1 MNIST 数据集简介

MNIST 数据集由美国国家标准与技术研究所制作的，它是由来自 250 个不同人的手写数字构成，其中 50% 是高中学生, 50% 来自人口普查局的工作人员，测试集也是同样比例的手写数字数据。该数据集包含四个文件，分别是训练集数据以及其标签、测试集数据以及其标签数据。训练集中包含 60000 个训练数据，其形状是一个 60000*784 的矩阵，每一行代表一张图片的 784 个特征，可以构成一张 28*28 的灰度图像。训练数据的标签是介于 0-9 之间的数字，其采用 one-hot 离散型特征编码方式，每个标签一共 10 个维度，除了对应位置上为 1 外，其他的位置上都是 0，一共能表示 10 个不同的数字，因此其标签为一个 60000*10 的矩阵。测试集中仅有 10000 个数据，数据结构与训练集相似。

2 卷积神经网路 CNN

卷积神经网路与常规神经网络相似，都是由神经元组成，神经元中具有学习能力的权重和偏差，每个神经元都得到一些输入数据，进行内积运算后进行激活函数的运算，最后一层往往是全连接层，最后输出得到一个可导的评分结果。与常规神经网络不同的是，卷积神经网络基于输入数据是图像数据，向该结构中使用不同的卷积核，使得前向传播实现起来更加高效，并且大幅度降低了网络中的参数数量。卷积神经网络通常由三种类型的层构成：卷积层，池化层和全连接层，将这些网络堆叠起来构建一个完整的卷积神经网络。

2.1 CNN 各层结构

2.1.1 卷积层

卷积层是构建卷积神经网络的核心层，产生了网络中大部分的计算量。在处理图像这样高维度的数据输入时，由于让神经元与前一层这样的数据进行全连接时得到的参数量太过巨大，在计算时不现实，因此只让神经元与输入数据的局部区域相连接，通过卷积核在输入数据上的宽度和高度上滑动，逐步与整个图像的局部区域做卷积，最后将卷积得到的内积和通过激活函数运算得到该卷积层的输出。卷积核在局部区域上与输入数据连接，但是在深度方向上，卷积核的深度与输入数据的深度是一致的。通过使用这种卷积核的方式使得该过程中的参数数量大大降低。 在训练过程中，卷积核的参数是通过不断地前向传播和反向传播中逐渐学习到的。

2.1.2 池化层

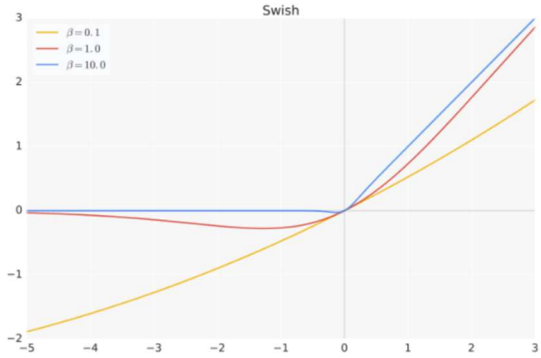
通常在连续的卷积层之间会周期性的插入一个池化层，其作用是逐渐降低数据体的空间尺寸，逐步减少网络中的参数的数量，使得耗费的资源变少。池化过程实际上就是一个下采样的过程，下采样使得图像数据减少，但是特征的统计特性任能描述图像，而且降低了数据维度，有效地避免了过拟合。Z 在实际操作中，可分为最大值下采样和平均值下采样。

2.1.2 全连接层

全连接层的每一个结点都与上一层的所有结点相连，用来把前边提取到的特征综合起来。由于其全相连的特性，一般全连接层的参数也是最多的。全连接层将学到的“分布式特征表示”映射到样本标记空间，但是缺点在于其会破坏图像的空间结构，因此一般卷积神经网络在最后几层“分类”时才会使用到全连接层。

2.2 Swish 激活函数

在神经网络中，激活函数的功能即是对加权的输入进行非线性组合产生非线性决策边界，简单来说就是增加其非线性特性。本实验中使用 Google



近年来提出的 Swish 激活函数，如图 1 所示，Swish 具备无上界有下界、平滑、非单调的特性，其表达式可如下式所示：

$$f(x) = x \cdot sigmoid(\beta x)$$

图 1：Swish 激活函数

2.3 退火算法

退火算法的思想借鉴于固体的退火原理。当固体的温度很高的时候，内能比较大，固体的内部粒子处于快速无序运动，当温度慢慢降低的过程中，固体的内能减小，粒子的慢慢趋于有序，最终，当固体处于常温时，内能达到最小，此时，粒子最为稳定。

模拟退火算法从某一较高的温度出发，这个温度称为初始温度，伴随着温度参数的不断下降，算法中的解趋于稳定，但是，可能这样的稳定解是一个局部最优解，此时，模拟退火算法中会以一定的概率跳出这样的局部最优解，以寻找目标函数的全局最优解。

2.4 Adam 优化器

Adam 优化器由 Kingma 和 Lei Ba 两位学者于 2014 年提出，结合了 daGrad 和 RMSProp 两种优化算法的优点。对梯度的一阶矩估计和二阶矩估计进行综合考虑，计算出更新步长。

Adam 优化器具有以下优点：实现简单，计算搞笑，对内存需求少；参数的更新不受梯度的伸缩变化影响；能自然地实现步长退火过程，自动调整学习率，适用于大规模数据及参数的场景。Adam 优化器是目前工作性能比较优秀的常用优化器。

3 实验方法

3.1 加载 mnist 数据集与数据预处理

实验所用深度学习框架为 keras 框架，加载 mnist 数据使用的是 keras 中的 datasets 模块，即 from keras.datasets import mnist。这些数据集是与官网的数据一致的，因而能够满足本实验数据集的公平性。通过 load_data 函数可以加载数据集的训练数据及其相应标签、测试数据及其相应标签。

对数据集的数据进行简单的归一化预处理，将数据范围从[0,255]转换至[0,1]，这种做法既能够在保留所有的特征的基础上保证输出数据中数值小的不被吞食，也能减少整体参数的大小。由于激活函数的非线性作用，当较大范围的数据进入网络，更加容易出现非线性抑制和饱和，那么反向传播参数调整时范围有限，其效果会被抑制，而输入数据的范围小一些的话，神经元的输出值非线性抑制的程度小些，这样参数调整的范围更大，效果会更好，同时，也会加快网络的收敛速度。

接着，为了获得图片的二维空间信息，需要将训练数据中含有 784 个值的 1 维向量转换成 28x28x1 的 3 维矩阵。另外，对标签数据进行 one-hot 编码，从而方便后续将问题转化为多分类问题。

最后，采用随机抽取的方法将训练集按照 9：1 的比例参数分为实际的训练集和验证集，其中，前者作为网络实际训练的数据集，后者仅在后续过程中作为迭代数以及学习率等超参数调整的验证指标。

3.2 建立训练网络

输入层传入 mnist 数据，对图片矩阵的边界填充 0，以控制卷积之后特征图的大小。前 2 个卷积层用的均是 32 个 5×5 的卷积核，步长为 1，使用的激活函数为 swish 函数即 sigmoid(x) * x，如图 2 所示。将结果传入池化层，使图片矩阵发生尺度变化。考虑到网络结构相对简单，加入了 dropout 操作，以防止过拟合。经过调整，该 dropout 层的比例参数设置为 0.25。

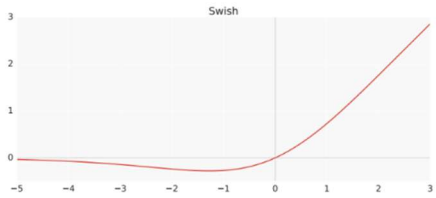


图 2Swish 函数

后 2 个卷积层用的是 64 个 3×3 的卷积核，步长为 1，使用的激活函数也是 swish 函数，将结果传入池化层后执行 dropout 操作，dropout 层的比例参数经调整确定为 0.25。

经过卷积池化后，将数据传入 Flatten 层，将多维的输入进行一维化展开，接着传入具有 1024 个神经元的全连接层，同样的，激活函数也使用 swish。经过 0.5 比例的 dropout 操作后，再传入具有 10 个神经元的全连接层，并以此层作为输出层，不同的是，此层使用 sigmoid 激活函数，以实现多分类操作。训练网络结构图如图 3 所示。

3.3 优化模型与学习率调整策略

选用类别交叉熵对训练模型进行优化，以经过 one-hot 处理的训练集 label 与输出层的分类结果计算类别交叉熵作为 loss 参数，并选用 Adam 优化器不断地减小该 loss 参数。Adam 优化器结合了 AdaGrad[4]和 RMSProp 两种优化算法的优点，是目前常用的优化器中效果较好的一个。

为了防止过拟合的出现，需要对训练数据进行数据增强。本文采用对数据集的图片数据进行旋转、缩放、水平平移、垂直平移等数据增强方法，提升训练模型的鲁棒性和泛化能力。需要注意的是，不能采用镜像翻转操作，避免 6 和 9 的混淆从而识别错误。

学习率的调整在一定程度上可以提升训练效果。已知适当偏大的学习率会加快收敛速度，但是却牺牲了识别准确度，而适当偏小的学习率会让系统更加趋近全局极小值，但收敛速度相对较慢。所以本实验训练过程中前期的学习率较大，让系统以较快的速度收敛，后续结合验证集的识别结果来确定学习率的调整结果。当发现训练过程中验证集的识别准确率在 3 次之内没有提升时，将会减半学习率，直到训练结束或者达到学习率的下限值。

3.4 训练模型并测试

对数据进行 batch 批处理，训练迭代次数设置为 30 次，运用退火算法调整模型参数。迭代 30 次后生成模型。最后使用测试集数据对生成的模型进行评估，计算测试集正确预测的数量并算出模型的识别准确率。

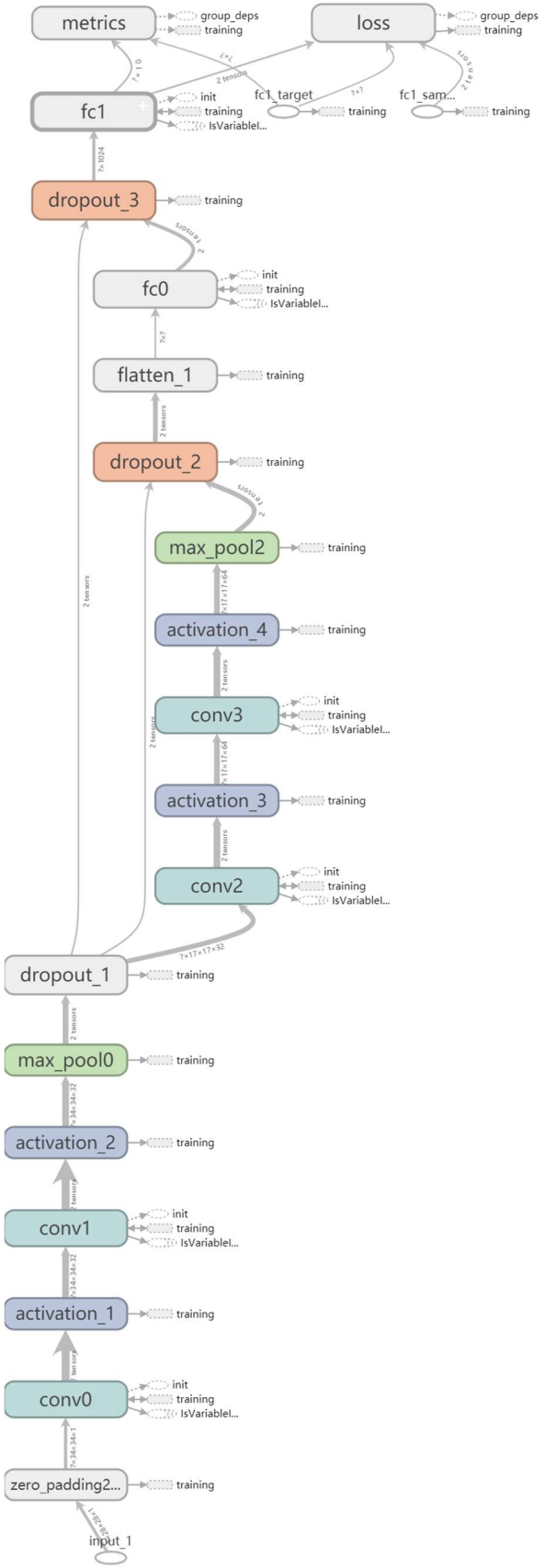


图 3 网络结构

4 实验结果

```
Using TensorFlow backend.
训练集特征大小 (60000, 28, 28)
训练集标签大小 (60000, 10)
测试集特征大小 (10000, 28, 28)
测试集标签大小 (10000, 10)
```

图 4 训练集大小

```
Epoch 1/30
 - 29s - loss: 0.3366 - acc: 0.8956 - val_loss: 0.0601 - val_acc: 0.9812
Epoch 2/30
 - 27s - loss: 0.1105 - acc: 0.9668 - val_loss: 0.0344 - val_acc: 0.9902
Epoch 3/30
 - 27s - loss: 0.0852 - acc: 0.9747 - val_loss: 0.0354 - val_acc: 0.9888
Epoch 4/30
 - 27s - loss: 0.0751 - acc: 0.9784 - val_loss: 0.0291 - val_acc: 0.9902
Epoch 5/30
 - 27s - loss: 0.0691 - acc: 0.9797 - val_loss: 0.0270 - val_acc: 0.9908
Epoch 6/30
 - 27s - loss: 0.0627 - acc: 0.9826 - val_loss: 0.0343 - val_acc: 0.9912
Epoch 7/30
 - 27s - loss: 0.0625 - acc: 0.9829 - val_loss: 0.0230 - val_acc: 0.9935
Epoch 8/30
 - 27s - loss: 0.0569 - acc: 0.9839 - val_loss: 0.0262 - val_acc: 0.9923
Epoch 9/30
 - 27s - loss: 0.0579 - acc: 0.9838 - val_loss: 0.0378 - val_acc: 0.9912
Epoch 10/30
 - 27s - loss: 0.0561 - acc: 0.9854 - val_loss: 0.0301 - val_acc: 0.9908
Epoch 11/30
 - 27s - loss: 0.0537 - acc: 0.9854 - val_loss: 0.0256 - val_acc: 0.9912

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 12/30
 - 28s - loss: 0.0375 - acc: 0.9894 - val_loss: 0.0224 - val_acc: 0.9938
Epoch 13/30
 - 28s - loss: 0.0343 - acc: 0.9909 - val_loss: 0.0224 - val_acc: 0.9933
Epoch 14/30
 - 28s - loss: 0.0310 - acc: 0.9915 - val_loss: 0.0207 - val_acc: 0.9942
Epoch 15/30
 - 28s - loss: 0.0315 - acc: 0.9910 - val_loss: 0.0196 - val_acc: 0.9940
Epoch 16/30
 - 28s - loss: 0.0288 - acc: 0.9921 - val_loss: 0.0219 - val_acc: 0.9952
Epoch 17/30
 - 28s - loss: 0.0290 - acc: 0.9915 - val_loss: 0.0225 - val_acc: 0.9938
Epoch 18/30
 - 28s - loss: 0.0278 - acc: 0.9918 - val_loss: 0.0233 - val_acc: 0.9945
Epoch 19/30
 - 28s - loss: 0.0280 - acc: 0.9918 - val_loss: 0.0226 - val_acc: 0.9947
Epoch 20/30
 - 28s - loss: 0.0272 - acc: 0.9922 - val_loss: 0.0192 - val_acc: 0.9955
Epoch 21/30
 - 28s - loss: 0.0261 - acc: 0.9927 - val_loss: 0.0243 - val_acc: 0.9942
Epoch 22/30
 - 28s - loss: 0.0246 - acc: 0.9928 - val_loss: 0.0197 - val_acc: 0.9963
Epoch 23/30
 - 28s - loss: 0.0241 - acc: 0.9931 - val_loss: 0.0196 - val_acc: 0.9952
Epoch 24/30
 - 28s - loss: 0.0241 - acc: 0.9933 - val_loss: 0.0247 - val_acc: 0.9937
Epoch 25/30
 - 28s - loss: 0.0262 - acc: 0.9929 - val_loss: 0.0213 - val_acc: 0.9945
Epoch 26/30
 - 28s - loss: 0.0233 - acc: 0.9936 - val_loss: 0.0230 - val_acc: 0.9945

Epoch 00026: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
Epoch 27/30
 - 28s - loss: 0.0213 - acc: 0.9941 - val_loss: 0.0201 - val_acc: 0.9950
Epoch 28/30
 - 28s - loss: 0.0180 - acc: 0.9947 - val_loss: 0.0202 - val_acc: 0.9952
Epoch 29/30
 - 30s - loss: 0.0169 - acc: 0.9948 - val_loss: 0.0197 - val_acc: 0.9948

Epoch 00029: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
Epoch 30/30
 - 29s - loss: 0.0156 - acc: 0.9955 - val_loss: 0.0177 - val_acc: 0.9952
```

图 5 训练结果

训练过程中，每一次迭代的输出结果中，训练集的 loss 值不断的总体上不断的降低，而验证集的 loss 值也相应的在不断降低，而且总体上比训练集的要小一些，从而说明系统的训练过程没有出现过拟合的现象，系统的泛化能力初步估计较强。

图 4 为训练集大小，图 5 为模型的训练结果。另外，结合训练过程曲线图 6-图 10，可知每当验证集的准确率减半后，训练集和 loss 曲线变化出现跳变点，即学习率的减小在一定程度上使得训练的模型进一步收敛。

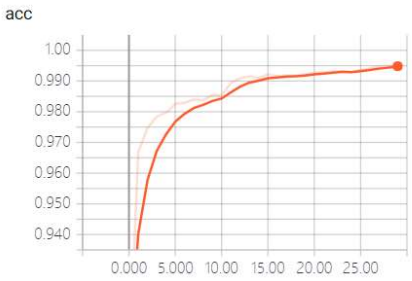


图 6 训练集准确率

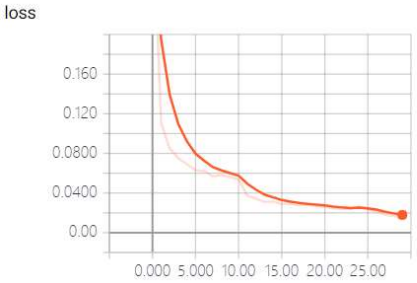


图 7 训练集损失

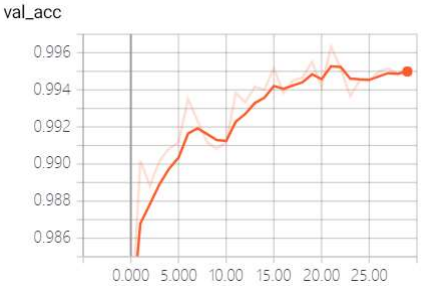


图 8 验证集准确率

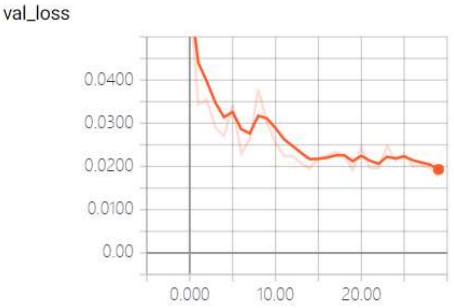


图 9 验证集损失

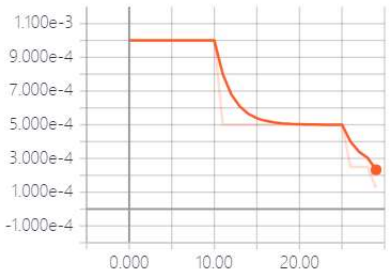


图 10 学习率变化

```
------------------------------
测试集数量：  10000
正确的数量：  9964
测试准确率：  0.9964
------------------------------
```

图 11 测试集数据的测试结果

从实验结果中可以看出，训练集共有 60000 个，其中，实际传入网络的进行训练的数据集测试集有 54000 个，而验证集有 6000 个，测试集共有 10000 个。实验使用的是 GTX1050Ti 显卡，对网络模型进行 30 次迭代训练，每次迭代时间为 27s~30s。

第 1 次的训练损失为 0.3366，训练精确度为 0.8956，验证集的验证损失为 0.0601，验证精确度为 0.9812。经过 30 次的迭代训练后，训练损失降为 0.0156，训练精确度提升至 0.9955，验证集的损失降为 0.0177，验证精确度提升至 0.9952。

Adam 优化器中设定的学习率初始值为 0.001，训练过程中训练网络对学习率进行自动调整，学习率变化曲线如图 10 所示。

对测试集数据进行测试，如图 11 所示，测试集数据数量为 10000，准确分类的数量为 9964，所以网络模型在测试集上的准确率为 99.64%。

5 结论

通过 keras 深度学习框架，建立了特定的卷积网络结构，并选用合适的优化函数等和损失函数，调整了合适的参数和策略，实现了 Mnist 手写字符数据集的识别分类。经过对模型的 30 次迭代训练后，训练集精确度为 0.9955，损失为 0.0156，在测试集上进行测试，达到了 99.64%的准确率。可见，本实验建立的深度网络模型具有较强的泛化能力和鲁棒性，能够获得对手写数字较高的识别准确率。