Pie-Lab 2025年暑期培训 Practice1

徐浩
Pie-Lab
北京理工大学计算机学院
北京市海淀区中关村南大街
526358612@qq.com

Abstract

本文是“任务一：图片分类”和“任务二：文本分类”的实验报告。

1 图片分类

1.1 各网络结构的原理解释

1.1.1 残差连接

在传统的神经网络中，当不断加深网络深度，会面临：梯度消失/爆炸、训练误差反而上升、训练难以收敛等问题。

引入残差连接可以解决这一问题，设输入为x，网络为F，则输出为：

$$F(x) + x$$

在反向传播时，对x求导，其导数就不会趋近于0，也就规避了梯度消失问题。

1.1.2 resnet34

resnet34是resnet系列中的一个模型，由多个残差块组成。其网络结构的文字表述见图1。

Layer Name	Output Size	Building Blocks	Stride
Conv1	112x112	7x7 conv, 64, stride 2	2
MaxPool	56x56	3x3 maxpool, stride 2	2
Conv2_x	56x56	[3x3, 64] x 3	1
Conv3_x	28x28	[3x3, 128] x 4	2
Conv4_x	14x14	[3x3, 256] x 6	2
Conv5_x	7x7	[3x3, 512] x 3	2
AvgPool + FC	1x1 → 1000		

Figure 1: resnet34_architecture

Conv1: 1 层

BasicBlock = 2 层卷积

Residual Blocks：

Conv2_x: 3 blocks × 2 = 6

Conv3_x: 4 blocks × 2 = 8

Layer Name	Output Size	Building Blocks	Stride	修改说明
Conv1	32x32	3x3 conv, 64, stride 1	1	替换原来的7x7+stride 2
MaxPool	❌移除			避免过度下采样
Layer1	32x32	[3x3, 64] x 3	1	
Layer2	16x16	[3x3, 128] x 4	2	下采样
Layer3	8x8	[3x3, 256] x 6	2	下采样
Layer4	4x4	[3x3, 512] x 3	2	下采样
AvgPool + FC	1x1 → 10			输出 CIFAR-10 分类

Figure 2: resnet34_architecture_cifar

Conv4_x: 6 blocks × 2 = 12

Conv5_x: 3 blocks × 2 = 6

fc = 1 层

合计：1 + 6 + 8 + 12 + 6 + 1 = 34 层

要想将resnet34用到CIFAR-10，需要对其网络进行微调，描述见图2。

1.2 模型结构设计与损失函数选择

resnet的架构在上文已经给出，这里仅说明本实验使用的模型：

mycnn：自己实现的resnet34去掉所有残差连接。

resnet34：封装的模型。

myresnet34：自己实现的模型。

1.3 实验设置

本实验的设置如下：

BATCH_SIZE = 128

NUM_EPOCHS = 20/30

LEARNING_RATE = 1e-3

NUM_CLASSES = 10

其他网络的参数见上文。

1.4 实验结果与可视化

本实验在租的服务器上训练，显卡为NVIDIA GeForce RTX 3090（做一晚上扣我10块，555）。

这里仅给出myresnet34的图表作为代表，损失见图3，准确率见图4。

本实验的所有输出如下所示：

model: mycnn

Epoch 1/30 | Train Loss: 1.9165, Train Acc: 0.2474 | Val Loss: 2.7610, Val Acc: 0.2405

Epoch 2/30 | Train Loss: 1.6152, Train Acc: 0.3693 | Val Loss: 1.6250, Val Acc: 0.3945

Epoch 3/30 | Train Loss: 1.4069, Train Acc: 0.4699 | Val Loss: 1.2601, Val Acc: 0.5259

Epoch 4/30 | Train Loss: 1.2242, Train Acc: 0.5503 | Val Loss: 1.3581, Val Acc: 0.5262

Epoch 5/30 | Train Loss: 1.0742, Train Acc: 0.6110 | Val Loss: 1.1760, Val Acc: 0.6024

Epoch 6/30 | Train Loss: 0.9571, Train Acc: 0.6576 | Val Loss: 1.0758, Val Acc: 0.6225

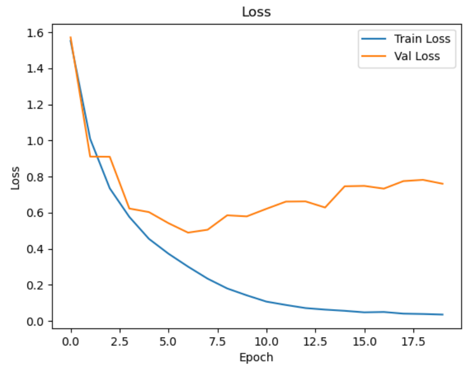

Figure 3: myresnet34_loss

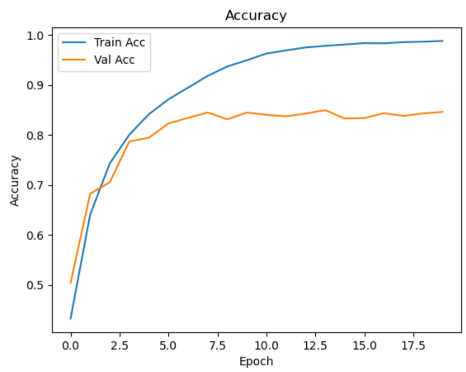

Figure 4: myresnet34_accuracy

Epoch 7/30 | Train Loss: 0.8627, Train Acc: 0.6920 | Val Loss: 1.0207, Val Acc: 0.6475

Epoch 8/30 | Train Loss: 0.7896, Train Acc: 0.7220 | Val Loss: 0.8764, Val Acc: 0.6953

Epoch 9/30 | Train Loss: 0.7172, Train Acc: 0.7486 | Val Loss: 1.3514, Val Acc: 0.5975

Epoch 10/30 | Train Loss: 0.6511, Train Acc: 0.7749 | Val Loss: 0.9171, Val Acc: 0.7046

Epoch 11/30 | Train Loss: 0.6058, Train Acc: 0.7910 | Val Loss: 0.9580, Val Acc: 0.6834

Epoch 12/30 | Train Loss: 0.5588, Train Acc: 0.8068 | Val Loss: 0.8354, Val Acc: 0.7277

Epoch 13/30 | Train Loss: 0.5159, Train Acc: 0.8216 | Val Loss: 0.7091, Val Acc: 0.7675

Epoch 14/30 | Train Loss: 0.4784, Train Acc: 0.8369 | Val Loss: 0.7872, Val Acc: 0.7423

Epoch 15/30 | Train Loss: 0.4419, Train Acc: 0.8490 | Val Loss: 0.6489, Val Acc: 0.7846

Epoch 16/30 | Train Loss: 0.4021, Train Acc: 0.8625 | Val Loss: 0.6489, Val Acc: 0.7885

Epoch 17/30 | Train Loss: 0.3763, Train Acc: 0.8718 | Val Loss: 0.6262, Val Acc: 0.7951

Epoch 18/30 | Train Loss: 0.3430, Train Acc: 0.8821 | Val Loss: 0.5970, Val Acc: 0.8177

Epoch 19/30 | Train Loss: 0.3252, Train Acc: 0.8877 | Val Loss: 0.5503, Val Acc: 0.8204

Epoch 20/30 | Train Loss: 0.2869, Train Acc: 0.9029 | Val Loss: 0.5693, Val Acc: 0.8236

Epoch 21/30 | Train Loss: 0.2687, Train Acc: 0.9080 | Val Loss: 0.8578, Val Acc: 0.7695

Epoch 22/30 | Train Loss: 0.2427, Train Acc: 0.9173 | Val Loss: 0.6308, Val Acc: 0.8274

Epoch 23/30 | Train Loss: 0.2240, Train Acc: 0.9231 | Val Loss: 0.6006, Val Acc: 0.8304

Epoch 24/30 | Train Loss: 0.1975, Train Acc: 0.9330 | Val Loss: 0.6797, Val Acc: 0.8217

Epoch 25/30 | Train Loss: 0.1882, Train Acc: 0.9351 | Val Loss: 0.5605, Val Acc: 0.8405

Epoch 26/30 | Train Loss: 0.1707, Train Acc: 0.9420 | Val Loss: 0.5996, Val Acc: 0.8312

Epoch 27/30 | Train Loss: 0.1560, Train Acc: 0.9469 | Val Loss: 0.7177, Val Acc: 0.8207

Epoch 28/30 | Train Loss: 0.1470, Train Acc: 0.9507 | Val Loss: 0.6621, Val Acc: 0.8278

Epoch 29/30 | Train Loss: 0.1374, Train Acc: 0.9525 | Val Loss: 0.6909, Val Acc: 0.8235

Epoch 30/30 | Train Loss: 0.1264, Train Acc: 0.9560 | Val Loss: 0.6056, Val Acc: 0.8442

Training completed in 868.96 seconds.

model: resnet34

Epoch 1/20 | Train Loss: 1.3207, Train Acc: 0.5204 | Val Loss: 1.3880, Val Acc: 0.5379

Epoch 2/20 | Train Loss: 0.8122, Train Acc: 0.7138 | Val Loss: 0.8336, Val Acc: 0.7197

Epoch 3/20 | Train Loss: 0.5992, Train Acc: 0.7907 | Val Loss: 0.6461, Val Acc: 0.7779

Epoch 4/20 | Train Loss: 0.4699, Train Acc: 0.8382 | Val Loss: 0.6697, Val Acc: 0.7773

Epoch 5/20 | Train Loss: 0.3778, Train Acc: 0.8693 | Val Loss: 0.6460, Val Acc: 0.7910

Epoch 6/20 | Train Loss: 0.2995, Train Acc: 0.8958 | Val Loss: 0.5204, Val Acc: 0.8320

Epoch 7/20 | Train Loss: 0.2355, Train Acc: 0.9160 | Val Loss: 0.5573, Val Acc: 0.8239

Epoch 8/20 | Train Loss: 0.1834, Train Acc: 0.9350 | Val Loss: 0.7652, Val Acc: 0.7915

Epoch 9/20 | Train Loss: 0.1498, Train Acc: 0.9475 | Val Loss: 0.6214, Val Acc: 0.8269

Epoch 10/20 | Train Loss: 0.1085, Train Acc: 0.9619 | Val Loss: 0.6762, Val Acc: 0.8225

Epoch 11/20 | Train Loss: 0.0880, Train Acc: 0.9684 | Val Loss: 0.7126, Val Acc: 0.8217

Epoch 12/20 | Train Loss: 0.0793, Train Acc: 0.9720 | Val Loss: 0.6548, Val Acc: 0.8369

Epoch 13/20 | Train Loss: 0.0641, Train Acc: 0.9777 | Val Loss: 0.7188, Val Acc: 0.8385

Epoch 14/20 | Train Loss: 0.0618, Train Acc: 0.9783 | Val Loss: 0.8034, Val Acc: 0.8291

Epoch 15/20 | Train Loss: 0.0549, Train Acc: 0.9807 | Val Loss: 0.7067, Val Acc: 0.8437

Epoch 16/20 | Train Loss: 0.0490, Train Acc: 0.9828 | Val Loss: 0.7375, Val Acc: 0.8355

Epoch 17/20 | Train Loss: 0.0451, Train Acc: 0.9837 | Val Loss: 0.7172, Val Acc: 0.8407

Epoch 18/20 | Train Loss: 0.0420, Train Acc: 0.9858 | Val Loss: 0.8694, Val Acc: 0.8316

Epoch 19/20 | Train Loss: 0.0352, Train Acc: 0.9877 | Val Loss: 0.7750, Val Acc: 0.8432

Epoch 20/20 | Train Loss: 0.0378, Train Acc: 0.9868 | Val Loss: 0.7775, Val Acc: 0.8459

Training completed in 611.57 seconds.

model: myresnet34

Epoch 1/20 | Train Loss: 1.5525, Train Acc: 0.4327 | Val Loss: 1.5719, Val Acc: 0.5041

Epoch 2/20 | Train Loss: 1.0093, Train Acc: 0.6407 | Val Loss: 0.9113, Val Acc: 0.6824

Epoch 3/20 | Train Loss: 0.7355, Train Acc: 0.7428 | Val Loss: 0.9104, Val Acc: 0.7052

Epoch 4/20 | Train Loss: 0.5771, Train Acc: 0.8003 | Val Loss: 0.6235, Val Acc: 0.7868

Epoch 5/20 | Train Loss: 0.4555, Train Acc: 0.8415 | Val Loss: 0.6038, Val Acc: 0.7943

Epoch 6/20 | Train Loss: 0.3731, Train Acc: 0.8711 | Val Loss: 0.5423, Val Acc: 0.8230

Epoch 7/20 | Train Loss: 0.3014, Train Acc: 0.8944 | Val Loss: 0.4899, Val Acc: 0.8340

Epoch 8/20 | Train Loss: 0.2347, Train Acc: 0.9179 | Val Loss: 0.5060, Val Acc: 0.8448

Epoch 9/20 | Train Loss: 0.1805, Train Acc: 0.9368 | Val Loss: 0.5859, Val Acc: 0.8312

Epoch 10/20 | Train Loss: 0.1424, Train Acc: 0.9493 | Val Loss: 0.5800, Val Acc: 0.8447

Epoch 11/20 | Train Loss: 0.1077, Train Acc: 0.9627 | Val Loss: 0.6218, Val Acc: 0.8401

Epoch 12/20 | Train Loss: 0.0889, Train Acc: 0.9690 | Val Loss: 0.6620, Val Acc: 0.8372

Epoch 13/20 | Train Loss: 0.0718, Train Acc: 0.9747 | Val Loss: 0.6630, Val Acc: 0.8426

Epoch 14/20 | Train Loss: 0.0632, Train Acc: 0.9781 | Val Loss: 0.6291, Val Acc: 0.8495

Epoch 15/20 | Train Loss: 0.0566, Train Acc: 0.9807 | Val Loss: 0.7467, Val Acc: 0.8328

Epoch 16/20 | Train Loss: 0.0481, Train Acc: 0.9836 | Val Loss: 0.7489, Val Acc: 0.8336

Epoch 17/20 | Train Loss: 0.0501, Train Acc: 0.9831 | Val Loss: 0.7337, Val Acc: 0.8432

Epoch 18/20 | Train Loss: 0.0410, Train Acc: 0.9855 | Val Loss: 0.7755, Val Acc: 0.8379

Epoch 19/20 | Train Loss: 0.0389, Train Acc: 0.9864 | Val Loss: 0.7825, Val Acc: 0.8429

Epoch 20/20 | Train Loss: 0.0357, Train Acc: 0.9879 | Val Loss: 0.7609, Val Acc: 0.8460

Training completed in 610.23 seconds.

1.5 总结与分析

对上述实验结果进行分析如下：

- **整体结果** 经过超参数调整之后，各个模型的结果大差不差，准确率基本都在0.84左右。
- **残差的作用** 实验中发现如果只是深层的cnn，例如上面的mycnn，其收敛速度非常慢；如果加上残差，变成resnet34，则收敛速度明显变快，这与前文提到的残差的作用是吻合的。

2 文本分类

2.1 各网络结构的原理解释

2.1.1 循环神经网络

循环神经网络（Recurrent Neural Network, RNN）是一种专门用于处理序列数据的神经网络结构。与前馈神经网络和卷积神经网络不同，RNN在网络中引入了时间维度上的循环连接，使得网络在每个时间步不仅接收当前输入，还能利用之前时刻的信息。这种特性使得RNN在处理时序相关的数据（如自然语言、时间序列信号、音频数据）时具有天然优势。

在标准的RNN结构中，给定一个长度为T的输入序列$(x_1, x_2, \dots, x_T)$，其中x_t是当前时刻t的输入向量。h_t表示t时刻的隐状态向量，它积累了$(x_1, x_2, \dots, x_{t-1})$的信息。则RNN计算$h_t$公式如下：

$$h_t = \sigma\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

其中W_xh和W_hh表示权重，b_h表示偏置，σ 表示激活函数。由该式子可以看出，h_t来源于之前的隐状态h_{t-1}和当前的输入x_t。随着t的推移，隐状态h_t就会存储历史信息，所以当前时刻的输出会受到之前输入的影响。

而t时刻的输出y_t则取决于隐状态h_t，公式如下：

$$y_t = w_{hy}h_t + b_y$$

上述所有权重和偏置通过学习得到。而在所有时间步中参数是共享的，也就是说，所有时刻使用相同的W_{xh}、W_{hh}、W_{hy}、b_h、b_y，这种设计大大减少了模型参数量，使得模型能够灵活地处理不同长度的输入序列，理论上RNN可以捕捉任意时间跨度的依赖关系。

2.1.2 LSTM

在实际训练过程中，标准RNN面临着严重的梯度消失（vanishing gradient）和梯度爆炸（exploding gradient）问题。具体而言，当进行反向传播时，误差信号在时间步上递归传播：

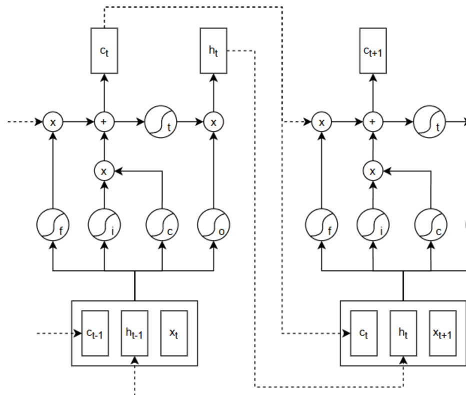

Figure 5: lstm_architecture

如果权重较小，梯度会逐步衰减，趋近于0，导致早期输入无法有效影响当前输出（梯度消失）；而如果权重较大，则可能出现梯度不断累积、数值发散的现象（梯度爆炸）。这使得标准RNN很难捕捉长距离的依赖关系。

为了克服上述问题，研究者提出了多种改进版本的RNN，其中最为经典的是长短期记忆网络（Long Short-Term Memory, LSTM），它在结构上引入了门控机制，能够有效缓解梯度问题，提高模型在长序列建模中的性能。

LSTM的架构如图5所示。

设t时刻输入为x_t，神经元为c_t，输出为y_t。设W表示权重，b表示偏置，σ和tanh表示激活函数。考虑t时刻到t+1时刻的变化如下。

遗忘门f的公式如下：

$$f_{t+1} = \sigma(W_f[c_t, h_t, x_{t+1}] + b_f)$$

输入门i的公式如下：

$$i_{t+1} = \sigma(W_i[c_t, h_t, x_{t+1}] + b_i)$$

候选神经元状态$\tilde{c}$的公式如下：

$$\tilde{c}_{t+1} = tanh(W_c[c_t, h_t, x_{t+1}] + b_c)$$

神经元c的公式如下：

$$c_{t+1} = f_{t+1} \odot c_t + i_{t+1} \odot \tilde{c}_{t+1}$$

输出门o的公式如下：

$$o_{t+1} = \sigma(W_o[c_t, h_t, x_{t+1}] + b_o)$$

隐状态h（也即输出）的公式如下：

$$h_{t+1} = o_{t+1} \odot tanh(c_{t+1})$$

2.2 模型结构设计与损失函数选择

lstm的架构与计算在上文已经给出，这里仅说明本实验使用的模型：

bilstm2：已封装，两层的双向lstm。

lstm4/8：已封装，4/8层的lstm。

res_lstm8：已封装，8层的lstm，每层lstm之后都带有残差连接。

my_lstm1：自己实现的1层lstm，训练速度极慢无比。

2.3 实验设置

本实验的设置如下：

NUM_LAYERS = ?，LSTM层数，视模型而定

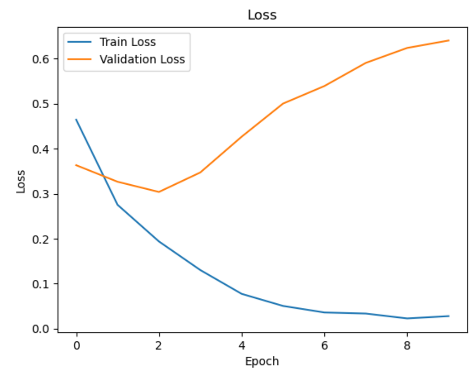

Figure 6: bilstm2_loss

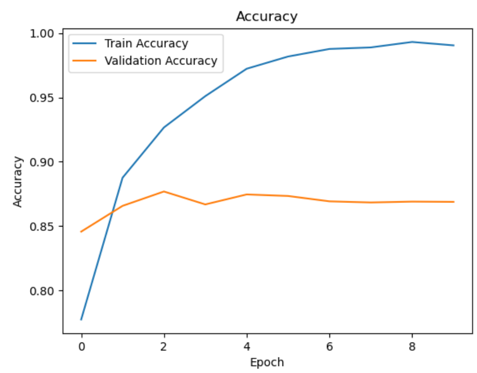

Figure 7: bilstm2_accuracy

MAX_LEN = 300

BATCH_SIZE = 64

EMBED_DIM = 200，词嵌入维度

HIDDEN_SIZE = 256，lstm隐藏层大小，每个词被表示为EMBED_DIM的向量，经过lstm计算变为HIDDEN_SIZE的向量

NUM_CLASSES = 2

NUM_EPOCHS = 10

LEARNING_RATE = 1e-3

DROPOUT = 0.5

MAX_VOCAB_SIZE = 20000，只留下最常用的词，其他一律当作unk

2.4 实验结果与可视化

本实验在租的服务器上训练，显卡为NVIDIA GeForce RTX 3090。

本实验的图表不如数据直观，这里仅给出BiLSTM2的图表作为代表，损失见图6，准确率见图7。

本实验的所有输出如下所示：

model: bilstm2（表示2层）

Epoch 1/10 | Train Loss: 0.4647, Train Acc: 0.7775 | Val Loss: 0.3635, Val Acc: 0.8458

Epoch 2/10 | Train Loss: 0.2755, Train Acc: 0.8876 | Val Loss: 0.3267, Val Acc: 0.8658

Epoch 3/10 | Train Loss: 0.1940, Train Acc: 0.9266 | Val Loss: 0.3040, Val Acc: 0.8769

Epoch 4/10 | Train Loss: 0.1305, Train Acc: 0.9510 | Val Loss: 0.3473, Val Acc: 0.8669

Epoch 5/10 | Train Loss: 0.0776, Train Acc: 0.9722 | Val Loss: 0.4268, Val Acc: 0.8746

Epoch 6/10 | Train Loss: 0.0507, Train Acc: 0.9817 | Val Loss: 0.5004, Val Acc: 0.8734

Epoch 7/10 | Train Loss: 0.0362, Train Acc: 0.9876 | Val Loss: 0.5394, Val Acc: 0.8693

Epoch 8/10 | Train Loss: 0.0338, Train Acc: 0.9888 | Val Loss: 0.5908, Val Acc: 0.8684

Epoch 9/10 | Train Loss: 0.0229, Train Acc: 0.9930 | Val Loss: 0.6240, Val Acc: 0.8691

Epoch 10/10 | Train Loss: 0.0281, Train Acc: 0.9904 | Val Loss: 0.6405, Val Acc: 0.8689

Training completed in 238.35 seconds.

model: lstm4

Epoch 1/10 | Train Loss: 0.5182, Train Acc: 0.7412 | Val Loss: 0.3940, Val Acc: 0.8284

Epoch 2/10 | Train Loss: 0.3702, Train Acc: 0.8395 | Val Loss: 0.3366, Val Acc: 0.8593

Epoch 3/10 | Train Loss: 0.3142, Train Acc: 0.8736 | Val Loss: 0.3630, Val Acc: 0.8401

Epoch 4/10 | Train Loss: 0.2227, Train Acc: 0.9135 | Val Loss: 0.3195, Val Acc: 0.8728

Epoch 5/10 | Train Loss: 0.1551, Train Acc: 0.9423 | Val Loss: 0.3167, Val Acc: 0.8802

Epoch 6/10 | Train Loss: 0.1099, Train Acc: 0.9610 | Val Loss: 0.3756, Val Acc: 0.8688

Epoch 7/10 | Train Loss: 0.0769, Train Acc: 0.9736 | Val Loss: 0.4680, Val Acc: 0.8652

Epoch 8/10 | Train Loss: 0.0504, Train Acc: 0.9824 | Val Loss: 0.5085, Val Acc: 0.8671

Epoch 9/10 | Train Loss: 0.0443, Train Acc: 0.9847 | Val Loss: 0.5582, Val Acc: 0.8680

Epoch 10/10 | Train Loss: 0.0336, Train Acc: 0.9886 | Val Loss: 0.6413, Val Acc: 0.8692

Training completed in 233.46 seconds.

model: lstm8

Epoch 1/10 | Train Loss: 0.6938, Train Acc: 0.4996 | Val Loss: 0.6932, Val Acc: 0.5000

Epoch 2/10 | Train Loss: 0.6935, Train Acc: 0.4966 | Val Loss: 0.6936, Val Acc: 0.5000

Epoch 3/10 | Train Loss: 0.6934, Train Acc: 0.5018 | Val Loss: 0.6931, Val Acc: 0.5000

Epoch 4/10 | Train Loss: 0.6933, Train Acc: 0.5044 | Val Loss: 0.6934, Val Acc: 0.5000

Epoch 5/10 | Train Loss: 0.6934, Train Acc: 0.4970 | Val Loss: 0.6932, Val Acc: 0.5000

Epoch 6/10 | Train Loss: 0.6933, Train Acc: 0.4998 | Val Loss: 0.6932, Val Acc: 0.5000

Epoch 7/10 | Train Loss: 0.6933, Train Acc: 0.4985 | Val Loss: 0.6933, Val Acc: 0.5000

Epoch 8/10 | Train Loss: 0.6933, Train Acc: 0.4956 | Val Loss: 0.6932, Val Acc: 0.5000

Epoch 9/10 | Train Loss: 0.6932, Train Acc: 0.5035 | Val Loss: 0.6935, Val Acc: 0.5000

Epoch 10/10 | Train Loss: 0.6933, Train Acc: 0.4999 | Val Loss: 0.6932, Val Acc: 0.5000

Training completed in 366.85 seconds.

model: res_lstm8

Epoch 1/10 | Train Loss: 0.5345, Train Acc: 0.7207 | Val Loss: 0.4372, Val Acc: 0.8200

Epoch 2/10 | Train Loss: 0.3075, Train Acc: 0.8736 | Val Loss: 0.3676, Val Acc: 0.8512

Epoch 3/10 | Train Loss: 0.2194, Train Acc: 0.9141 | Val Loss: 0.3388, Val Acc: 0.8818

Epoch 4/10 | Train Loss: 0.1602, Train Acc: 0.9398 | Val Loss: 0.3708, Val Acc: 0.8755

Epoch 5/10 | Train Loss: 0.1027, Train Acc: 0.9620 | Val Loss: 0.4595, Val Acc: 0.8745
Epoch 6/10 | Train Loss: 0.0761, Train Acc: 0.9719 | Val Loss: 0.5454, Val Acc: 0.8675
Epoch 7/10 | Train Loss: 0.0515, Train Acc: 0.9821 | Val Loss: 0.6953, Val Acc: 0.8627
Epoch 8/10 | Train Loss: 0.0430, Train Acc: 0.9850 | Val Loss: 0.6839, Val Acc: 0.8715
Epoch 9/10 | Train Loss: 0.0348, Train Acc: 0.9883 | Val Loss: 0.7477, Val Acc: 0.8738
Epoch 10/10 | Train Loss: 0.0314, Train Acc: 0.9895 | Val Loss: 0.7476, Val Acc: 0.8744
Training completed in 437.83 seconds.
model: my_lstm1（自己实现的lstm，一层）
Epoch 1/10 | Train Loss: 0.5083, Train Acc: 0.7460 | Val Loss: 0.3495, Val Acc: 0.8486
Epoch 2/10 | Train Loss: 0.2873, Train Acc: 0.8843 | Val Loss: 0.3002, Val Acc: 0.8787
Epoch 3/10 | Train Loss: 0.2042, Train Acc: 0.9217 | Val Loss: 0.3260, Val Acc: 0.8641
Epoch 4/10 | Train Loss: 0.1408, Train Acc: 0.9477 | Val Loss: 0.2915, Val Acc: 0.8805
Epoch 5/10 | Train Loss: 0.0905, Train Acc: 0.9680 | Val Loss: 0.3594, Val Acc: 0.8839
Epoch 6/10 | Train Loss: 0.0581, Train Acc: 0.9800 | Val Loss: 0.4855, Val Acc: 0.8791
Epoch 7/10 | Train Loss: 0.0405, Train Acc: 0.9875 | Val Loss: 0.4711, Val Acc: 0.8778
Epoch 8/10 | Train Loss: 0.0317, Train Acc: 0.9898 | Val Loss: 0.4913, Val Acc: 0.8681
Epoch 9/10 | Train Loss: 0.0218, Train Acc: 0.9932 | Val Loss: 0.5798, Val Acc: 0.8755
Epoch 10/10 | Train Loss: 0.0245, Train Acc: 0.9922 | Val Loss: 0.7133, Val Acc: 0.8722
Training completed in 904.25 seconds.

2.5 总结与分析

对上述实验结果进行分析如下：

- **整体结果** 经过超参数调整之后，各个模型的结果大差不差，准确率基本都在0.88左右。
- **残差的作用** 实验中发现一个特殊现象，lstm设置为4层，还能训得动，结果较为正常；lstm设置为8层，完全训不动，几乎没变化；加入残差连接之后，就能回归正常。这是因为，lstm堆叠较深时存在梯度消失，加入残差连接可以避免这个问题。
- **自定义lstm运行太慢** 实验中发现自己实现的lstm，虽然只有1层，但是训练速度极慢无比，其训练时间是加了残差连接的封装好的8层lstm的两倍多，暂时不清楚原因。