基于BERT模型进行中文地址的分类识别

23年秋计算机应用技术 文本数据挖掘

作业组员：

冯　新、赵　云、吴宪斌、何忠远

一、项目背景

中文地址作为社会发展过程中不可或缺的地理空间数据资源和战略性基础信息资源，已成为智慧城市时空基础框架的重要组成部分，也是社会大数据组织、关联与共享应用的桥梁。中文地址匹配是将自然语言描述的地址数据，在地址模型或者编码规范的基础上通过智能化的地址解析，将解析结果与地址数据库中已有的数据进行比对，从而建立地址描述信息与空间坐标转换的过程。随着位置服务技术的广泛应用，积累了大量的地址数据，其中包含的地址要素类型及其组合方式更加复杂多样。当前，中文地址的管理和使用还处于较为初级的阶段，地址服务水平远不能满足社会各个行业的应用需求，因此必须深入挖掘中文地址的描述规律，实现中文地址的智能解析和高效匹配，才能更好的应对大数据时代下城市快速发展带来的严峻挑战。

二、解决思路

使用 Neural Chinese Address Parsing 作为样例，采用 BERT 模型进行训练，保存模型数据后，通过载入训练后的模型数据，解决中文地址的分类识别，并整合输出。

三、技术选型

1.技术环境：采用在 windows 系统上通过 WSL 部署 Ubuntu22.04.3LTS 作为该项目的实施平台。软件版本选型为 CUDA 版本为 12.3、Python 版本为 3.10.13、Pytorch 版本为 1.13.1。

2.模型选择：基于 Hugging Face 社区上的“bert-base-chinese”模型，通过 Neural Chinese Address Parsing 进行训练。

3.最终实现：选择编制 python 脚本，实现最终的中文地址的分类识别并输出

的功能实现。考虑的广泛适用性，该脚本 device 参数设置为“CPU”，以规避无英伟达显卡的情况。

四、具体实现

4.1 环境部署

通过 windows store 下载 Ubuntu22.04.3LTS 并安装，开发环境选用 VSCode。在 Ubuntu22.04.3LTS 上部署 anconda3、jupyter-notebook、pytorch、cuda、transformers、sklearn 等环境。

4.2 下载数据集

使用的数据集来自论文 Neural Chinese Address Parsing，下载地址为：git clone https://github.com/leodotnet/neural-chinese-address-parsing

4.3 训练

4.3.1 代码编写

4.3.1.1 训练

依赖、模型载入

```
import os
import time
import sklearn
import torch
import torch.nn.functional as F
from torch import nn
from tqdm import tqdm
from sklearn import metrics


from transformers import AdamW
from transformers import get_linear_schedule_with_warmup
device = torch.device("cuda")
max_train_epochs = 5
warmup_proportion = 0.05
gradient_accumulation_steps = 1
train_batch_size = 32
valid_batch_size = train_batch_size
test_batch_size = train_batch_size
data_workers= 2
learning_rate=2e-5
weight_decay=0.01
max_grad_norm=1.0
```

```
cur_time = time.strftime("%Y-%m-%d_%H:%M:%S")
base_path =
'/home/zhy/anaconda3/envs/trans/neural-chinese-address-parsing/data/'
from transformers import BertConfig, BertTokenizer, BertModel,
BertForTokenClassification
cls_token='[CLS]'
eos_token='[SEP]'
unk_token='[UNK]'
pad_token='[PAD]'
mask_token='[MASK]'
tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
config = BertConfig.from_pretrained('bert-base-chinese')
TheModel = BertModel
ModelForTokenClassification = BertForTokenClassification
eos_id = tokenizer.convert_tokens_to_ids([eos_token])[0]
unk_id = tokenizer.convert_tokens_to_ids([unk_token])[0]
period_id = tokenizer.convert_tokens_to_ids(['.'])[0]
```

标签

```
labels = ['B-assist', 'I-assist', 'B-cellno', 'I-cellno', 'B-city',
'I-city', 'B-community', 'I-community', 'B-country', 'I-country',
'B-devZone', 'I-devZone', 'B-district', 'I-district', 'B-floorno',
'I-floorno', 'B-houseno', 'I-houseno', 'B-otherinfo', 'I-otherinfo',
'B-person','I-person','B-poi','I-poi','B-prov','I-prov','B-redundant',
'I-redundant', 'B-road', 'I-road', 'B-roadno', 'I-roadno', 'B-roomno',
'I-roomno', 'B-subRoad', 'I-subRoad', 'B-subRoadno', 'I-subRoadno',
'B-subpoi', 'I-subpoi', 'B-subroad', 'I-subroad', 'B-subroadno',
'I-subroadno', 'B-town', 'I-town']
label2id = {}
for i, l in enumerate(labels):
    label2id[l] = i
num_labels = len(labels)
```

数据载入

```
def get_data_list(f):
    data_list = []
    origin_token, token, label = [], [], []
    for l in f:
        l = l.strip().split()
        if not l:
            data_list.append([token, label, origin_token])
            origin_token, token, label = [], [], []
            continue
        for i, tok in enumerate(l[0]):
            token.append(tok)
            label.append(label2id[l[1]])
        origin_token.append(l[0])
    assert len(token) == 0
    return data_list

f_train = open(base_path + 'train.txt')
f_test = open(base_path + 'test.txt')
f_dev = open(base_path + 'dev.txt')
train_list = get_data_list(f_train)
test_list = get_data_list(f_test)
dev_list = get_data_list(f_dev)
print(len(train_list), len(test_list), len(dev_list))
max_token_len = 0
for ls in [train_list, test_list, dev_list]:
    for l in ls:
        max_token_len = max(max_token_len, len(l[0]))
```

```
class MyDataSet(torch.utils.data.Dataset):
    def __init__(self, examples):
        self.examples = examples

    def __len__(self):
        return len(self.examples)
    def __getitem__(self, index):
        example = self.examples[index]
        sentence = example[0]
        #vaild_id = example[1]
        label = example[1]

        pad_len = max_token_len - len(sentence)
        total_len = len(sentence)+2

        input_token = [cls_token] + sentence + [eos_token] + [pad_token] *
pad_len
        input_ids = tokenizer.convert_tokens_to_ids(input_token)
        attention_mask = [1] + [1] * len(sentence) + [1] + [0] * pad_len
        label = [-100] + label + [-100] + [-100] * pad_len
        assert max_token_len + 2 == len(input_ids) == len(attention_mask)
== len(input_token)
```

```
        return input_ids, attention_mask, total_len, label, index
def the_collate_fn(batch):
    total_lens = [b[2] for b in batch]
    total_len = max(total_lens)
    input_ids = torch.LongTensor([b[0] for b in batch])
    attention_mask = torch.LongTensor([b[1] for b in batch])
    label = torch.LongTensor([b[3] for b in batch])
    input_ids = input_ids[:,:total_len]
    attention_mask = attention_mask[:,:total_len]
    label = label[:,:total_len]
    indexs = [b[4] for b in batch]
    return input_ids, attention_mask, label, indexs
train_dataset = MyDataSet(train_list)
train_data_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=train_batch_size,
    shuffle = True,
    num_workers=data_workers,
    collate_fn=the_collate_fn,
)
test_dataset = MyDataSet(test_list)
test_data_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=train_batch_size,
    shuffle = False,
    num_workers=data_workers,
    collate_fn=the_collate_fn,
)
```

结果输出

```
def eval():
    result = []
    for step, batch in enumerate(tqdm(test_data_loader)):
        input_ids, attention_mask, label = (b.to(device) for b in batch[:-1])
        with torch.no_grad():
            logits = model(input_ids, attention_mask)
            logits = F.softmax(logits, dim=-1)
        logits = logits.data.cpu()
        logit_list = []
        sum_len = 0
        for m in attention_mask:
            l = m.sum().cpu().item()
            logit_list.append(logits[sum_len:sum_len+l])
            sum_len += l
        assert sum_len == len(logits)
        for i, l in enumerate(logit_list):
            rr = torch.argmax(l, dim=1)
            for j, w in enumerate(test_list[batch[-1][i]][0]):
                result.append([w,
labels[label[i][j+1].cpu().item()],labels[rr[j+1]]])
            result.append([])
    print(result[:20])

def log(msg):
    print(msg)
```

```
class BertForSeqTagging(ModelForTokenClassification):
    def __init__(self):
        super().__init__(config)
        self.num_labels = num_labels
        self.bert = TheModel.from_pretrained('bert-base-chinese')
        self.dropout = torch.nn.Dropout(config.hidden_dropout_prob)
        self.classifier = torch.nn.Linear(config.hidden_size, num_labels)
        self.init_weights()

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask)
        sequence_output = outputs[0]
        batch_size, max_len, feature_dim = sequence_output.shape
        sequence_output = self.dropout(sequence_output)
        logits = self.classifier(sequence_output)
        active_loss = attention_mask.view(-1) == 1
        active_logits = logits.view(-1, self.num_labels)[active_loss]

        if labels is not None:
            loss_fct = torch.nn.CrossEntropyLoss()
            active_labels = labels.view(-1)[active_loss]
            loss = loss_fct(active_logits, active_labels)
            return loss
        else:
            return active_logits

model = BertForSeqTagging()
model.to(device)
t_total = len(train_data_loader) // gradient_accumulation_steps *
max_train_epochs + 1
```

```
num_warmup_steps = int(warmup_proportion * t_total)
log('warmup steps : %d' % num_warmup_steps)
no_decay = ['bias', 'LayerNorm.weight']
param_optimizer = list(model.named_parameters())
optimizer_grouped_parameters = [
    {'params':[p for n, p in param_optimizer if not any(nd in n for nd in
no_decay)],'weight_decay': weight_decay},
    {'params':[p for n, p in param_optimizer if any(nd in n for nd in
no_decay)],'weight_decay': 0.0}
]
optimizer = AdamW(optimizer_grouped_parameters, lr=learning_rate,
correct_bias=False)
scheduler = get_linear_schedule_with_warmup(optimizer,
num_warmup_steps=num_warmup_steps, num_training_steps=t_total)
```

训练

```
for epoch in range(max_train_epochs):
    # train
    epoch_loss = None
    epoch_step = 0
    start_time = time.time()
    model.train()
    for step, batch in enumerate(tqdm(train_data_loader)):
        input_ids, attention_mask, label = (b.to(device) for b in batch[:-1])
        loss = model(input_ids, attention_mask, label)
        loss.backward()

        if (step + 1) % gradient_accumulation_steps == 0:
            optimizer.step()
            scheduler.step()
            optimizer.zero_grad()

        if epoch_loss is None:
            epoch_loss = loss.item()
        else:
            epoch_loss = 0.98*epoch_loss + 0.02*loss.item()
        epoch_step += 1

    used_time = (time.time() - start_time)/60
    log('Epoch = %d Epoch Mean Loss %.4f Time %.2f min' % (epoch, epoch_loss,
used_time))
    result = eval()
    with open('result.txt', 'w') as f:
        for r in result:
            f.write('\t'.join(r) + '\n')
    y_true = []
    y_pred = []
    for r in result:
        if not r: continue
        y_true.append(label2id[r[1]])
        y_pred.append(label2id[r[2]])
    print(sklearn.metrics.f1_score(y_true, y_pred, average='micro'))
```

模型保存

```
torch.save(model.state_dict(),
'Neural_Chinese_Address_Parsing_BERT_state_dict.pkl')
```

4.3.1.2 使用实例

依赖引用、模型调用、函数定义

```
import os
import time
import sklearn
import torch
import torch.nn.functional as F
from torch import nn
from tqdm import tqdm
from sklearn import metrics


from transformers import AdamW
from transformers import get_linear_schedule_with_warmup
device = torch.device("cpu")
max_train_epochs = 5
warmup_proportion = 0.05
gradient_accumulation_steps = 1
train_batch_size = 32
valid_batch_size = train_batch_size
test_batch_size = train_batch_size
data_workers= 2
learning_rate=2e-5
weight_decay=0.01
max_grad_norm=1.0

cur_time = time.strftime("%Y-%m-%d_%H:%M:%S")
```

```
model_path =
'/home/zhy/anaconda3/envs/trans/trans-p/Neural_Chinese_Address_Parsing_
BERT_state_dict.pkl'
from transformers import BertConfig, BertTokenizer, BertModel,
BertForTokenClassification
cls_token='[CLS]'
eos_token='[SEP]'
unk_token='[UNK]'
pad_token='[PAD]'
mask_token='[MASK]'
tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
config = BertConfig.from_pretrained('bert-base-chinese')
TheModel = BertModel
ModelForTokenClassification = BertForTokenClassification
labels = ['B-assist', 'I-assist', 'B-cellno', 'I-cellno', 'B-city',
'I-city', 'B-community', 'I-community', 'B-country', 'I-country',
'B-devZone', 'I-devZone', 'B-district', 'I-district', 'B-floorno',
'I-floorno', 'B-houseno', 'I-houseno', 'B-otherinfo', 'I-otherinfo',
'B-person', 'I-person', 'B-poi', 'I-poi', 'B-prov', 'I-prov', 'B-redundant',
'I-redundant', 'B-road', 'I-road', 'B-roadno', 'I-roadno', 'B-roomno',
'I-roomno', 'B-subRoad', 'I-subRoad', 'B-subRoadno', 'I-subRoadno',
'B-subpoi', 'I-subpoi', 'B-subroad', 'I-subroad', 'B-subroadno',
'I-subroadno', 'B-town', 'I-town']
label2id = {}
for i, l in enumerate(labels):
    label2id[l] = i
num_labels = len(labels)
class BertForSeqTagging(ModelForTokenClassification):
    def __init__(self):
        super().__init__(config)
        self.num_labels = num_labels
        self.bert = TheModel.from_pretrained('bert-base-chinese')
        self.dropout = torch.nn.Dropout(config.hidden_dropout_prob)
        self.classifier = torch.nn.Linear(config.hidden_size, num_labels)
        self.init_weights()

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.bert(input_ids=input_ids,
attention_mask=attention_mask)
        sequence_output = outputs[0]
        batch_size, max_len, feature_dim = sequence_output.shape
        sequence_output = self.dropout(sequence_output)
        logits = self.classifier(sequence_output)
        active_loss = attention_mask.view(-1) == 1
```

```python
        active_logits = logits.view(-1, self.num_labels)[active_loss]

        if labels is not None:
            loss_fct = torch.nn.CrossEntropyLoss()
            active_labels = labels.view(-1)[active_loss]
            loss = loss_fct(active_logits, active_labels)
            return loss
        else:
            return active_logits

def eval():
    result = []
    for step, batch in enumerate(tqdm(test_data_loader)):
        input_ids, attention_mask, label = (b.to(device) for b in batch[:-1])
        with torch.no_grad():
            logits = model(input_ids, attention_mask)
            logits = F.softmax(logits, dim=-1)
        logits = logits.data.cpu()
        logit_list = []
        sum_len = 0
        for m in attention_mask:
            l = m.sum().cpu().item()
            logit_list.append(logits[sum_len:sum_len+l])
            sum_len += l
        assert sum_len == len(logits)
        for i, l in enumerate(logit_list):
            rr = torch.argmax(l, dim=1)
            for j, w in enumerate(test_list[batch[-1][i]][0]):
                result.append([w,
labels[label[i][j+1].cpu().item()],labels[rr[j+1]]])
            result.append([])
    print(result[:20])
    return result
class MyDataSet(torch.utils.data.Dataset):
    def __init__(self, examples):
        self.examples = examples

    def __len__(self):
        return len(self.examples)
    def __getitem__(self, index):
        example = self.examples[index]
        sentence = example[0]
        #vaild_id = example[1]
        label = example[1]
        pad_len = max_token_len - len(sentence)
        total_len = len(sentence)+2

        input_token = [cls_token] + sentence + [eos_token] + [pad_token] *
pad_len
        input_ids = tokenizer.convert_tokens_to_ids(input_token)
        attention_mask = [1] + [1] * len(sentence) + [1] + [0] * pad_len
        # vaild_mask = [0] + vaild_id + [0] + [0] * pad_len
        # active_mask = [1] * len(label) + [0] * (max_token_len+2-len(label))
        label = [-100] + label + [-100] + [-100] * pad_len
        assert max_token_len + 2 == len(input_ids) == len(attention_mask)
== len(input_token)# == len(vaild_mask)

        return input_ids, attention_mask, total_len, label, index

def the_collate_fn(batch):
    total_lens = [b[2] for b in batch]
    total_len = max(total_lens)
    input_ids = torch.LongTensor([b[0] for b in batch])
    attention_mask = torch.LongTensor([b[1] for b in batch])
    label = torch.LongTensor([b[3] for b in batch])
    input_ids = input_ids[:,:total_len]
    attention_mask = attention_mask[:,:total_len]
    label = label[:,:total_len]
    indexs = [b[4] for b in batch]
    return input_ids, attention_mask, label, indexs
```

模型训练数据载入

```python
model = BertForSeqTagging()
model.load_state_dict(torch.load(model_path))
```

数据载入

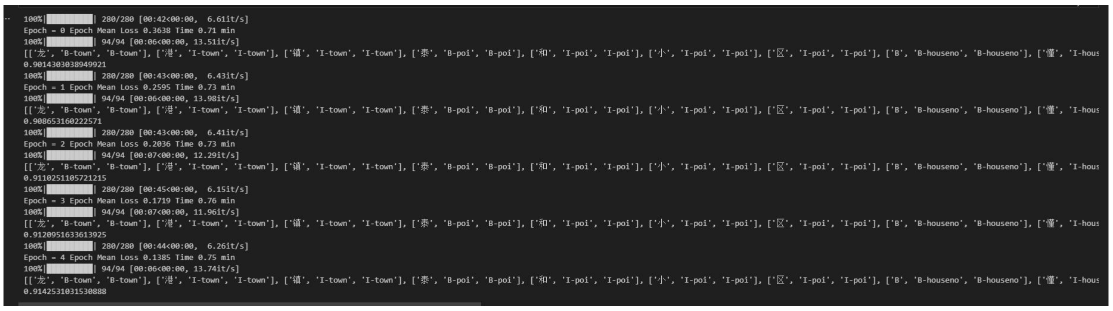
```python
cent = input("Enter any value: ")
inputs = tokenizer(cent, add_special_tokens=False)
i2 =[]
for aa in cent:
    i2.append(aa)

a1=inputs['attention_mask']
a2 =[]
for aa in a1:
    a2.append(aa)
```

```python
test_list=[]
test_list.append([i2,a2])
test_dataset = MyDataSet(test_list)
test_data_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=train_batch_size,
    shuffle = False,
    num_workers=data_workers,
    collate_fn=the_collate_fn,
)
```

结果计算并处理

```python
max_token_len=0
for ls in test_list:
    max_token_len = max(max_token_len, len(ls[0]))
print('max_token_len', max_token_len)
result = eval()
ii = -1
aas = []
tt = []
for aa in result[:-1]:
        a1,a2,a3=aa
        if a3[0] == "B":
                print('')
                print(a3[2:],':')
        print(a1,end='')
```

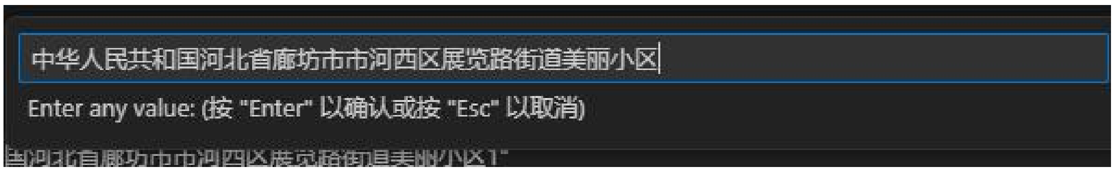
4.3.2 训练结果



```
100%|██████████| 280/280 [00:42<00:00,  6.61it/s]
Epoch = 0 Epoch Mean Loss 0.3638 Time 0.71 min
100%|██████████| 94/94 [00:06<00:00, 13.51it/s]
[['龙', 'B-town', 'B-town'], ['港', 'I-town', 'I-town'], ['镇', 'I-town', 'I-town'], ['泰', 'B-poi', 'B-poi'], ['和', 'I-poi', 'I-poi'], ['小', 'I-poi', 'I-poi'], ['区', 'I-poi', 'I-poi'], ['B', 'B-houseno', 'B-houseno'], ['懂', 'I-hous
0.9014303038949921
100%|██████████| 280/280 [00:43<00:00,  6.43it/s]
Epoch = 1 Epoch Mean Loss 0.2595 Time 0.73 min
100%|██████████| 94/94 [00:06<00:00, 13.98it/s]
[['龙', 'B-town', 'B-town'], ['港', 'I-town', 'I-town'], ['镇', 'I-town', 'I-town'], ['泰', 'B-poi', 'B-poi'], ['和', 'I-poi', 'I-poi'], ['小', 'I-poi', 'I-poi'], ['区', 'I-poi', 'I-poi'], ['B', 'B-houseno', 'B-houseno'], ['懂', 'I-hous
0.908653160222571
100%|██████████| 280/280 [00:43<00:00,  6.41it/s]
Epoch = 2 Epoch Mean Loss 0.2036 Time 0.73 min
100%|██████████| 94/94 [00:07<00:00, 12.29it/s]
[['龙', 'B-town', 'B-town'], ['港', 'I-town', 'I-town'], ['镇', 'I-town', 'I-town'], ['泰', 'B-poi', 'B-poi'], ['和', 'I-poi', 'I-poi'], ['小', 'I-poi', 'I-poi'], ['区', 'I-poi', 'I-poi'], ['B', 'B-houseno', 'B-houseno'], ['懂', 'I-hous
0.9110251105721215
100%|██████████| 280/280 [00:45<00:00,  6.15it/s]
Epoch = 3 Epoch Mean Loss 0.1719 Time 0.76 min
100%|██████████| 94/94 [00:07<00:00, 11.96it/s]
[['龙', 'B-town', 'B-town'], ['港', 'I-town', 'I-town'], ['镇', 'I-town', 'I-town'], ['泰', 'B-poi', 'B-poi'], ['和', 'I-poi', 'I-poi'], ['小', 'I-poi', 'I-poi'], ['区', 'I-poi', 'I-poi'], ['B', 'B-houseno', 'B-houseno'], ['懂', 'I-hous
0.9120951633613925
100%|██████████| 280/280 [00:44<00:00,  6.26it/s]
Epoch = 4 Epoch Mean Loss 0.1385 Time 0.75 min
100%|██████████| 94/94 [00:06<00:00, 13.74it/s]
[['龙', 'B-town', 'B-town'], ['港', 'I-town', 'I-town'], ['镇', 'I-town', 'I-town'], ['泰', 'B-poi', 'B-poi'], ['和', 'I-poi', 'I-poi'], ['小', 'I-poi', 'I-poi'], ['区', 'I-poi', 'I-poi'], ['B', 'B-houseno', 'B-houseno'], ['懂', 'I-hous
0.9142531031530888
```

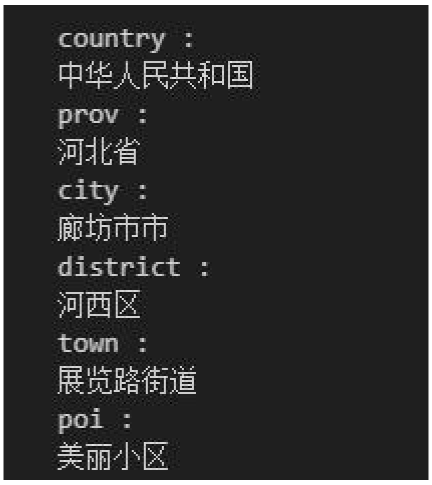
4.3.3 最终实现

输入数据



```
中华人民共和国河北省廊坊市市河西区展览路街道美丽小区
Enter any value: (按 "Enter" 以确认或按 "Esc" 以取消)
```

输出结果



```
country :
中华人民共和国
prov :
河北省
city :
廊坊市市
district :
河西区
town :
展览路街道
poi :
美丽小区
```

五、分析和进一步展望

当前模型使用的 bert 模型，训练量也不大，就已经可以达到 90%的正确率。中文地址信息在绝大多数情况下特征明显，运算难度并不大，在较为标准的地址文本分析中，通过微调模型提升效果已进入瓶颈。且中文语境也有其特有的复杂性，部分特殊地名的识别也会出现识别异常缺，暂无针对性训练数据的情况。进一步想要更好地实现该功能，可能应使用更加复杂的模型才可以进一步提高识别的准确性。

附录 ：人员名单及负责工作内容

工作内容	负责组员
1. 组织工作	冯　新
2. 整理数据集	何忠远（数据集收集）、吴宪斌（整理）
3. 环境搭建	吴宪斌、何忠远

4. 模型训练	赵　云、冯　新
5. 功能实现	赵　云
6. 演示文稿	何忠远、冯　新