实验 4 关系抽取实验

一、基于卷积神经网络的关系抽取

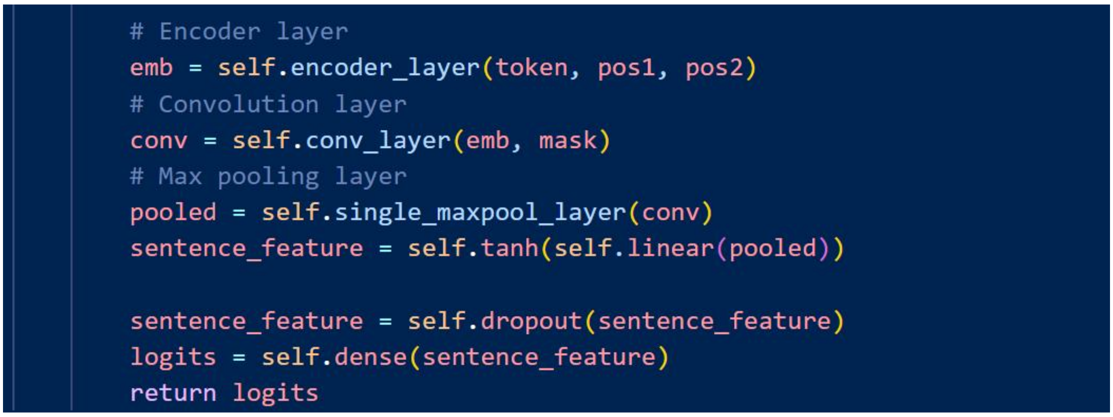
（一）实验内容

- 根据论文补充 ./CNN/model.py 的 forward 部分(101 行)，跑通训练代码；
- 运行 run.py 训练模型并预测结果；
- 对预测结果进行评估，要求复现结果 F1 值大于 82.7(原论文)*0.9；
- 进行消融实验，尝试去除 PF(Position features)重复实验，并报告结果；
- 提交要求：在实验报告中报告复现的 F1 值，同时报告消融实验的 F1 值，提交代码文件./CNN/model.py。

（二）实验过程

1. 根据论文补充 ./CNN/model.py

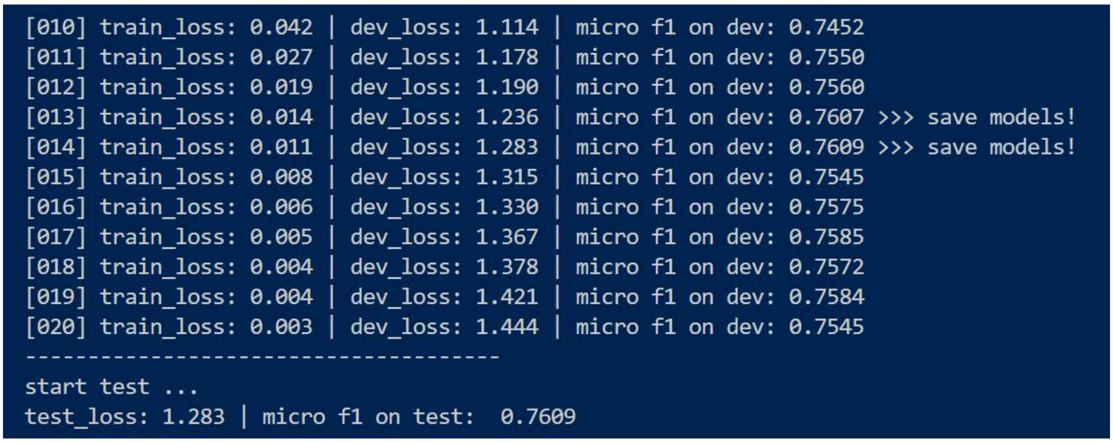
补充代码如下。



```
        # Encoder layer
        emb = self.encoder_layer(token, pos1, pos2)
        # Convolution layer
        conv = self.conv_layer(emb, mask)
        # Max pooling layer
        pooled = self.single_maxpool_layer(conv)
        sentence_feature = self.tanh(self.linear(pooled))

        sentence_feature = self.dropout(sentence_feature)
        logits = self.dense(sentence_feature)
        return logits
```

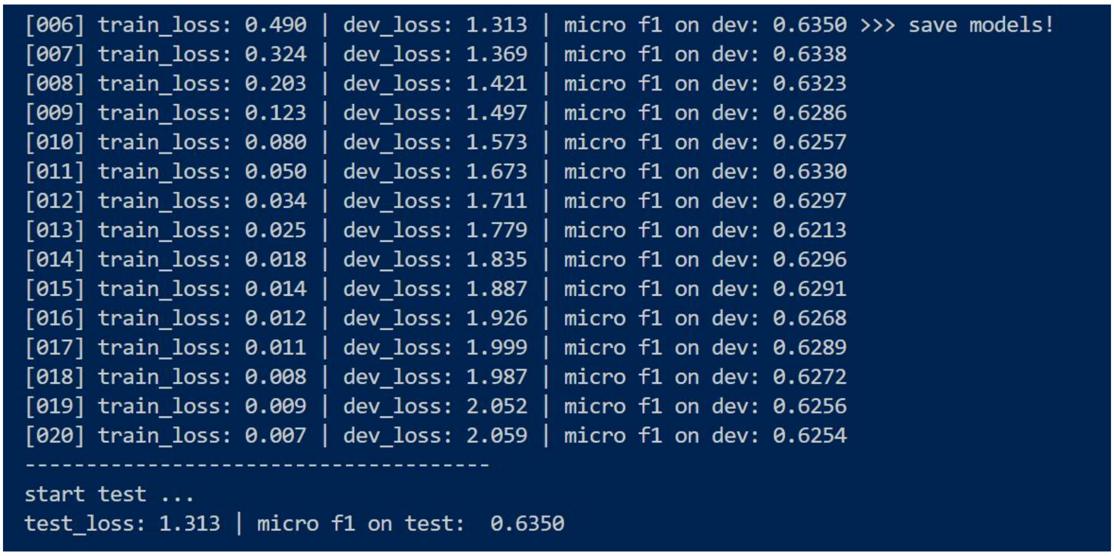
2. 训练模型并评估预测结果



```
[010] train_loss: 0.042 | dev_loss: 1.114 | micro f1 on dev: 0.7452
[011] train_loss: 0.027 | dev_loss: 1.178 | micro f1 on dev: 0.7550
[012] train_loss: 0.019 | dev_loss: 1.190 | micro f1 on dev: 0.7560
[013] train_loss: 0.014 | dev_loss: 1.236 | micro f1 on dev: 0.7607 >>> save models!
[014] train_loss: 0.011 | dev_loss: 1.283 | micro f1 on dev: 0.7609 >>> save models!
[015] train_loss: 0.008 | dev_loss: 1.315 | micro f1 on dev: 0.7545
[016] train_loss: 0.006 | dev_loss: 1.330 | micro f1 on dev: 0.7575
[017] train_loss: 0.005 | dev_loss: 1.367 | micro f1 on dev: 0.7585
[018] train_loss: 0.004 | dev_loss: 1.378 | micro f1 on dev: 0.7572
[019] train_loss: 0.004 | dev_loss: 1.421 | micro f1 on dev: 0.7584
[020] train_loss: 0.003 | dev_loss: 1.444 | micro f1 on dev: 0.7545
--------------------------------------
start test ...
test_loss: 1.283 | micro f1 on test:  0.7609
```

3.　消融实验

去除 PF(Position features)后，训练结果如下：



```
[006] train_loss: 0.490 | dev_loss: 1.313 | micro f1 on dev: 0.6350 >>> save models!
[007] train_loss: 0.324 | dev_loss: 1.369 | micro f1 on dev: 0.6338
[008] train_loss: 0.203 | dev_loss: 1.421 | micro f1 on dev: 0.6323
[009] train_loss: 0.123 | dev_loss: 1.497 | micro f1 on dev: 0.6286
[010] train_loss: 0.080 | dev_loss: 1.573 | micro f1 on dev: 0.6257
[011] train_loss: 0.050 | dev_loss: 1.673 | micro f1 on dev: 0.6330
[012] train_loss: 0.034 | dev_loss: 1.711 | micro f1 on dev: 0.6297
[013] train_loss: 0.025 | dev_loss: 1.779 | micro f1 on dev: 0.6213
[014] train_loss: 0.018 | dev_loss: 1.835 | micro f1 on dev: 0.6296
[015] train_loss: 0.014 | dev_loss: 1.887 | micro f1 on dev: 0.6291
[016] train_loss: 0.012 | dev_loss: 1.926 | micro f1 on dev: 0.6268
[017] train_loss: 0.011 | dev_loss: 1.999 | micro f1 on dev: 0.6289
[018] train_loss: 0.008 | dev_loss: 1.987 | micro f1 on dev: 0.6272
[019] train_loss: 0.009 | dev_loss: 2.052 | micro f1 on dev: 0.6256
[020] train_loss: 0.007 | dev_loss: 2.059 | micro f1 on dev: 0.6254
--------------------------------------
start test ...
test_loss: 1.313 | micro f1 on test:  0.6350
```

二、远程监督关系抽取

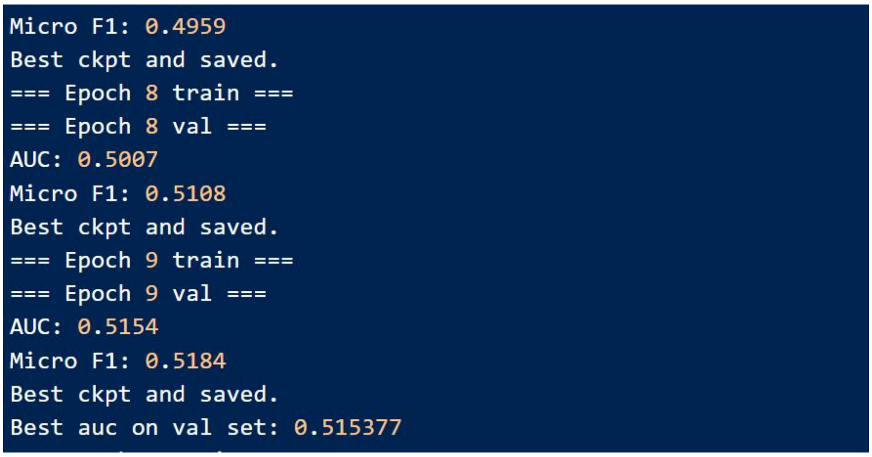
（一）实验内容

- 安装开源库 OpenNER，并运行 ./benchmark/download_nyt10m.sh 下载相关数据集，将 train.sh 的 --dataset 改为对应数据集；
- 使用 cnn 作为编码器，设置 --aggr 为 att ，也就是使用句子级注意力，训练以及推理，报告 AUC 以及 F1 值；
- 使用 cnn 作为编码器，设置 --aggr 为 avg ，也就是使用句子平均向量，训练以及推理，报告 AUC 以及 F1 值；
- (选做)使用 pcnn 作为编码器，设置 --aggr 为 att ，也就是使用句子级注意力，训练以及推理，报告 AUC 以及 F1 值；
- (选做)使用 pcnn 作为编码器，设置 --aggr 为 avg ，也就是使用句子平均向量，训练以及推理，报告 AUC 以及 F1 值；
- 提交要求：在实验报告中给出前两种设置的 AUC(accuracy)以及 F1。

（二）实验过程

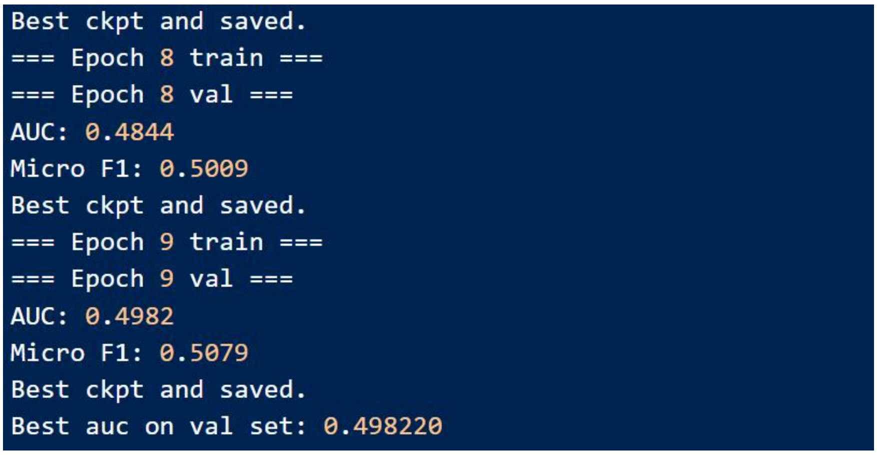
实验设置：使用 nyt10m 数据集，并将 epoch 设置为 10。

1. cnn+att 的 AUC 以及 F1



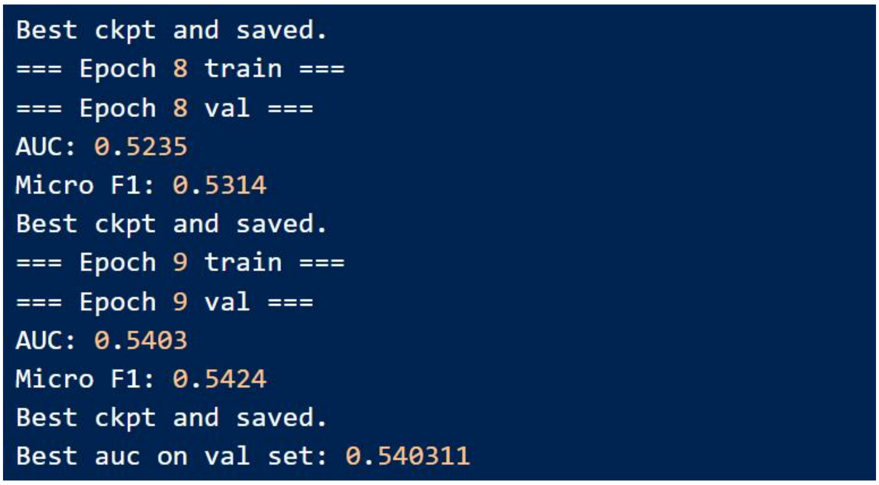
```
Micro F1: 0.4959
Best ckpt and saved.
=== Epoch 8 train ===
=== Epoch 8 val ===
AUC: 0.5007
Micro F1: 0.5108
Best ckpt and saved.
=== Epoch 9 train ===
=== Epoch 9 val ===
AUC: 0.5154
Micro F1: 0.5184
Best ckpt and saved.
Best auc on val set: 0.515377
```

2. cnn+avg 的 AUC 以及 F1



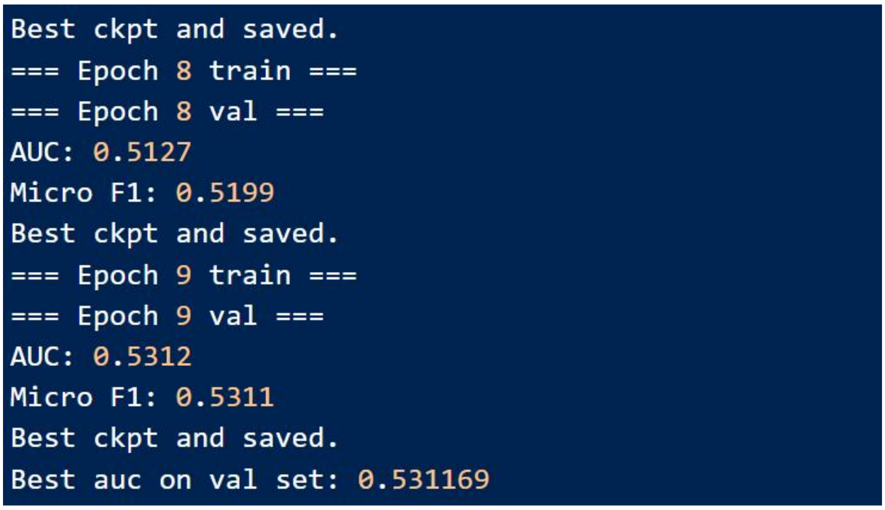
```
Best ckpt and saved.
=== Epoch 8 train ===
=== Epoch 8 val ===
AUC: 0.4844
Micro F1: 0.5009
Best ckpt and saved.
=== Epoch 9 train ===
=== Epoch 9 val ===
AUC: 0.4982
Micro F1: 0.5079
Best ckpt and saved.
Best auc on val set: 0.498220
```

3. pcnn+att 的 AUC 以及 F1



```
Best ckpt and saved.
=== Epoch 8 train ===
=== Epoch 8 val ===
AUC: 0.5235
Micro F1: 0.5314
Best ckpt and saved.
=== Epoch 9 train ===
=== Epoch 9 val ===
AUC: 0.5403
Micro F1: 0.5424
Best ckpt and saved.
Best auc on val set: 0.540311
```

4. pcnn+avg 的 AUC 以及 F1



```
Best ckpt and saved.
=== Epoch 8 train ===
=== Epoch 8 val ===
AUC: 0.5127
Micro F1: 0.5199
Best ckpt and saved.
=== Epoch 9 train ===
=== Epoch 9 val ===
AUC: 0.5312
Micro F1: 0.5311
Best ckpt and saved.
Best auc on val set: 0.531169
```

三、预训练模型关系抽取

（一）实验内容

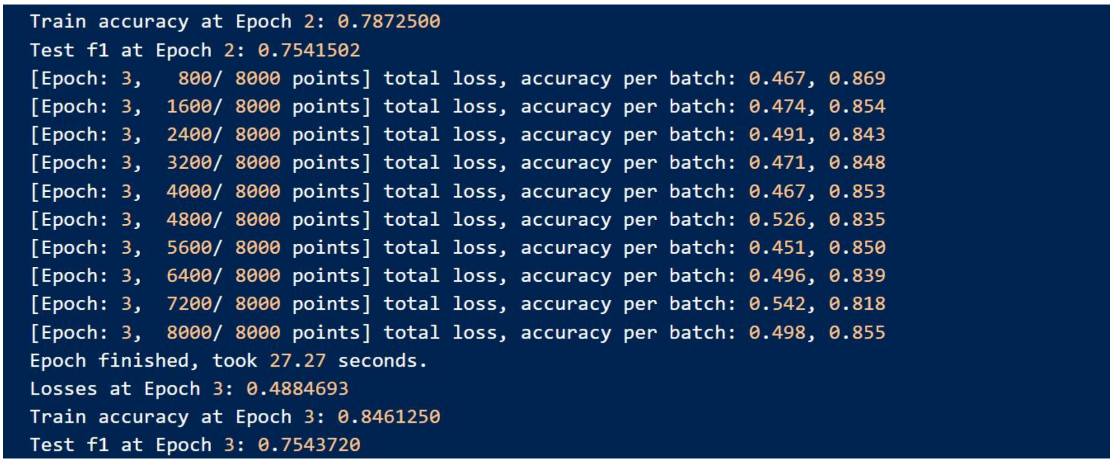
运行 main_task.py 代码，要求复现 accuracy>0.74；

模型默认使用了论文中的 ENTITY MARKERS+ENTITY START，修改成 ENTITY MARKERS+[CLS]并重复实验。并报告 accuracy；

提交要求：将两种设置的 accuracy 报告在实验报告中，提交代码文件 modeling_albert.py。

（二）实验过程

1. ENTITY MARKERS+ENTITY START 的 accuracy

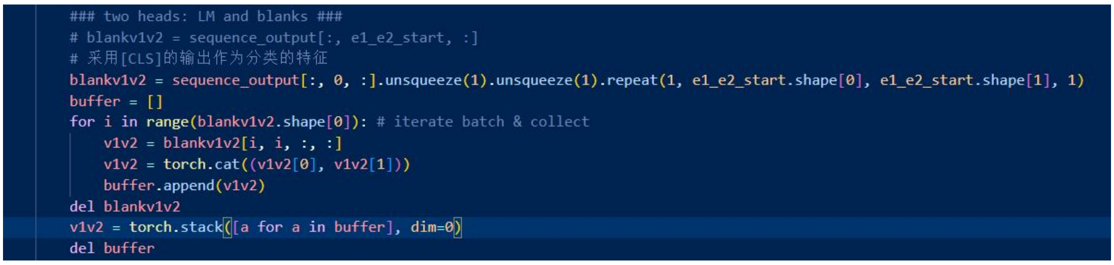


```
Train accuracy at Epoch 2: 0.7872500
Test f1 at Epoch 2: 0.7541502
[Epoch: 3,   800/ 8000 points] total loss, accuracy per batch: 0.467, 0.869
[Epoch: 3,  1600/ 8000 points] total loss, accuracy per batch: 0.474, 0.854
[Epoch: 3,  2400/ 8000 points] total loss, accuracy per batch: 0.491, 0.843
[Epoch: 3,  3200/ 8000 points] total loss, accuracy per batch: 0.471, 0.848
[Epoch: 3,  4000/ 8000 points] total loss, accuracy per batch: 0.467, 0.853
[Epoch: 3,  4800/ 8000 points] total loss, accuracy per batch: 0.526, 0.835
[Epoch: 3,  5600/ 8000 points] total loss, accuracy per batch: 0.451, 0.850
[Epoch: 3,  6400/ 8000 points] total loss, accuracy per batch: 0.496, 0.839
[Epoch: 3,  7200/ 8000 points] total loss, accuracy per batch: 0.542, 0.818
[Epoch: 3,  8000/ 8000 points] total loss, accuracy per batch: 0.498, 0.855
Epoch finished, took 27.27 seconds.
Losses at Epoch 3: 0.4884693
Train accuracy at Epoch 3: 0.8461250
Test f1 at Epoch 3: 0.7543720
```

2. ENTITY MARKERS+[CLS]的 accuracy

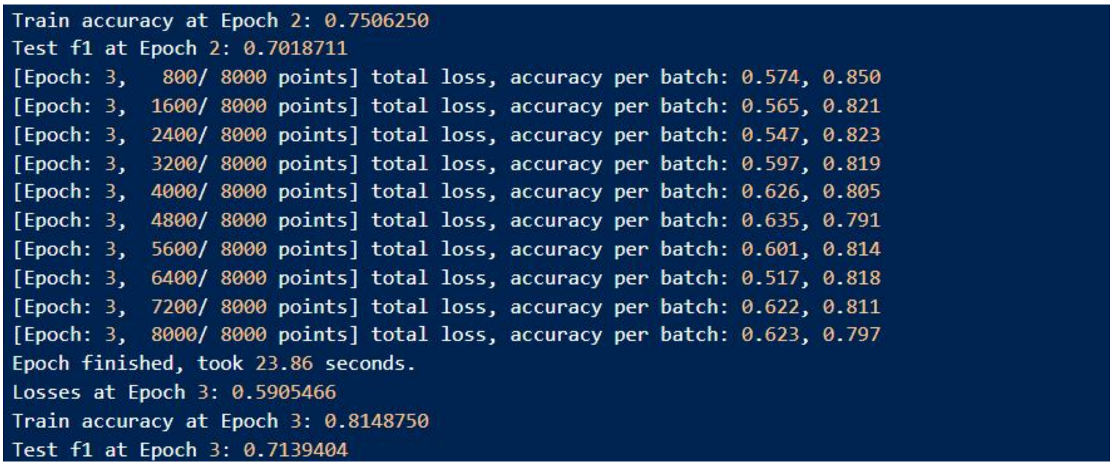
将 ENTITY MARKERS+ENTITY START，修改成 ENTITY MARKERS+[CLS]

并重复实验，代码如下：



```
### two heads: LM and blanks ###
# blankv1v2 = sequence_output[:, e1_e2_start, :]
# 采用[CLS]的输出作为分类的特征
blankv1v2 = sequence_output[:, 0, :].unsqueeze(1).unsqueeze(1).repeat(1, e1_e2_start.shape[0], e1_e2_start.shape[1], 1)
buffer = []
for i in range(blankv1v2.shape[0]): # iterate batch & collect
    v1v2 = blankv1v2[i, i, :, :]
    v1v2 = torch.cat((v1v2[0], v1v2[1]))
    buffer.append(v1v2)
del blankv1v2
v1v2 = torch.stack([a for a in buffer], dim=0)
del buffer
```

结果如下：



```
Train accuracy at Epoch 2: 0.7506250
Test f1 at Epoch 2: 0.7018711
[Epoch: 3,   800/ 8000 points] total loss, accuracy per batch: 0.574, 0.850
[Epoch: 3,  1600/ 8000 points] total loss, accuracy per batch: 0.565, 0.821
[Epoch: 3,  2400/ 8000 points] total loss, accuracy per batch: 0.547, 0.823
[Epoch: 3,  3200/ 8000 points] total loss, accuracy per batch: 0.597, 0.819
[Epoch: 3,  4000/ 8000 points] total loss, accuracy per batch: 0.626, 0.805
[Epoch: 3,  4800/ 8000 points] total loss, accuracy per batch: 0.635, 0.791
[Epoch: 3,  5600/ 8000 points] total loss, accuracy per batch: 0.601, 0.814
[Epoch: 3,  6400/ 8000 points] total loss, accuracy per batch: 0.517, 0.818
[Epoch: 3,  7200/ 8000 points] total loss, accuracy per batch: 0.622, 0.811
[Epoch: 3,  8000/ 8000 points] total loss, accuracy per batch: 0.623, 0.797
Epoch finished, took 23.86 seconds.
Losses at Epoch 3: 0.5905466
Train accuracy at Epoch 3: 0.8148750
Test f1 at Epoch 3: 0.7139404
```