# 实验报告

**陈婧明 202318019427004**

实验代码用 Python 语言编写，在 PyCharm 2022 平台上开发。

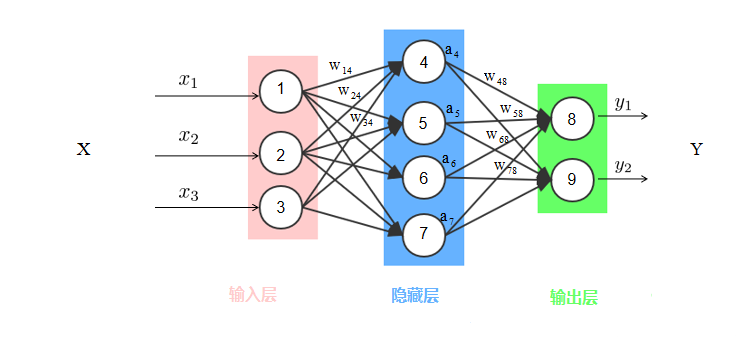
本机上的实验使用的深度学习框架是 PyTorch 1.8.2，显卡为 NVIDIA GeForce RTX 3050，CUDA 版本为 11.4。

在colab平台上实验使用PyTorch 2.0.1，显卡Tesla T4，CUDA 版本为 11.8。

数据读取：读入所有的2500张图片（dog：1250张，cat：1250张），按照训练集：验证集：测试集=6:2:2的比例划分数据集。

## DNN模型

DNN（全连接神经网络）一般分为三层，即输入层、隐藏层和输出层，网络中除输入层之外的每个节点都和上一层的所有节点有连接。网络结构如下图所示。



DNN网络结构

* + - 1. **网络结构**

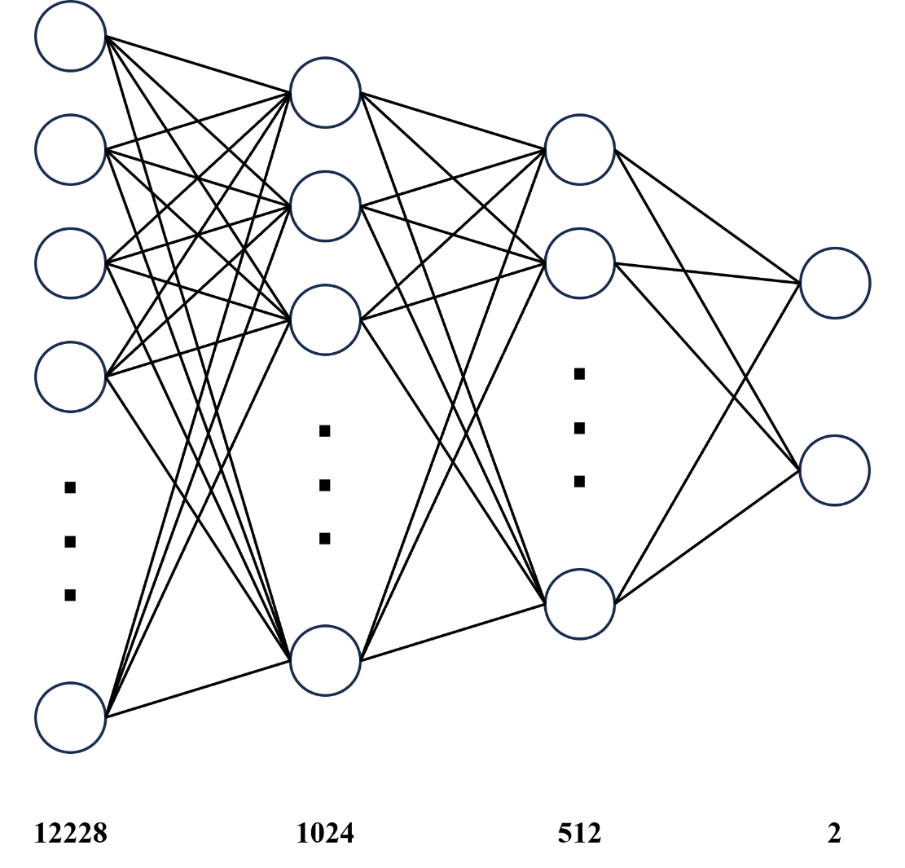
在pytorch中使用函数torch.nn.Linear()实现线性变换，使用函数nn.functional.relu()实现relu激活函数。

fc1：输入维度3 \* 64 \* 64，输出维度1024，激活函数relu，加dropout防止过拟合；

fc2：输入维度1024，输出维度512，激活函数relu；

fc3：输入维度512，输出维度2（二分类），激活函数relu，使用softmax将输出映射到[0, 1]之间。

网络结构如图所示。



* + - 1. **实验结果**

在DNN的实验中，图像尺寸为64\*64，进行降采样的目的是减少训练时间。

随机抽取训练集、验证集和测试集，进行了5次实验。记录测试集的分类准确率结果如下表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | Ave |
| 准确率 | 77% | 80% | 74% | 76% | 77% | 76.8% |

* + - 1. **问题思考**

1. pytorch中的softmax函数与交叉熵损失函数。

Softmax把输入映射为 0-1 之间的实数，并且通过归一化保证和为 1。假设一个数组为 V，表示 V 中的第 i 个元素，那么这个元素经历了 Softmax 函数后的输出为，这里进行指数化的目的是为了扩大任意两个输入之间的差距。在利用神经网络解决分类问题时，一般都会在输出的最后一层加上 Softmax 函数，用于规则化输出 —— 输出为每个类别的概率，且概率之和为 1。

在 二分类的任务 中，模型最后需要预测的结果只有两种情况，对于每个类别，我们的预测得到的概率为 p 和 1−p 。此时的交叉熵损失（又叫二进制交叉熵）为：



其中，表示样本 i 的真实标签值，正类为 1，负类为 0，表示样本 i 的预测为正的概率。

 在多分类任务中，交叉熵损失的函数为：

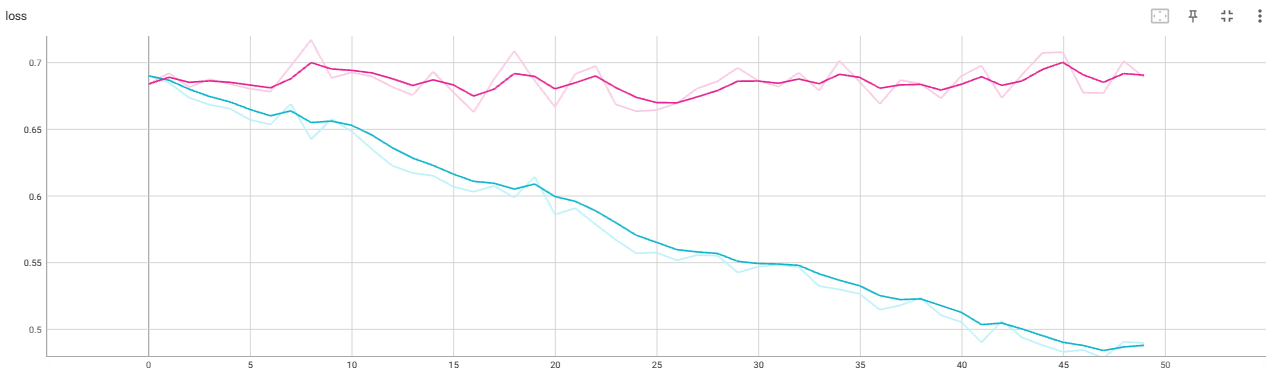


其中，M为类别的数量，标签的 one-hot 编码，如果该类别和样本 i 的类别相同，就是 1，否则为 0。对于观察样本 i 属于类别 c 的预测概率。

利用 PyTorch 中的 nn.CrossEntropyLoss() 定义多分类任务的交叉熵损失函数。但是，实际上 nn.CrossEntropyLoss() 是包含了 nn.LogSoftmax() 和 nn.NLLLoss() ，所以在使用nn.CrossEntropyLoss() 时，网络中没有写nn.LogSoftmax()。

1. DNN网络的参数量巨大，对于一幅H\*W的彩色图像，第一层的输入为各个像素值，因此共有C\*H\*W个输入神经元，受GPU算力限制，只能将图像降采样至64\*64，再输入网络进行训练。
2. 由于训练集数据量有限（2500\*0.6张），网络很容易过拟合，在训练过程中，训练集损失下降，但验证集损失基本不变，损失函数下降趋势如图所示，图中蓝色为训练集的loss，红色为验证集的loss。

尝试通过图像增强，即添加噪声、对图像进行水平或翻转，在网络中添加Dropout后，仍然无法解决过拟合的问题。



DNN网络loss

## CNN模型

卷积神经网络适用于处理图像，它由若干卷积池化层以及若干全连接层构成。

* + - 1. **网络结构**

如图所示，网络使用3层卷积层（每层包含卷积层、BN层和最大值池化层）+3层全连接层。

在pytorch中使用函数torch.nn.Conv2d()实现2D卷积 ，使用函数nn.BatchNorm2d ()进行数据的归一化处理，使用函数nn.MaxPool2d() 实现最大池化。

conv1：

nn.Conv2d(in\_channels=3, out\_channels=16, kernel\_size=3, stride=2)：输入3\*256\*256，输出16\*127\*127；

nn.BatchNorm2d(16)：输入16\*127\*127，输出16\*127\*127；

nn.ReLU()

nn.MaxPool2d(kernel\_size=2)：输入16\*127\*127，输出16\*63\*63

conv2：

nn.Conv2d(in\_channels=16, out\_channels=32, kernel\_size=3, stride=2) ：输入16\*63\*63，输出32\*31\*31；

nn.BatchNorm2d(32) ：输入32\*31\*31，输出32\*31\*31；

nn.ReLU()；

nn.MaxPool2d(kernel\_size=2) ：输入32\*31\*31，输出32\*15\*15

conv3：

nn.Conv2d(in\_channels=32, out\_channels=64, kernel\_size=3, stride=2) ：输入32\*15\*15，输出64\*7\*7；

nn.BatchNorm2d(32) ：输入64\*7\*7，输出64\*7\*7；

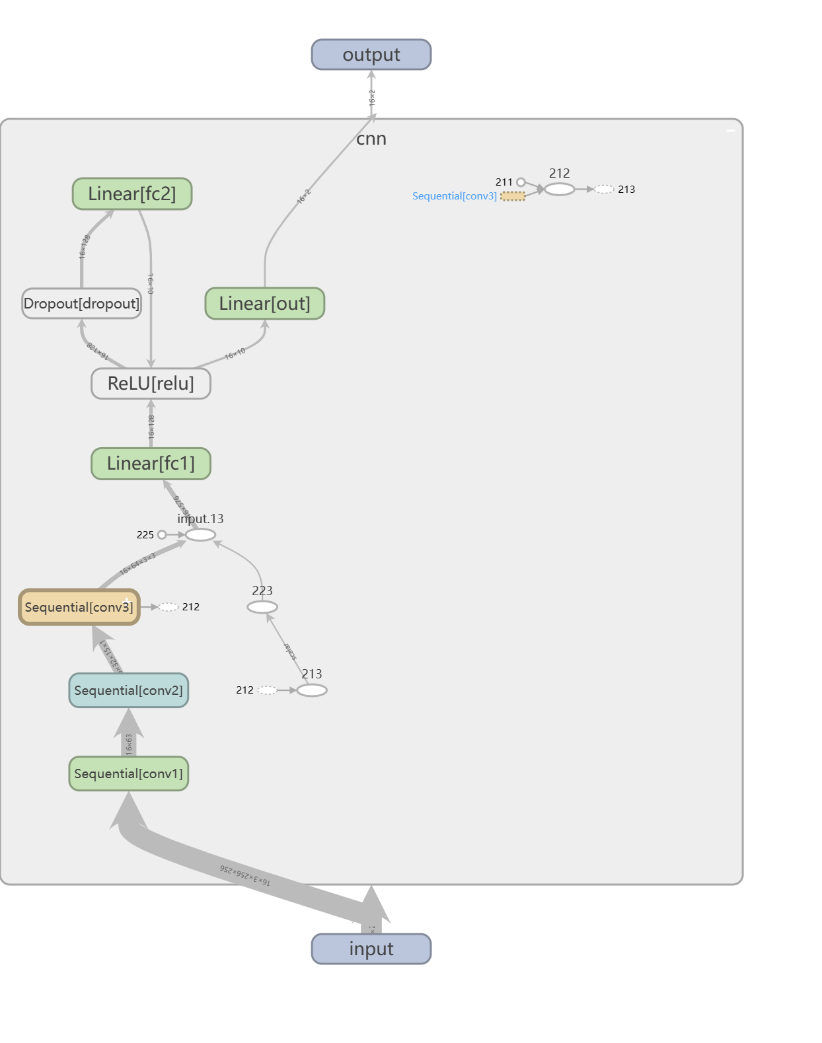
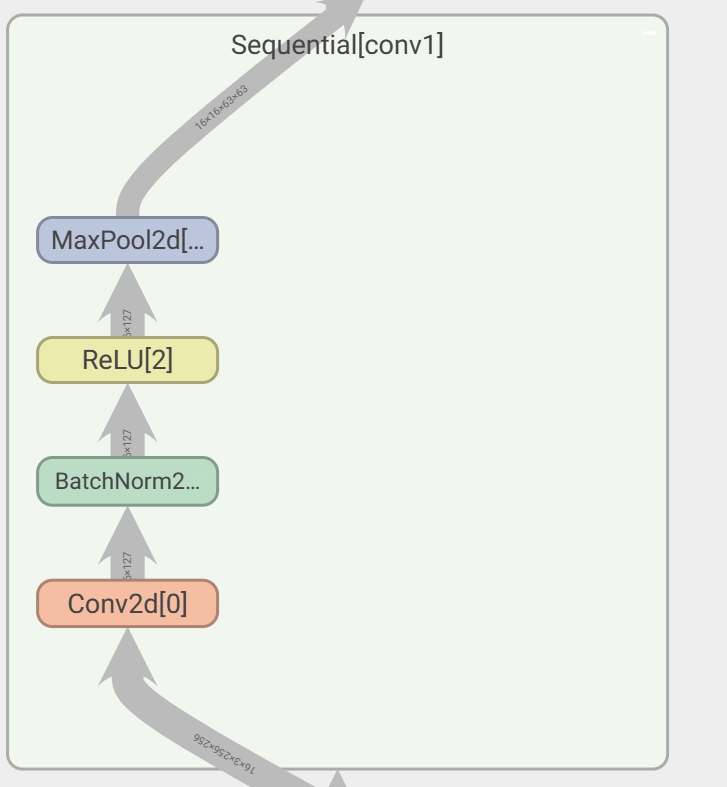
nn.ReLU()；

nn.MaxPool2d(kernel\_size=2) ：输入64\*7\*7，输出64\*3\*3

fc1：输入维度576，输出维度128，激活函数relu，加dropout防止过拟合；

fc2：输入维度128，输出维度10，激活函数relu；

fc3：输入维度10，输出维度2（二分类）。

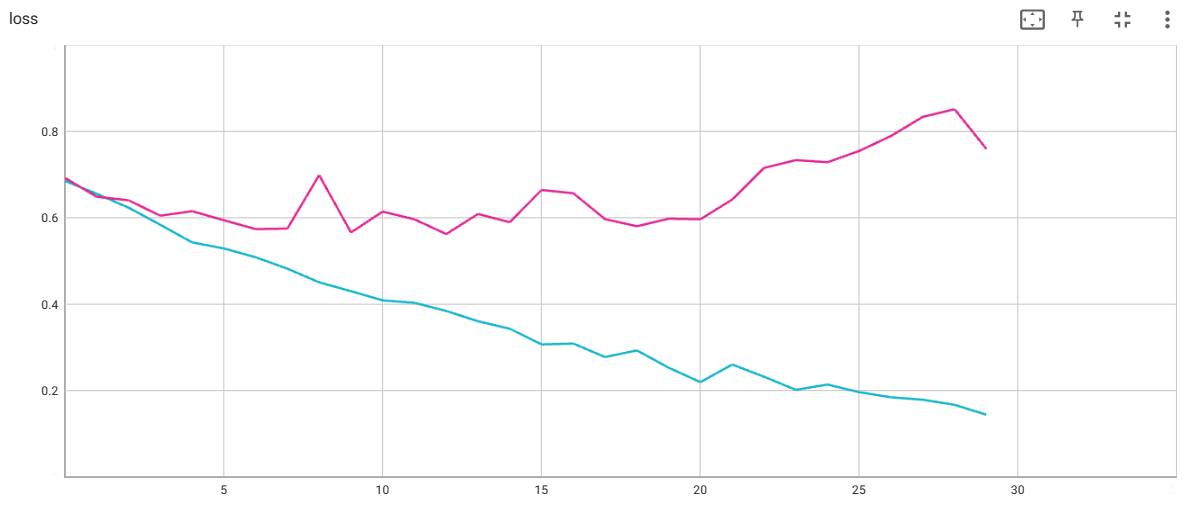
* + - 1. **实验结果**

随机抽取训练集、验证集和测试集，进行了5次实验。记录测试集的分类准确率结果如下表：

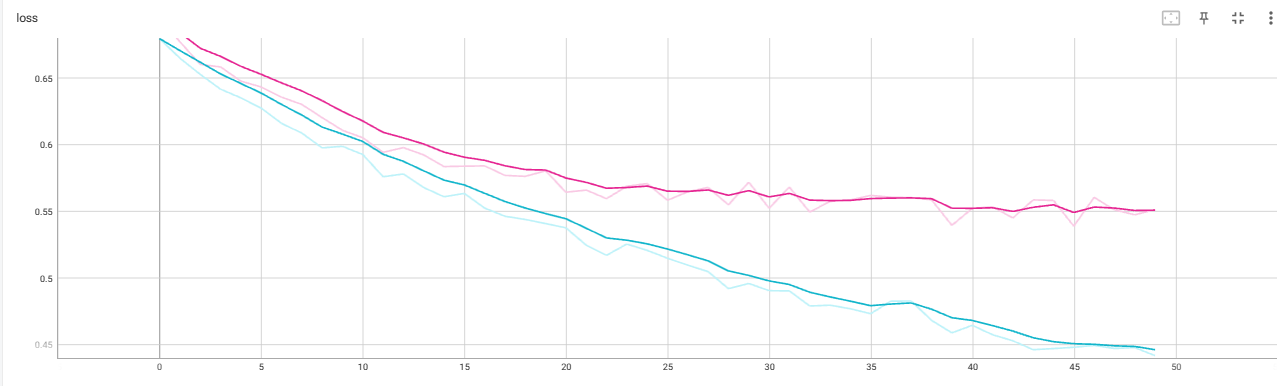
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | Ave |
| 准确率 | 89% | 86% | 82% | 84% | 82% | 84.6% |

* + - 1. **问题思考**

1. 在CNN的实验中，由于卷积神经网络减少了网络参数，减小了网络规模，所以将输入图像尺寸设为256\*256，提高图像分辨率。
2. 最初的模型中没有加入Dropout，图中蓝色为训练集的loss，红色为验证集的loss，可以看出，训练集的损失在不断下降，但是验证集损失很快就开始上升了，这说明网络很容易过拟合。在网络中加入Dropout，并对降低学习率后，过拟合现象得到改善。另外，加入Dropout后，网络的分类准确率也得到了提高，由原来的79%提升至84.6%。



没有加入Dropout的loss



加入Dropout的loss

## RNN模型

RNN是一种递归神经网络，可以用于处理序列数据，如文本、音频等。它的特点是可以将前面的信息传递到后面，一般适用于处理序列数据。

LSTM是一种特殊的RNN，相比于原始的RNN的隐层(hidden state)， LSTM增加了一个细胞状态(cell state)，可以解决RNN无法处理长距离的依赖的问题，一般用于处理序列数据，LSTM能够在更长的序列中有更好的表现。

* + - 1. **网络结构**

如图所示，网络使用1层LSTM网络，2层全连接层。

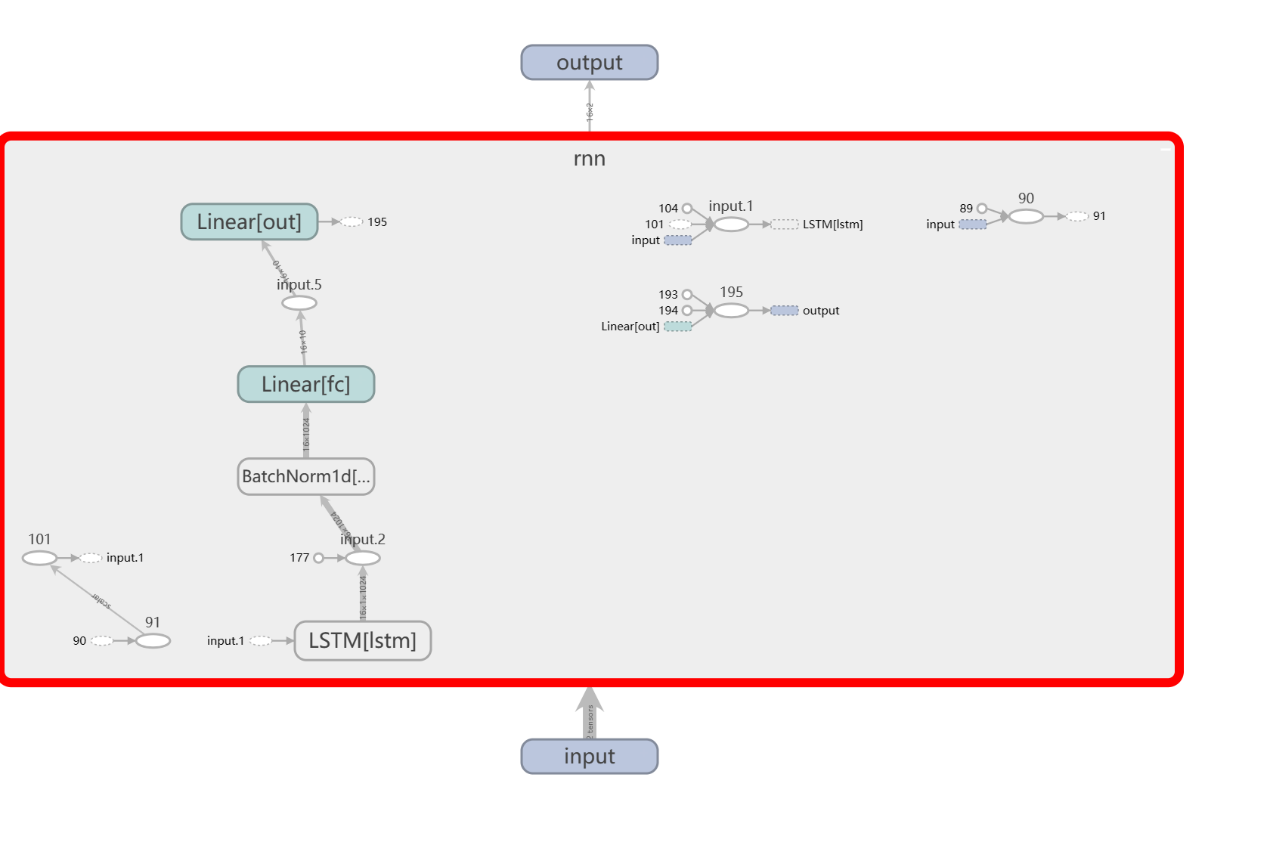
Pytorch中，使用函数torch.nn.LSTM（）实现LSTM网络，其中参数input\_size 应与图像尺寸一致（3\*64\*64），隐藏层状态的维数hidden\_size应与全连接层的输入维度一致（1024）。

LSTM：input\_size=3 \* 64 \* 64, hidden\_size=1024, num\_layers=1, batch\_first=True；

BN层：输入维度1024，输出维度1024；

fc1：输入维度1024，输出维度10，激活函数relu；

fc2（out）：输入维度10，输出维度2（二分类）。



* + - 1. **实验结果**

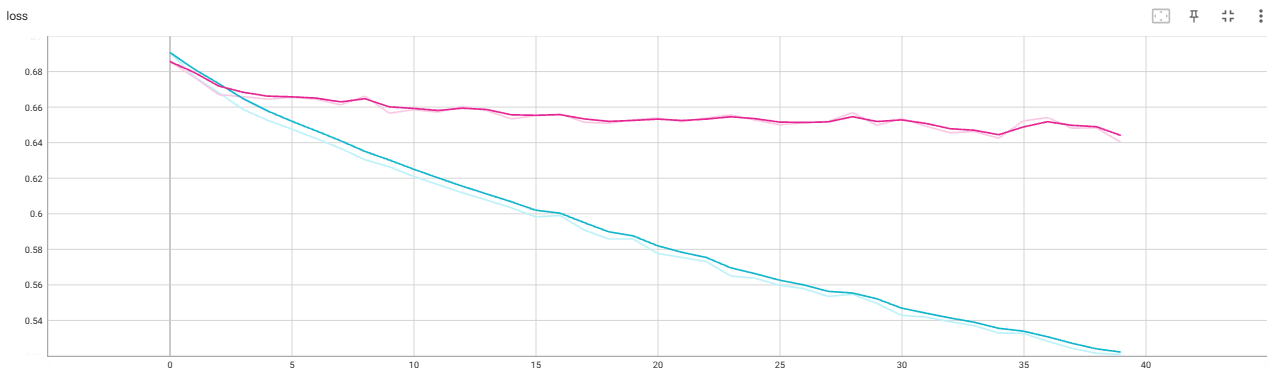
随机抽取训练集、验证集和测试集，进行了5次实验。记录测试集的分类准确率结果如下表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | Ave |
| 准确率 | 72% | 74% | 74% | 74% | 71% | 73% |

* + - 1. **问题思考**

1. 如前所述，RNN可以将前面的信息传递到后面，一般用于处理序列数据，所以在猫狗图像的分类问题中，该模型表现并不理想。
2. 与DNN和CNN一样，RNN网络同样存在过拟合问题，在训练过程中，训练集损失下降，但验证集损失基本不变，损失函数下降趋势如图所示，图中蓝色为训练集的loss，红色为验证集的loss。

这是因为：①数据集中的图像大都比较相似，而且训练数据并不充足（1500张），可以尝试现在规模更大的数据集上进行预训练，再在这个数据集上进行微调的方法解决该问题；②在实验中使用的网络结构简单，参数规模较小，不能充分理解图像信息，若要解决这个问题，可以尝试使用更精巧的网络结构（如ResNet等）。



loss

1. 比较DNN、CNN、RNN的分类准确率，可以得出在本实验中的分类性能CNN（84.6%） > DNN（76.8%） > RNN（73%），即CNN网络在猫狗分类任务重表现最好。