(img)
        # 缩放增强
        for scale in scales:
            img = zoom(image, scale)
            images.append(img)
        return images
# 对训练集图像进行增强
augmented_images = []
augmented_labels = []
for i in range(len(train_dataset)):
        image = train_dataset[i]
        label = train_labels[i]
        augmented_images.append(image)
        augmented_labels.append(label)
        enhanced_images = enhance(image)
        for enhanced_image in enhanced_images:
            augmented_images.append(enhanced_image)
            augmented_labels.append(label)

# 转化为 numpy 数组
augmented_images = np.array(augmented_images)
augmented_labels = np.array(augmented_labels)

# 数据打散，将原来顺序排列的图像打乱，增加随机性
shuffle = np.random.permutation(len(augmented_images))
augmented_images, augmented_labels = augmented_images[shuffle],
```

```
augmented_labels[shuffle]
## 数据打散，将原来顺序排列的图像打乱，增加随机性
# shuffle = np.random.permutation(len(train_dataset))
# train_dataset, train_labels = train_dataset[shuffle], train_labels[shuffle]

# 使用 HOG 描述符
hog = get_hog()
hog_descriptors = []
for img in augmented_images:
    # hog_descriptors.append(hog.compute(deskew(img)))
    img = cv2.convertScaleAbs(img)
    hog_descriptors.append(hog.compute(img))
hog_descriptors = np.squeeze(hog_descriptors)

# 训练数据与测试数据划分
partition = int(0.9 * len(hog_descriptors))
hog_descriptors_train, hog_descriptors_test = np.split(hog_descriptors, [partition])
labels_train, labels_test = np.split(augmented_labels, [partition])

print('Training SVM model ...')
results = defaultdict(list)

result_df = pd.DataFrame(columns=["C", "gamma", "accuracy"])

result = []

for C in [0.1, 1, 3, 5, 8, 12, 20, 40, 50, 100]:
    results[C] = []
    for gamma in [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.015, 0.025, 0.05, 0.1, 0.2,
0.3, 0.5, 1]:
        model = svm_init(C, gamma)
        svm_train(model, hog_descriptors_train, labels_train)
        testacc = svm_evaluate(model, hog_descriptors_test, labels_test)
        print("C = %.1f , gamma = %.4f" % (C, gamma))
```

```
            print(" {}".format("accuracy = %.2f" % testacc))
            result_df = pd.concat([result_df, pd.DataFrame({"C": [C], "gamma":
[gamma], "accuracy": [testacc]})], ignore_index=True)
            results[C].append(testacc)
            result.append(testacc)
result_df.to_excel("CHI2_result.xlsx", index=False)
result.sort(reverse=True)
print(result)


# 可视化结果
fig = plt.figure(figsize=(10, 6))
plt.suptitle("SVM WITH CHI2 KERNEL", fontsize=16, fontweight='bold')
ax = plt.subplot(1,1,1)
ax.set_xlim(0, 0.5)
dim = [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.015, 0.025, 0.05, 0.1, 0.2, 0.3, 0.5, 1]

for key in results:
    ax.plot(dim, results[key], linestyle='--', marker='o', label=str(key))
plt.legend(loc='upper right', title="C")
plt.title('Accuracy of the SVM model varying both C and gamma')
plt.xlabel("gamma")
plt.ylabel("accuracy")
plt.show()
```