机器学习工程师纳米学位结题报告

-算式识别（挑战项目）

2019 年 4 月 9 日

1 问题的定义

1.1 项目概述

本项目来自 Udacity 机器学习毕业项目，计算机视觉小类。项目的其任务是利用深度学习方法，识别不同图片中的数学算式。其输入为彩色图片，其输出为算式的内容。

本项目的数据集来自 Udacity [1]。原始文件中包括如图 1 所示样式的十万张数学运算图，此外还有一个 CSV 文件，文件中记录了图片的存储路径以及每张图片的真实标签，如图 1.2 所示。

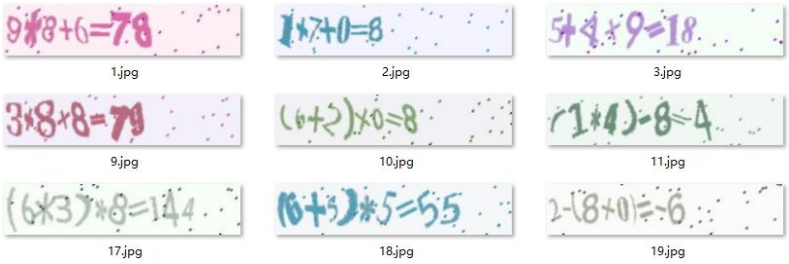


图 1.1 所需识别的图片实例

1	filename	label
2	train/0.jpg	(0+0)+9=9
3	train/1.jpg	9*8+6=78
4	train/2.jpg	1+7+0=8

图 1.2 文件数据存储格式

1.2 问题陈述

本项目需要识别不同图片中的数学算式。从本质上讲，这首先是一个 OCR（Optical Character Recognition）问题，但进一步地，由于数学算式同时具有序列的相关特征，这又是一个序列识别的问题。在现实世界中，场景文字，手写字符等，往往以序列的形式出现，识别序列需要系统预测一系列对象标签，序列对

象的另一个独特之处在于它们的长度可能会有很大变化。

进一步对所需识别的算式进行分析，算式中可能包含“+”，“-”以及“*”三种运算符，此外可能包含一对括号，也可能不包含括号，并且，每个字符都可能旋转。由于以上这些特征，可能在以下几个方面增加问题的复杂度：

1. 括号的存在以及计算后的数值大小不同，会导致序列长度不固定；
2. 字符在平面内可旋转，会对其识别带来较大影响，比如“+”以及“*”在某些情况下会更加相似。

通过对任务进一步的分析，此问题基本可以被归纳为一个端到端的不定长序列识别问题。解决此类问题较好的方法是通过卷积循环神经元网络（CRNN）[2]，它是 DCNN 和 RNN 的组合，是一种将特征提取，序列建模和转录整合到统一框架中的新型神经网络架构。其优点包括[2]：

（1）与大多数现有的组件需要单独训练和协调的算法相比，它是端对端训练的。

（2）它自然地处理任意长度的序列，不涉及字符分割或水平尺度归一化。

（3）它不仅限于任何预定义的词汇，并且在无词典和基于词典的场景文本识别任务中都取得了显著的表现。

程序的大致框架如下：

1. 分割数据集为训练，测试，验证。
2. 定义图像矩阵转换函数，在此问题中，可将彩图均转换成灰度图片，可选择性进行归一化操作。
3. 定义网络。
4. 定义解码函数，将网络输出结果转化成字符串格式的算式。
5. 定义准确率，损失函数。

6. 通过训练和验证集，完成模型训练。

7. 在测试集中测试模型，如果达不到要求，则修改相关参数，重新训练。

8. 绘制相关损失，准确率曲线。

项目所需达到的效果是获得 99%及以上的准确率。

1.3 评价指标

对解码后的字符串与图片对应的标签进行匹配，当两者相同时，就记为正确识别，否则记为识别错误，最后统计测试集中识别正确的图片数占测试集总数的比例，成为模型识别的正确率。

2 分析

2.1 数据探索

图 1 已展示了图片的基本样式，通过比对，可以发现部分字符的形态有较大差异。如图 2.1 所示，在不同算式中，括号呈现了不同的形态，(a)与（b）更为粗壮，（c）中呈现细长状，（d）中则更符合括号的一般形态。与此同时，在部分图片中，算式与底色在对比度上也呈现了明显不同，如图 2.2 所示，在部分图中，算式与底色的颜色接近，难以辨认，这可能会导致部分识别失败。

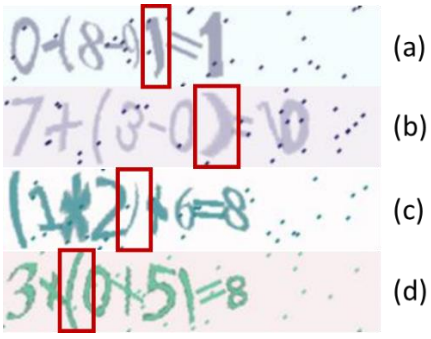


图 2.1 括号的几种形态

图 2.2 低对比度的图片

2.2 算法和技术

2.2.1 CRNN

卷积循环神经网络（CRNN）是 DCNN 和 RNN 的组合，其网络架构由三部

分组成，包括卷积层，循环层和转录层。卷积层负责从输入图像中提取特征序列，循环层负责预测每一帧的标签分布，转录层则将每一帧的预测变为最终的标签序列[2]，其具体网络架构如图 2.3 所示。在整个架构中，循环层位于卷积层的顶部，由长短时记忆（LSTM）单元[3]组成了深度双向循环神经网络。

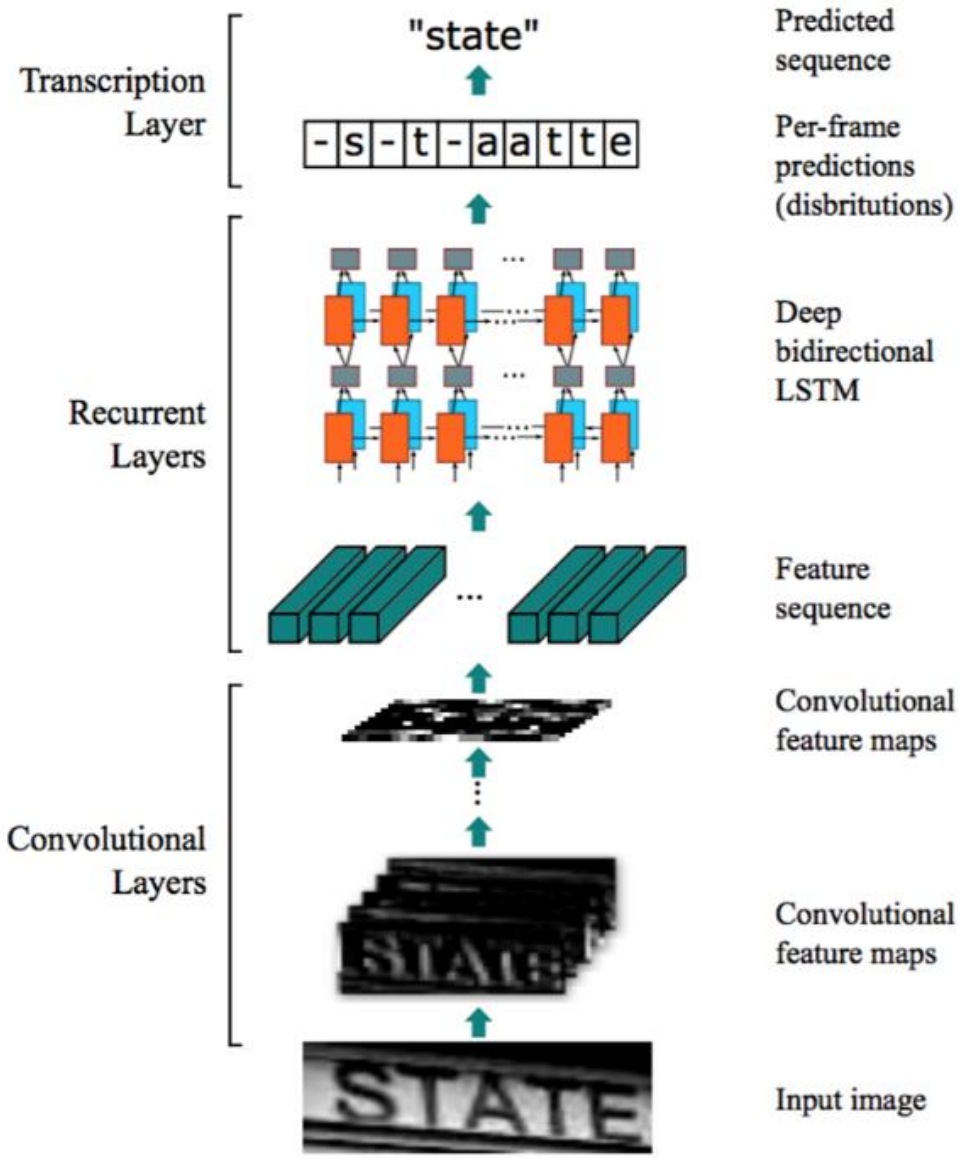
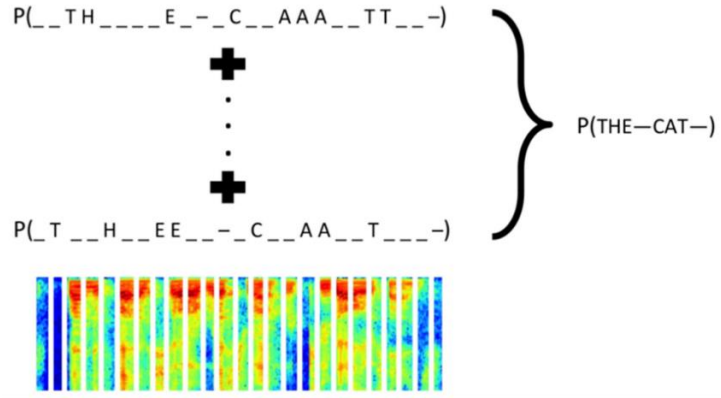


图 2.3 CRNN 的网络架构

2.2.2 CTC 损失

CTC 全称为 Connectionist Temporal Classification，CTC 损失允许神经网络在任意一个时间段内进行预测，无需对序列进行对齐操作，而只需要确保输出的

序列顺序正确即可。在 Keras 里面，CTC Loss 已经内置，我们只需定义如图 2.4 所示的一个函数即可。




```
from keras import backend as K

def ctc_lambda_func(args):
    y_pred, labels, input_length, label_length = args
    y_pred = y_pred[:, 2:, :]
    return K.ctc_batch_cost(labels, y_pred, input_length, label_length)
```

图 2.4 CTC 损失在 keras 中的实现

2.3 基准模型

根据项目要求，将以 99%的正确率作为基准阈值，即通过训练的 CRNN 模型需要在测试集上获得大于等于 99%的准确率。此要求对一个 OCR 识别问题来说是较为合适的。

3 方法

3.1 数据处理

3.1.1 数据分割

通过 sklearn 中的 train_test_split，将原始数据分割成训练，验证，测试以及预测三部分，具体比例为 8：1：1。

```
#split the data set, get 100000 test data
img_path_rest, img_path_test, labels_rest, labels_test = train_test_split(paths, labels, test_size=10000, random_state=10)
#split the data set, get 100000 valid data
img_path_train, img_path_valid, labels_train, labels_valid = train_test_split(img_path_rest, labels_rest, test_size=10000, random_state=10)
print(len(img_path_train),len(img_path_valid),len(img_path_test))
print(img_path_train[0])
#should be updated
```

```
80000 10000 10000
```

图 3.1 数据集分割

本项目主要针对图像进行端到端的序列识别，序列的长度是其中之一的属性，

通过查询所有标签长度可以发现，标签的最大长度为 11，最小长度为 7。通过统计所有标签的长度，可以获得标签的分布状况，具体如图 3.2 所示，可以发现，有 30%以上的图片标签为 10，仅有 5%左右的标签长度为 11，在训练，验证，测试集中，其分布性基本一致。

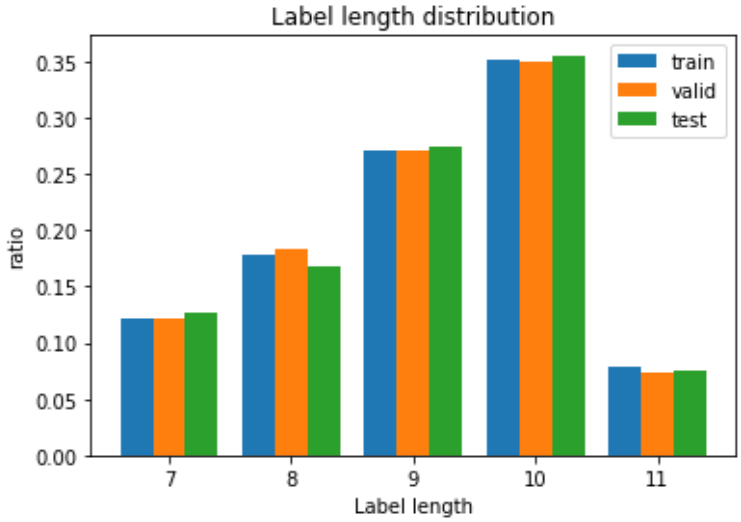


图 3.2 标签长度占比分布图

3.1.2 图片读取

通过 CV2 完成图片的读取与灰度转化，对灰度转化后的数组进行归一化操作，其具体方法为当前灰度值除以最大灰度值 255（对应于白色），通过 Numpy 对归一化后的所有数组进行拼接操作。其实现方法如图 3.3 所示。

```
#turn a path list into numpy array
def tensors_trans(paths):
    img_list=[]
    for element in paths:
        ele=os.path.join(src_path,element)
        img=cv2.imread(ele)
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)/255
        img_gray = np.expand_dims(image.img_to_array(img_gray),axis=0)
        img_list.append(img_gray)
    img_tensor=np.vstack(img_list)
    return img_tensor
```

图 3.3 图像预处理函数

3.2 执行过程

根据文献给出的网络架构，搭建了网络，具体如图 3.4 所示。由于输出为多

分类问题，因此使用 softmax 函数。

```
height, width, channel = X_train.shape[1:]
input_tensor = Input(shape=[height, width, channel])
x = input_tensor

x = Conv2D(32*2**0, 3, strides=1, padding='same', activation='relu',kernel_initializer='he_normal')(x)
x = MaxPooling2D((2, 2), strides=2, padding='same')(x)
x = Conv2D(32*2**1, 3, strides=1, padding='same', activation='relu',kernel_initializer='he_normal')(x)
x = MaxPooling2D((2, 2), strides=2, padding='same')(x)
x = Conv2D(32*2**2, 3, strides=1, padding='same', activation='relu',kernel_initializer='he_normal')(x)
x = MaxPooling2D((2, 2), strides=2, padding='same')(x)
x = Conv2D(256, 3, strides=1, padding='same', activation='relu',kernel_initializer='he_normal')(x)
x = BatchNormalization(axis=-1)(x)

x = Conv2D(512, 3, strides=1, padding='same', activation='relu')(x)
x = BatchNormalization(axis=-1)(x)

x = MaxPooling2D((1, 2), strides=2, padding='same')(x)
x = Dropout(rate=0.5)(x)

x = Permute((2,1,3))(x)

x = TimeDistributed(Flatten())(x)

x = Bidirectional(CuDNNLSTM(64, return_sequences=True))(x)
x = Bidirectional(CuDNNLSTM(64, return_sequences=True))(x)

y_pred = Dense(n_class,activation='softmax')(x)

base_model = Model(inputs=input_tensor,outputs = y_pred)
```

图 3.4 网络架构

网络输出为对应每个字符的概率，需要将其转换成字符序列，设计了如下 3.5 所示的函数，对输出进行转化，选择概率最大的字符，去除多余空格，形成最终所需的算术表达式。

```
chars_space = chars + u' '#unicode
def decode(pred):
    pred = pred.argmax(axis=2)#get the most possible char
    orig_labels = []
    new_labels = []
    for i, code in enumerate(pred):
        orig_labels.append(''.join([chars_space[x] for x in code]))
    for string in orig_labels:
        new_labels.append(re.sub(r'\s','',re.sub(r'(.)\1+', r'\1', string)))#remove uncessary space and ```
    return [np.array(orig_labels), np.array(new_labels)]
```

图 3.5 输出转换函数

3.2 完善

3.2.1 初次运行

考虑到模型的鲁棒性，为了防止过拟合，在三个初始的池化层后均加入了 dropout，并且，由于部分图片存在对比度过低的问题，在图像读取时，还加入了灰度图像直方图均衡化 equalizeHist [4]。最终在测试集上的精度为 98%。

3.2.2 优化

主要优化了 dropout 的比例，并去除了直方图均衡化（均衡化效果不明显）。最终的网络架构如图 3.6 所示。

Layer (type)	Output Shape	Param #	Connected to
input_9 (InputLayer)	(None, 64, 300, 1)	0	
conv2d_71 (Conv2D)	(None, 64, 300, 32)	320	input_9[0][0]
max_pooling2d_28 (MaxPooling2D)	(None, 32, 150, 32)	0	conv2d_71[0][0]
dropout_13 (Dropout)	(None, 32, 150, 32)	0	max_pooling2d_28[0][0]
conv2d_72 (Conv2D)	(None, 32, 150, 64)	18496	dropout_13[0][0]
max_pooling2d_29 (MaxPooling2D)	(None, 16, 75, 64)	0	conv2d_72[0][0]
dropout_14 (Dropout)	(None, 16, 75, 64)	0	max_pooling2d_29[0][0]
conv2d_73 (Conv2D)	(None, 16, 75, 128)	73856	dropout_14[0][0]
max_pooling2d_30 (MaxPooling2D)	(None, 8, 38, 128)	0	conv2d_73[0][0]
dropout_15 (Dropout)	(None, 8, 38, 128)	0	max_pooling2d_30[0][0]
conv2d_74 (Conv2D)	(None, 8, 38, 256)	295168	dropout_15[0][0]
batch_normalization_27 (BatchNo	(None, 8, 38, 256)	1024	conv2d_74[0][0]
conv2d_75 (Conv2D)	(None, 8, 38, 512)	1180160	batch_normalization_27[0][0]
batch_normalization_28 (BatchNo	(None, 8, 38, 512)	2048	conv2d_75[0][0]
max_pooling2d_31 (MaxPooling2D)	(None, 4, 19, 512)	0	batch_normalization_28[0][0]
dropout_16 (Dropout)	(None, 4, 19, 512)	0	max_pooling2d_31[0][0]
permute_8 (Permute)	(None, 19, 4, 512)	0	dropout_16[0][0]
time_distributed_6 (TimeDistrib	(None, 19, 2048)	0	permute_8[0][0]
bidirectional_12 (Bidirectional	(None, 19, 128)	1082368	time_distributed_6[0][0]
bidirectional_13 (Bidirectional	(None, 19, 128)	99328	bidirectional_12[0][0]
dense_12 (Dense)	(None, 19, 17)	2193	bidirectional_13[0][0]
the_labels (InputLayer)	(None, None)	0	
input_length (InputLayer)	(None, 1)	0	
label_length (InputLayer)	(None, 1)	0	
ctc (Lambda)	(None, 1)	0	dense_12[0][0] the_labels[0][0] input_length[0][0] label_length[0][0]

Total params: 2,754,961
Trainable params: 2,753,425
Non-trainable params: 1,536

图 3.6 最终网络设置

4 结果

4.1 模型的评价与验证

本模型的结构来源于文献[2]，在增强鲁棒性上，增加了一些 DropOut 层，通过双向 LSTM 搭建循环层（由于 kears 与 TensorFlow 的版本匹配问题，暂时没

有实现 GRU 的使用)。从整个网络结构来看，是较为合理的，可信度较高。

4.1.1 合理性分析

经过参数调整后，模型的准确率为 99.3%，高于要求的的阈值，证明模型的准确率与鲁棒性都有所提高，高于要求的 99%。

4.2 项目结论

4.2.1 结果可视化

本项目建立了 OCR 算术识别的 CRNN 模型，通过训练与验证，在 10000 测试样本上取得了 99.3%的准确率。

```
base_model.load_weights(model_file_path)
pred = base_model.predict(X_test)
sequence, new_labels = decode(pred)
test_acc = (new_labels==labels_test).mean()*100
print('test_acc: {}%'.format(round(test_acc,2)))
#1round acc:98.09%
#2round acc:97.05%
#```
#finial
```

```
test_acc: 99.26%
```

图 4.1 最终测试集正确率

在训练过程中，训练集上的损失逐渐降低，准确率逐渐提高，如图所示。为了防止过拟合，设置了早停机制，模型在第 61 个 epoch 时停止迭代（预设值为 400），从此以后，模型的过拟合程度会逐渐升高，不利于其对未知算式的识别的检测，因此放弃。

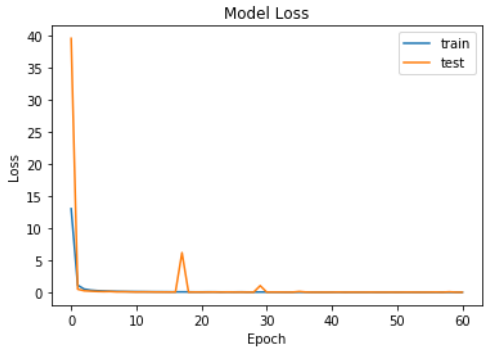


图 4.2 Loss 随 epoch 变化图

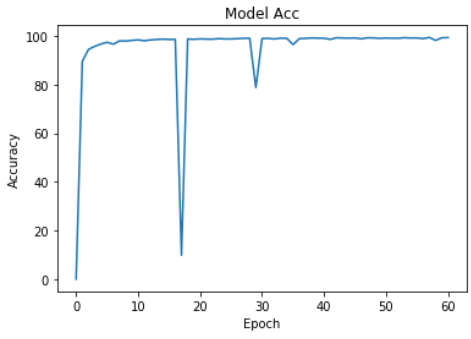


图 4.3 Accuracy 随 epoch 变化图

4.3 对项目的思考

本项目是一个不定长的图像序列识别问题，通过 CRNN，较好解决了这一问题，项目结果符合预期。此项目的难点主要集中在序列的不定长上，而 CTC 损失正是解决这个问题的有效方法之一。通过这一项目，仔细了解了 CNN，RNN 的工作原理，以及图像中识别序列的基本方法。此技术的运用广泛，Github 中有许多关于此技术的实际运用项目，比如利用 CRNN 识别车牌 [5]，比如基于不定长序列识别的现实场景中的文字检测[6]等。

4.4 需要做出的改进

1） 没有进行图像增强操作，如果有此操作，理应更好提高项目准确度。

2） 使用迭代器能减少内存占用。

3） 没有使用 GRU 作为对比。

参考文献

[1] https://github.com/udacity/cn-machine-learning/tree/master/mathematical_expression_recognition

[2] https://arxiv.org/abs/1507.05717

[3] https://en.wikipedia.org/wiki/Long_short-term_memory

[4] https://github.com/ypwhs/baiduyun_deeplearning_competition

[5] https://github.com/qjadud1994/CRNN-Keras

[6] https://github.com/xiaofengShi/CHINESE-OCR