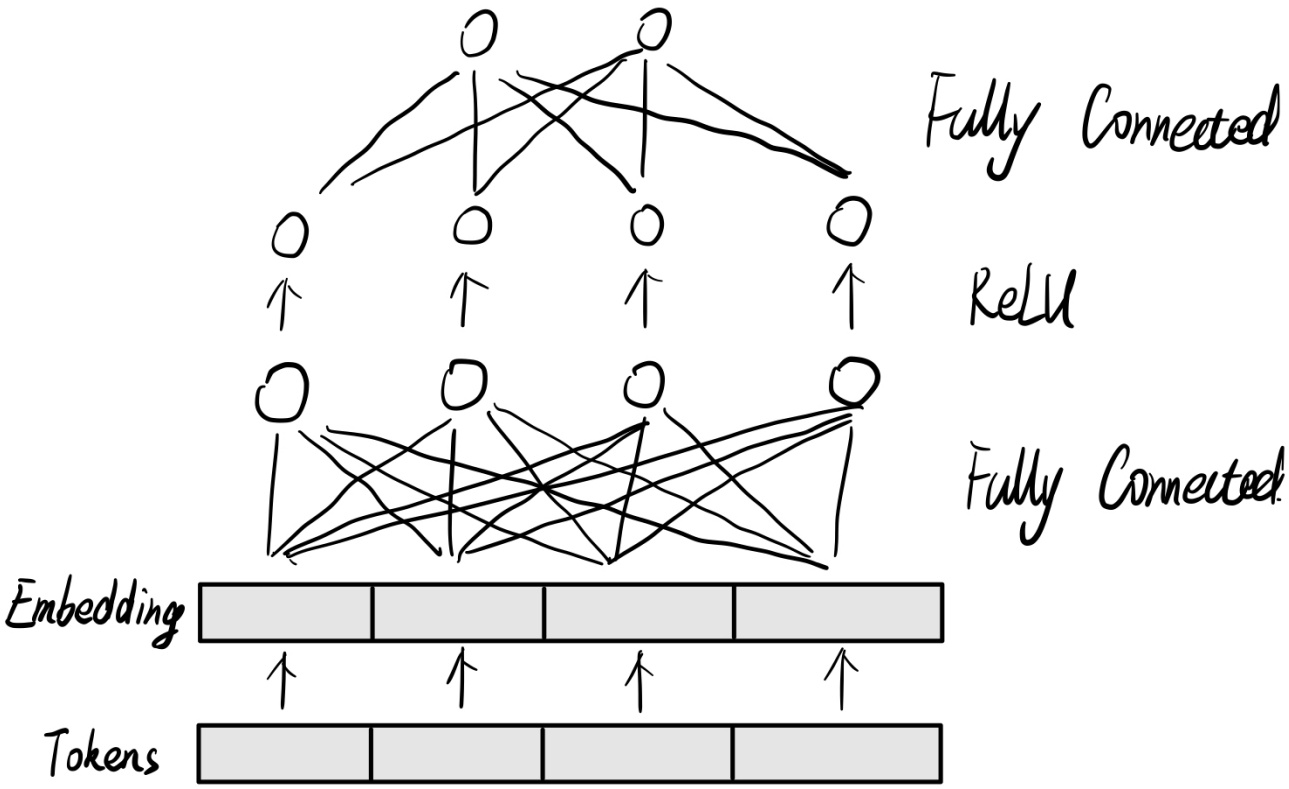
Report

模型结构图

共实现了以下模型：两个MLP模型，一个CNN模型，两个RNN模型和一个Transformer模型。

MLP结构图

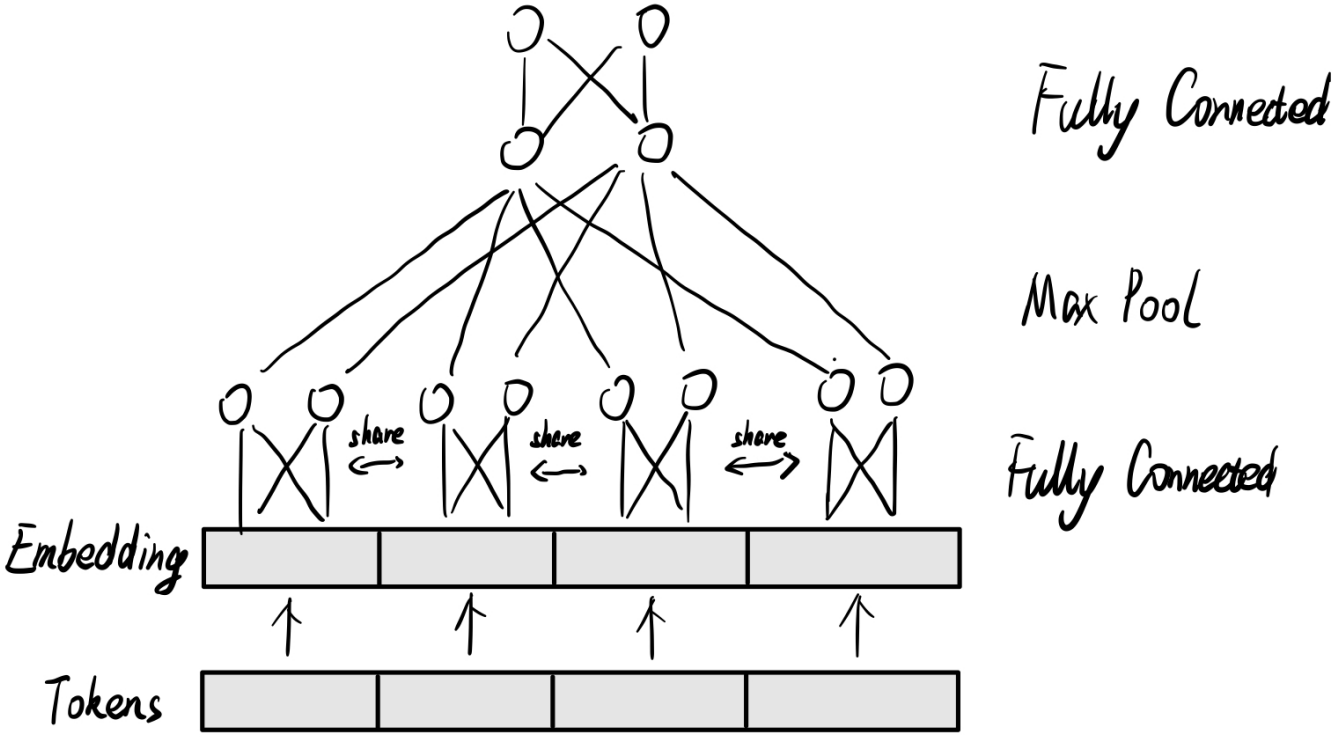
第一个MLP模型的结构图如下：



将所有token转化为词向量后，将所有 $max\_len \times embed\_dim$ （其中 max_len 为文本中提取的最多token数）个元素通过一个全连接层，再通过一个dropout和一个全连接层，取 argmax 便可得到结果。
其中第一个全连接层将所有表征转化为一个一维向量，用于提取特征。
最后一个全连接层使用每个特征来进行分类。

MLP2结构图

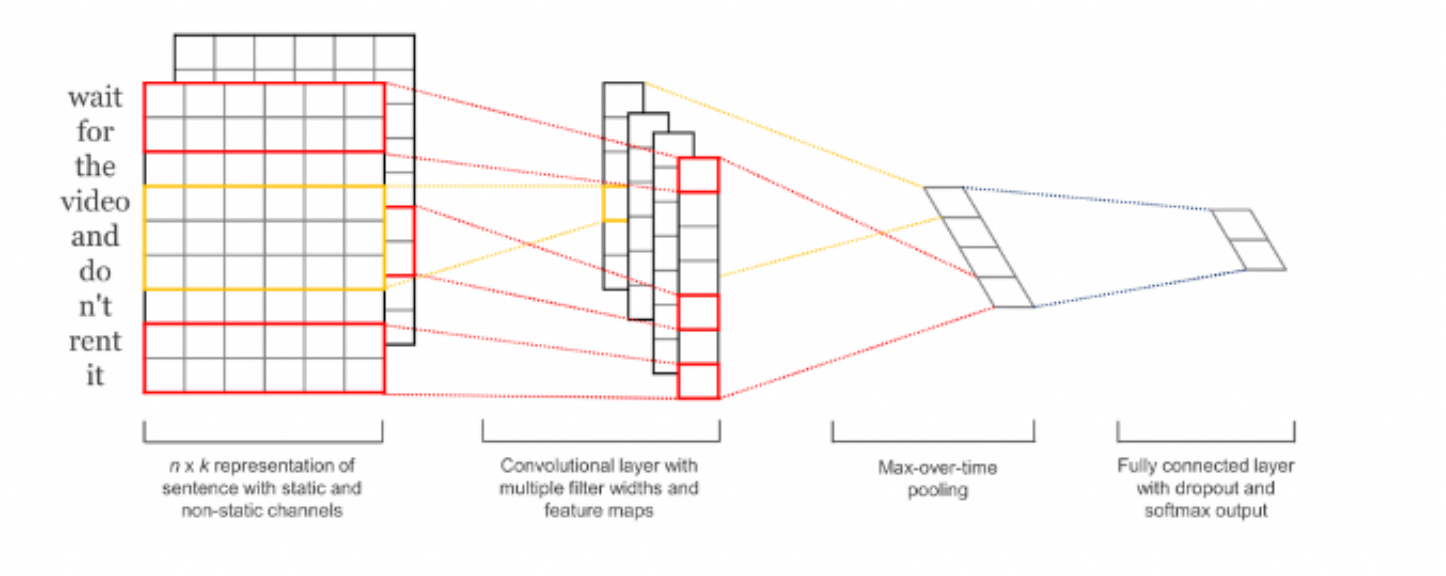
第二个MLP模型的结构图如下：



将所有的token转化为词向量后，每一个token的embedding通过一个全连接层并dropout，再通过一个max pool对 max_len 个元素取最大值，得到一个一维向量，再通过一个全连接层，取 argmax 便可得到结果。
其中第一个全连接层的权重是共享的，即对于每一个token，都使用相同的权重进行计算，用于提取特征。
Max pool层即用于对每一类特征，提取一个句子中所有token中最重要的特征。
最后一个全连接层使用每个最重要的特征来进行分类。

CNN结构图

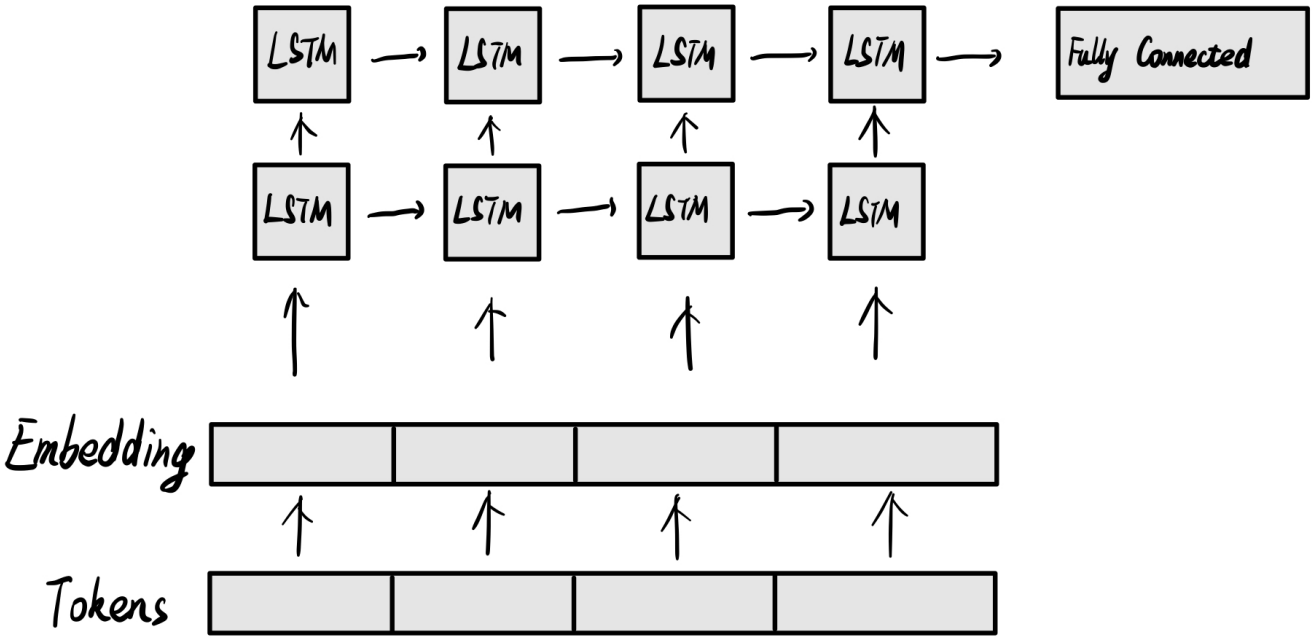
根据参考文献有以下结构图：



有三个不同的卷积核大小：$3, 4, 5 \times embed\_dim$（此处卷积是直接对连续几个token的词向量形成的矩阵做的），每个卷积核的个数为 10 个，首先将整句的每个token转化为词向量后，得到 $len \times embed\_dim$ 的表征，通过所有卷积核之后，得到每个卷积核对应的向量，然后通过一个max pool，得到一个一维向量，再通过一个dropout和一个全连接层，取 argmax 便可得到结果。
其中卷积核的作用是提取局部特征，max pool的作用是提取最重要的特征。
最后一个全连接层使用每个最重要的特征来进行分类。

RNN结构图

第一个RNN模型的结构图如下：

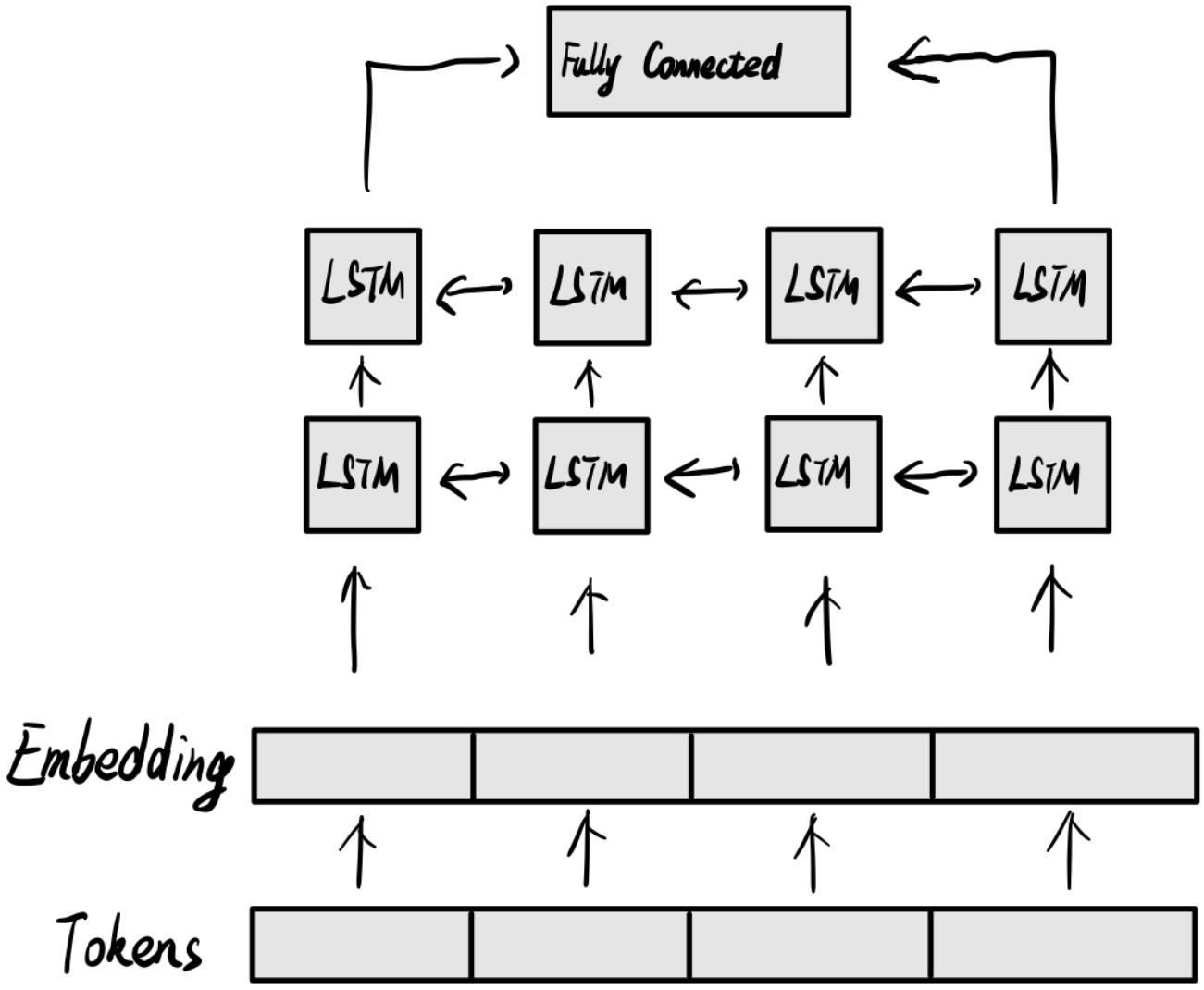


为一个双层单向LSTM，每一层的输出通过一个dropout层，最后一个输出通过一个全连接层，取

argmax 便可得到结果。

RNN2结构图

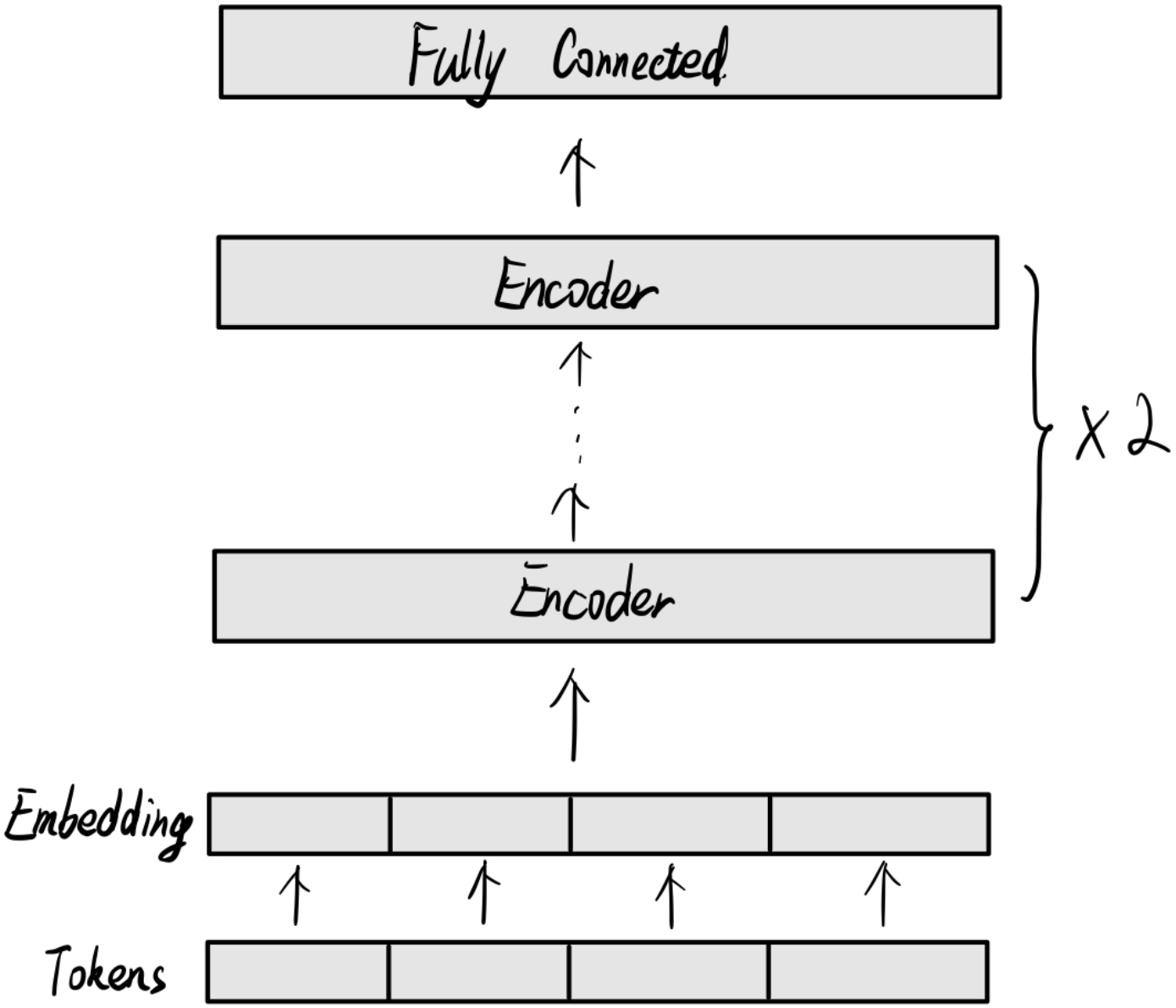
第二个RNN模型的结构图如下：



为一个双层双向LSTM，每一层的输出通过一个dropout层，最后将双向LSTM的输出拼接，再通过一个全连接层，取 argmax 便可得到结果。

Transformer结构图

Transformer模型的结构图如下：



由于为一个分类任务，所以只使用了Transformer的Encoder部分，即将所有token转化为词向量后，通过两层的Transformer Encoder，再通过一个全连接层，取 argmax 便可得到结果。

流程分析

在主程序 `main.py` 中，首先调用 `load_text` 函数，该函数会从训练集、验证集、测试集中读取文本与标签，并使用预训练好的 `Word2Vec` 将单个文本token转化为向量（为了加速只取前40个token，不足的使用 0 向量补足，如果token不在 `Word2Vec` 中则直接丢弃），得到 `{train, val, test}_{inputs, labels}` 六个Tensor。

然后根据命令行参数使用 `models.py` 中选定的模型并初始化，如果可以使用GPU则转移。

接下来调用 `train` 函数，经过 `epochs` 轮数的训练，每一轮使用 `Adam` 优化器对模型进行优

化，并且输出每一轮验证集的 `loss` 和 `accuracy` 。

训练结束后，调用 `evaluate` 函数使用模型在测试集上进行评估，给出模型输出的 `accuracy, precision, recall, f1` 等评价指标。

实验结果

以下为对于每个模型选择参数
$batch\_size = 512, epochs = 500, lr = 0.0001, dropout = 0.5$
后较优的结果：

```
MLP
Accuracy: 0.7073, Precision: 0.8264, Recall: 0.5348, F1: 0.6494
MLP2
Accuracy: 0.7832, Precision: 0.8092, Recall: 0.7487, F1: 0.7778
CNN
Accuracy: 0.7669, Precision: 0.8301, Recall: 0.6791, F1: 0.7471
RNN
Accuracy: 0.7642, Precision: 0.8247, Recall: 0.6791, F1: 0.7449
RNN2
Accuracy: 0.7859, Precision: 0.8253, Recall: 0.7326, F1: 0.7762
Transformer
Accuracy: 0.7561, Precision: 0.8129, Recall: 0.6738, F1: 0.7368
```

不同参数效果比较及分析

以下以RNN模型为例，比较不同的超参数选择对于最终效果的影响

不同的learning rate

以下控制 $batch\_size = 512, epochs = 500, dropout = 0.5$ 固定

lr	Accuracy	Precision	Recall	F1
0.001	0.7317	0.7500	0.7059	0.7273
0.0001	0.7561	0.7904	0.7059	0.7458
0.00001	0.7425	0.7949	0.6631	0.7230

可以发现在一定条件下， learning rate的减小有助于准确率等指标的提升。推测是因为learning

rate减小时，可以使得梯度下降算法更加稳定，从而提高训练效果。但是当learning rate过小时，可能会导致模型收敛速度过慢，在固定epochs时未较好收敛，从而降低训练效果。

不同的batch size

以下控制 $epochs = 500, lr = 0.0001, dropout = 0.5$ 固定

$batch\_size$	Accuracy	Precision	Recall	F1
16	0.5501	0.9565	0.1176	0.2095
32	0.6883	0.8529	0.4652	0.6021
64	0.7453	0.8298	0.6257	0.7134
128	0.7154	0.7929	0.5936	0.6789
256	0.7263	0.7756	0.6471	0.7055
512	0.7561	0.7904	0.7059	0.7458
1024	0.7669	0.7988	0.7219	0.7584
2048	0.7642	0.8049	0.7059	0.7521

可以发现在一定条件下，batch size的增大有助于准确率等指标的提升。推测是因为batch size的增大可以在每一步迭代时减少噪声，使得梯度下降算法更加稳定，从而提高训练效果。但是当batch size过大时，噪声过低，可能会导致模型难以跳出局部最优解，从而降低训练效果。

不同的epochs

以下控制 $batch\_size = 512, lr = 0.0001, dropout = 0.5$ 固定

$epochs$	Accuracy	Precision	Recall	F1
10	0.6883	0.6579	0.8021	0.7229
50	0.7100	0.7532	0.6364	0.6899
100	0.7317	0.7821	0.6524	0.7114
200	0.7344	0.7871	0.6524	0.7135
500	0.7561	0.7904	0.7059	0.7458
1000	0.7371	0.7711	0.6845	0.7252

可以发现在一定条件下，epochs的增大有助于准确率等指标的提升。推测是因为epochs的增大可以使得模型更好的拟合数据，从而提高训练效果。但是当epochs过大时，可能会导致模型过拟合，从而降低训练效果。

不同的dropout

以下控制 $batch\_size = 512, epochs = 500, lr = 0.0001$ 固定

$dropout$	Accuracy	Precision	Recall	F1
0.0	0.6829	0.7244	0.6043	0.6589
0.3	0.7561	0.7870	0.7112	0.7472
0.5	0.7561	0.7904	0.7059	0.7458
0.7	0.7588	0.7917	0.7112	0.7493

可以发现在一定条件下，dropout的增大有助于准确率等指标的提升。推测是此时模型已经有些过拟合，dropout的增大可以使得模型更好的泛化，从而提高训练效果。但是当dropout过大时，也有可能会导致模型欠拟合，从而降低训练效果。

Baseline效果差异

以MLP为基线模型，我们可以发现在相同的超参数设置下，其余模型的表现均优于MLP模型，其中RNN2模型的表现最好。
推测是因为MLP模型的局限性，无法提取局部特征而且无法很好地处理序列数据。
相较于原始的MLP，推测MLP2中加入了MaxPool池化层用于提取最明显的特征，从而提升了效果。
推测RNN，RNN2模型可以较好地处理序列信息，从而提高训练效果。
推测CNN模型可以较好地提取局部信息，从而提高训练效果。
推测Transformer模型可以提取token之间相互影响的信息，提高了token包含的特征信息，从而提高训练效果。

问题思考

1. 实验训练什么时候停止是最合适的？简要陈述你的实现方式，并试分析固定迭代次数与通过验证集调整等方法的优缺点。

目前实现的方式是固定迭代次数，通过 `epochs` 参数控制。
固定迭代次数的优点是简单，而且可以预期训练时间，缺点是可能会导致模型过拟合或者欠拟合。通过验证集调整的优点是可以减轻过拟合之类的问题，缺点是可能会导致训练时间不确定。

2. 实验参数的初始化是怎么做的？不同的方法适合哪些地方？（现有的初始化方法为零均值初始化，高斯分布初始化，正交初始化等）

   目前本实验的模型参数使用Pytorch默认的Kaiming初始化对全连接层的权重进行初始化，对于bias使用零均值分布初始化，这种初始化方法适合于ReLU激活函数，因为它可以使得每一层的输出的方差尽可能保持一致，从而使得每一层的梯度分布相对均匀，避免梯度消失或者梯度爆炸。
   对于卷积层的权重同样使用Kaiming初始化，bias使用零均值初始化。
   对于RNN层的权重使用零均值初始化。

   如果使用零均值初始化，可能会导致梯度消失或者梯度爆炸，因为每一层的梯度是在后一层的梯度的基础上计算的，如果初始值过大或者过小，梯度累计可能会导致梯度爆炸或者梯度消失。

   如果使用高斯分布初始化，可以有效避免梯度消失或梯度爆炸，但是容易导致过拟合，需要通过正则化等方法来避免。

   如果使用正交初始化，可以有效减少过拟合，但是训练时间可能会增加。

3. 过拟合是深度学习常见的问题，有什么方法可以方式训练过程陷入过拟合。

   以下为几种可以避免过拟合的方法：

   - Dropout：在训练过程中，随机的将一部分神经元的输出置为0，可以有效减少过拟合。
   - 增大数据集：增大数据集可以有效减少过拟合。
   - 正则化：L1正则化和L2正则化可以有效减少过拟合。
   - 早停：在验证集上的loss不再下降时停止训练。

4. 试分析CNN，RNN，全连接神经网络（MLP）三者的优缺点。

   - CNN 的优点是可以提取局部的特征，可以减少参数的数量，还可以进行并行计算，缺点是可能会丢失全局信息。
   - RNN 的优点是可以较好的处理序列类型的数据，可以保留之前的信息，而且对输入的长度没有严格的要求，缺点是可能更容易出现梯度消失和梯度爆炸，而且由于需要一步一步的计算，无法进行并行化计算。
   - MLP 的优点是较为简单，同样可以进行并行计算，缺点是需要较多的参数，而且可能会出现过拟合。

心得体会

在本次实验中，我使用PyTorch框架实现了多类模型，包括MLP，CNN，RNN，Transformer等，并进行训练与评估。通过调整超参数，发现不同的超参数对于模型的训练效果有着不同的影响。通过本次实验，我对于深度学习模型的训绨有了更深的理解，对于超参数的选择也有了更深的认识。