人工智能入门

Keras 的 example 代码解析

Luke

May the force be with you.

https://blog.csdn.net/zhqh100/article/details/105145986

Keras 的 example 源码地址为：https://github.com/keras-team/keras/tree/master/examples
如有问题请访问 https://blog.csdn.net/zhqh100/article/details/105145986 交流
转载请注明出处

目录

keras 的 example 文件 addition_rnn.py 解析

该代码实现了通过神经网络来计算两个三位数的相加

先生成一堆训练数据，打印一下

print(questions[:10])
print(expected[:10])
结果为：

[' 31+991', ' 46+154', '   0+2', '   9+9', '   1+7', '   827+2', '   97+09', '   0+8', '   5+3', '   5+239']
['212 ', '515 ', '2   ', '18 ', '8   ', '730 ', '169 ', '8   ', '8   ', '937 ']
编码的时候，questions 是前面加空格，后面是真实的计算字符串，也就是右对齐

expected 是后面加空格，也就是说 expected 字符串是左对齐

然后进行编码，参考下面的 questions 编码方式

```
 31+991
[[ True False False False False False False False False False False False]
 [False False False False False  True False False False False False False]
 [False False False  True False False False False False False False False]
 [False  True False False False False False False False False False False]
 [False False False False False False False False False False False  True]
 [False False False False False False False False False False False  True]
 [False False False  True False False False False False False False False]]
 46+154
[[ True False False False False False False False False False False False]
 [False False False False False False  True False False False False False]
 [False False False False False False False False  True False False False]
 [False  True False False False False False False False False False False]
 [False False False  True False False False False False False False False]
 [False False False False False False False  True False False False False]
 [False False False False False False  True False False False False False]]
    0+2
[[ True False False False False False False False False False False False]
 [ True False False False False False False False False False False False]
 [ True False False False False False False False False False False False]
 [ True False False False False False False False False False False False]
 [False False  True False False False False False False False False False]
 [False  True False False False False False False False False False False]
 [False False False False  True False False False False False False False]]
```

https://blog.csdn.net/zhqh100/article/details/105145986

上面的一行，分别对应[空格, +, 0,1,2,3,4,5,6,7,8,9]，所以字符串进行了类似的 one-hot 编码

expected 也是一样：

```
212
[[False False False False    True False False False False False False False]
  [False False False    True False False False False False False False False]
  [False False False False    True False False False False False False False]
  [ True False False False False False False False False False False False]]
515
[[False False False False False False False    True False False False False]
  [False False False    True False False False False False False False False]
  [False False False False False False False    True False False False False]
  [ True False False False False False False False False False False False]]
2
[[False False False False    True False False False False False False False]
  [ True False False False False False False False False False False False]
  [ True False False False False False False False False False False False]
  [ True False False False False False False False False False False False]]
```

因为 expected 中没有加号，所以第二列永远为 False

x_train.shape 和 y_train.shape 分别为(45000, 7, 12) (45000, 4, 12)

神经网络模型为：

```
_________________________________________________________________________________
Layer (type)                          Output Shape                        Param
#
=================================================================================
lstm_1 (LSTM)                          (None, 128)                        72192
_________________________________________________________________________________
repeat_vector_1 (RepeatVector)        (None, 4, 128)                      0
_________________________________________________________________________________
lstm_2 (LSTM)                          (None, 4, 128)                     131584
_________________________________________________________________________________
time_distributed_1 (TimeDistributed)  (None, 4, 12)                       1548
=================================================================================
```

===========
Total params: 205,324
Trainable params: 205,324
Non-trainable params: 0
____
____
上面可以看到，两个 LSTM 的输出 shape 不一样，一个是(None, 128)，另一个是(None, 4, 128)，这是因为第一个 RNN 的 return_sequences 为 False，而第一个 RNN 的 return_sequences 为 True

代码解释参考官方教程：

https://keras.io/zh/examples/addition_rnn/

keras 的 example 文件 antirectifier.py 解析

该代码的功能是进行 mnist 的数字识别，主要是用于指导大家如何自己封装一个层，也就是自定义层

这里的 Antirectifier 就是自定义的一个层，代码是进行一个正则化，然后正向结果进行一个 relu 激活函数，和取反(负数)结果进行一个 relu，之后再进行一个 concatenate

输入 shape 和输出 shape 分别为：

(60000, 784)
(60000, 10)

神经网络结构为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 256)               200960
_________________________________________________________________
antirectifier_1 (Antirectifier) (None, 512)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
antirectifier_2 (Antirectifier) (None, 512)            0
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
```

https://blog.csdn.net/zhqh100/article/details/105145986

=
Total params: 337,418
Trainable params: 337,418
Non-trainable params: 0
____
_
不过可以看到，这里的 Antirectifier 层，参数个数为 0，所以没有参数需要训练

keras 的 example 文件 babi_memnn.py 解析

该代码功能是实现一个阅读理解的神经网络，就是给一段材料，提一个问题，然后看是否能给出答案；

首先这个代码有一个 bug，在 Python2 下应该可以运行，但是在 Python3 下会报错，

有人提交了 pull request，https://github.com/keras-team/keras/pull/13519/commits/3fc48bcd9a9cc931c43cf4e9e63ae35b61af8910，

但是官方对这个工程已经不咋用心了，所以至今还未合并，

可以把第 37 行

return [x.strip() for x in re.split(r'(\W+)?', sent) if x.strip()]
中的问号去掉

数据集是 Facebook 的 babi 数据集，官方地址为 https://research.fb.com/downloads/babi/

我这人有一个毛病，就是看到一个名称，总喜欢搞懂这个名称本身是什么意思，不过搜了一下确实没有搜到，只是看到一个非官方猜测 https://www.quora.com/What-does-bAbI-stand-for， babi 这个名称的含义：

babi，官方的叫法其实是 bAbI， 发音，和意思，都是 baby，大致就是婴儿学习的意思，而把 baby，改为 bAbI，就是 geek 们把 AI 嵌入到了 baby 这个单词中，因为 AI 这两个字母刻意大写，而其余字符刻意小写

把数据集 https://s3.amazonaws.com/text-datasets/babi_tasks_1-20_v1-2.tar.gz 下载之后，解压缩，打开 qa1_single-supporting-fact_train.txt 就可以大致明白是怎么回事了，

如第一个示例

1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary? bathroom 1
1 和 2 是材料，3 是问题和答案

从这个数据集中，自己给每个单词和标点符号搞了一个编码：

{'.': 1, '?': 2, 'Daniel': 3, 'John': 4, 'Mary': 5, 'Sandra': 6, 'Where': 7, 'back': 8, 'bathroom': 9,

'bedroom': 10, 'garden': 11, 'hallway': 12, 'is': 13, 'journeyed': 14, 'kitchen': 15, 'moved': 16, 'office': 17, 'the': 18, 'to': 19, 'travelled': 20, 'went': 21}
可以看到总共是 21 个编码，再加上补齐的 pad，数值为 0，那总共就是 22 个编码

然后编码每个材料，问题，和答案：

(['Mary', 'moved', 'to', 'the', 'bathroom', '.', 'John', 'went', 'to', 'the', 'hallway', '.'], ['Where', 'is', 'Mary', '?'], 'bathroom')

```
[0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  5 16 19 18  9  1  4 21 19 18 12  1]
[ 7 13  5  2]
9
```

0 代码啥也没有，pad 的字符

输入的 shape

inputs_train shape: (10000, 68)
queries_train shape: (10000, 4)
answers_train shape: (10000,)
神经网络结构

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 68)           0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 4)            0
__________________________________________________________________________________________________
sequential_1 (Sequential)       multiple             1408        input_1[0][0]
__________________________________________________________________________________________________
sequential_3 (Sequential)       (None, 4, 64)        1408        input_2[0][0]
__________________________________________________________________________________________________
dot_1 (Dot)                     (None, 68, 4)        0           sequential_1[1][0]
                                                                 sequential_3[1][0]
```

https://blog.csdn.net/zhqh100/article/details/105145986

```
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 68, 4)        0           dot_1[0][0]
__________________________________________________________________________________________________
sequential_2 (Sequential)       multiple             88          input_1[0][0]
__________________________________________________________________________________________________
add_1 (Add)                     (None, 68, 4)        0           activation_1[0][0]

sequential_2[1][0]
__________________________________________________________________________________________________
permute_1 (Permute)             (None, 4, 68)        0           add_1[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 4, 132)       0           permute_1[0][0]

sequential_3[1][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 32)           21120       concatenate_1[0][0]
__________________________________________________________________________________________________
dropout_4 (Dropout)             (None, 32)           0           lstm_1[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 22)           726         dropout_4[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 22)           0           dense_1[0][0]
==================================================================================================
Total params: 24,750
Trainable params: 24,750
Non-trainable params: 0
__________________________________________________________________________________________________
```

keras 的 example 文件 babi_rnn.py 解析

该代码的目的和 https://blog.csdn.net/zhqh100/article/details/105193991 类似

数据集也是同一个数据集，只不过这个是从 qa2_two-supporting-facts_train.txt 中获取的文本，文本量会大一些

第一个示例

1 Mary moved to the bathroom.
2 Sandra journeyed to the bedroom.
3 Mary got the football there.
4 John went to the kitchen.
5 Mary went back to the kitchen.
6 Mary went back to the garden.
7 Where is the football?	garden	3 6

单词映射为：

{'.': 1, '?': 2, 'Daniel': 3, 'John': 4, 'Mary': 5, 'Sandra': 6, 'Where': 7, 'apple': 8, 'back': 9, 'bathroom': 10, 'bedroom': 11, 'discarded': 12, 'down': 13, 'dropped': 14, 'football': 15, 'garden': 16, 'got': 17, 'grabbed': 18, 'hallway': 19, 'is': 20, 'journeyed': 21, 'kitchen': 22, 'left': 23, 'milk': 24, 'moved': 25, 'office': 26, 'picked': 27, 'put': 28, 'the': 29, 'there': 30, 'to': 31, 'took': 32, 'travelled': 33, 'up': 34, 'went': 35}

上面的材料编码后为：

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0  5 25 31 29 10   1   6 21 31 29 11   1   5 17
 29 15 30   1   4 35 31 29 22   1   5 35   9 31 29 22   1   5 35   9 31 29 16   1]
[ 7 20 29 15   2]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

这里把 ans 进行了 one-hot 编码，所以 loss 用的是 categorical_crossentropy，而 babi_memnn.py 用的是 sparse_categorical_crossentropy，所以不用进行 one-hot 编码

训练数据 shape

x.shape = (1000, 552)
xq.shape = (1000, 5)
y.shape = (1000, 36)
神经网络结构：

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 552)          0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 5)            0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 552, 50)      1800        input_1[0][0]
__________________________________________________________________________________________________
embedding_2 (Embedding)         (None, 5, 50)        1800        input_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   (None, 100)          60400       embedding_1[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   (None, 100)          60400       embedding_2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 200)          0           lstm_1[0][0]
                                                                 lstm_2[0][0]
__________________________________________________________________________________________________
```

```
____________________
dense_1 (Dense)                 (None, 36)           7236        concatenate_1[0][0]
==================================================================================
==================
Total params: 131,636
Trainable params: 131,636
Non-trainable params: 0
__________________________________________________________________________________
____________________
```

keras 的 example 文件 cifar10_cnn.py 解析

这个示例很简单，就是从 cifar10 中读取数据集，通过卷积神经网络进行图像识别

输入数据的 shape

x_train.shape (50000, 32, 32, 3)
y_train.shape (50000, 10)

神经网络结构：

```
_________________________________________________________________
_
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
_
activation_1 (Activation)    (None, 32, 32, 32)        0
_________________________________________________________________
_
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
_
activation_2 (Activation)    (None, 30, 30, 32)        0
_________________________________________________________________
_
max_pooling2d_1 (MaxPooling2D) (None, 15, 15, 32)      0
_________________________________________________________________
_
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
_
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
_
activation_3 (Activation)    (None, 15, 15, 64)        0
_________________________________________________________________
_
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
```

```
________________________________________________________________________________
_
activation_4 (Activation)        (None, 13, 13, 64)        0
________________________________________________________________________________
_
max_pooling2d_2 (MaxPooling2D)   (None, 6, 6, 64)          0
________________________________________________________________________________
_
dropout_2 (Dropout)              (None, 6, 6, 64)          0
________________________________________________________________________________
_
flatten_1 (Flatten)              (None, 2304)              0
________________________________________________________________________________
_
dense_1 (Dense)                  (None, 512)               1180160
________________________________________________________________________________
_
activation_5 (Activation)        (None, 512)               0
________________________________________________________________________________
_
dropout_3 (Dropout)              (None, 512)               0
________________________________________________________________________________
_
dense_2 (Dense)                  (None, 10)                5130
________________________________________________________________________________
_
activation_6 (Activation)        (None, 10)                0
================================================================================
=
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
________________________________________________________________________________
_
```

代码同时演示了 ImageDataGenerator 的使用

keras 的 example 文件 cifar10_resnet.py 解析

该代码功能是卷积神经网络进行图像识别，数据集是 cifar10

同时演示了回调函数 ModelCheckpoint， LearningRateScheduler， ReduceLROnPlateau 的用法

输入数据的 shape

x_train shape: (50000, 32, 32, 3)
y_train shape: (50000, 1)
默认的神经网络结构：

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 32, 32, 3)    0
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 32, 32, 16)   448         input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNormalization) (None, 32, 32, 16)   64    conv2d_1[0][0]
__________________________________________________________________________________________________
activation_1 (Activation)       (None, 32, 32, 16)   0           batch_normalization_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 32, 32, 16)   2320        activation_1[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNormalization) (None, 32, 32, 16)   64    conv2d_2[0][0]
__________________________________________________________________________________________________
activation_2 (Activation)       (None, 32, 32, 16)   0
```

batch_normalization_2[0][0]

____

____

conv2d_3 (Conv2D) (None, 32, 32, 16) 2320
activation_2[0][0]

____

____

batch_normalization_3 (BatchNormalization) (None, 32, 32, 16) 64
conv2d_3[0][0]

____

____

add_1 (Add) (None, 32, 32, 16) 0
activation_1[0][0]

batch_normalization_3[0][0]

____

____

activation_3 (Activation) (None, 32, 32, 16) 0
add_1[0][0]

____

____

conv2d_4 (Conv2D) (None, 32, 32, 16) 2320
activation_3[0][0]

____

____

batch_normalization_4 (BatchNormalization) (None, 32, 32, 16) 64
conv2d_4[0][0]

____

____

activation_4 (Activation) (None, 32, 32, 16) 0
batch_normalization_4[0][0]

____

____

conv2d_5 (Conv2D) (None, 32, 32, 16) 2320
activation_4[0][0]

____

____

batch_normalization_5 (BatchNormalization) (None, 32, 32, 16) 64
conv2d_5[0][0]

____

____

add_2 (Add) (None, 32, 32, 16) 0
activation_3[0][0]

batch_normalization_5[0][0]

____

activation_5 (Activation) (None, 32, 32, 16) 0
add_2[0][0]

____

conv2d_6 (Conv2D) (None, 32, 32, 16) 2320
activation_5[0][0]

____

batch_normalization_6 (BatchNormalization) (None, 32, 32, 16) 64
conv2d_6[0][0]

____

activation_6 (Activation) (None, 32, 32, 16) 0
batch_normalization_6[0][0]

____

conv2d_7 (Conv2D) (None, 32, 32, 16) 2320
activation_6[0][0]

____

batch_normalization_7 (BatchNormalization) (None, 32, 32, 16) 64
conv2d_7[0][0]

____

add_3 (Add) (None, 32, 32, 16) 0
activation_5[0][0]

batch_normalization_7[0][0]

____

activation_7 (Activation) (None, 32, 32, 16) 0
add_3[0][0]

____

conv2d_8 (Conv2D) (None, 16, 16, 32) 4640
activation_7[0][0]

____

batch_normalization_8 (BatchNormalization) (None, 16, 16, 32) 128
conv2d_8[0][0]

____

activation_8 (Activation) (None, 16, 16, 32) 0
batch_normalization_8[0][0]

conv2d_9 (Conv2D) (None, 16, 16, 32) 9248
activation_8[0][0]

conv2d_10 (Conv2D) (None, 16, 16, 32) 544
activation_7[0][0]

batch_normalization_9 (BatchNormalization) (None, 16, 16, 32) 128
conv2d_9[0][0]

add_4 (Add) (None, 16, 16, 32) 0
conv2d_10[0][0]

batch_normalization_9[0][0]

activation_9 (Activation) (None, 16, 16, 32) 0
add_4[0][0]

conv2d_11 (Conv2D) (None, 16, 16, 32) 9248
activation_9[0][0]

batch_normalization_10 (BatchNormalization) (None, 16, 16, 32) 128
conv2d_11[0][0]

activation_10 (Activation) (None, 16, 16, 32) 0
batch_normalization_10[0][0]

conv2d_12 (Conv2D) (None, 16, 16, 32) 9248
activation_10[0][0]

batch_normalization_11 (BatchNormalization) (None, 16, 16, 32) 128 conv2d_12[0][0]

____

____

add_5 (Add) (None, 16, 16, 32) 0
activation_9[0][0]

batch_normalization_11[0][0]

____

____

activation_11 (Activation) (None, 16, 16, 32) 0
add_5[0][0]

____

____

conv2d_13 (Conv2D) (None, 16, 16, 32) 9248
activation_11[0][0]

____

____

batch_normalization_12 (BatchNormalization) (None, 16, 16, 32) 128
conv2d_13[0][0]

____

____

activation_12 (Activation) (None, 16, 16, 32) 0
batch_normalization_12[0][0]

____

____

conv2d_14 (Conv2D) (None, 16, 16, 32) 9248
activation_12[0][0]

____

____

batch_normalization_13 (BatchNormalization) (None, 16, 16, 32) 128
conv2d_14[0][0]

____

____

add_6 (Add) (None, 16, 16, 32) 0
activation_11[0][0]

batch_normalization_13[0][0]

____

____

activation_13 (Activation) (None, 16, 16, 32) 0
add_6[0][0]

____

____

conv2d_15 (Conv2D) (None, 8, 8, 64) 18496 activation_13[0][0]

batch_normalization_14 (BatchNormalization) (None, 8, 8, 64) 256
conv2d_15[0][0]

activation_14 (Activation) (None, 8, 8, 64) 0
batch_normalization_14[0][0]

conv2d_16 (Conv2D) (None, 8, 8, 64) 36928
activation_14[0][0]

conv2d_17 (Conv2D) (None, 8, 8, 64) 2112
activation_13[0][0]

batch_normalization_15 (BatchNormalization) (None, 8, 8, 64) 256
conv2d_16[0][0]

add_7 (Add) (None, 8, 8, 64) 0
conv2d_17[0][0]

batch_normalization_15[0][0]

activation_15 (Activation) (None, 8, 8, 64) 0
add_7[0][0]

conv2d_18 (Conv2D) (None, 8, 8, 64) 36928
activation_15[0][0]

batch_normalization_16 (BatchNormalization) (None, 8, 8, 64) 256
conv2d_18[0][0]

activation_16 (Activation) (None, 8, 8, 64) 0
batch_normalization_16[0][0]

____
conv2d_19 (Conv2D) (None, 8, 8, 64) 36928
activation_16[0][0]
____
____
batch_normalization_17 (BatchNormalization) (None, 8, 8, 64) 256
conv2d_19[0][0]
____
____
add_8 (Add) (None, 8, 8, 64) 0
activation_15[0][0]

batch_normalization_17[0][0]
____
____
activation_17 (Activation) (None, 8, 8, 64) 0
add_8[0][0]
____
____
conv2d_20 (Conv2D) (None, 8, 8, 64) 36928
activation_17[0][0]
____
____
batch_normalization_18 (BatchNormalization) (None, 8, 8, 64) 256
conv2d_20[0][0]
____
____
activation_18 (Activation) (None, 8, 8, 64) 0
batch_normalization_18[0][0]
____
____
conv2d_21 (Conv2D) (None, 8, 8, 64) 36928
activation_18[0][0]
____
____
batch_normalization_19 (BatchNormalization) (None, 8, 8, 64) 256
conv2d_21[0][0]
____
____
add_9 (Add) (None, 8, 8, 64) 0
activation_17[0][0]

batch_normalization_19[0][0]
____

____

activation_19 (Activation) (None, 8, 8, 64) 0
add_9[0][0]

____

____

average_pooling2d_1 (AveragePooling2D) (None, 1, 1, 64) 0
activation_19[0][0]

____

____

flatten_1 (Flatten) (None, 64) 0
average_pooling2d_1[0][0]

____

____

dense_1 (Dense) (None, 10) 650
flatten_1[0][0]

==
==

Total params: 274,442
Trainable params: 273,066
Non-trainable params: 1,376

____

____

按说 ReduceLROnPlateau 和 LearningRateScheduler 都可以调整学习率，但是两个同时用就很奇怪，下面添加一个很无聊的日志打印，可以看到，在这个演示中， ReduceLROnPlateau 没有机会起到作用，实际学习率被 LearningRateScheduler 掌控了

日志太长了，如需参考可访问 https://blog.csdn.net/zhqh100/article/details/105198673

keras 的 example 文件 class_activation_maps.py 解析

该文件功能是实现一个 CNN 可视化，我不知道这个叫法算不算专业，可以参考别人写的文章

https://zhuanlan.zhihu.com/p/51631163

https://blog.csdn.net/weixin_40955254/article/details/81191896

应该就是看下神经网络是通过关注哪部分区域预测出的最终结果

其基础网络是 resnet50，get_cam_model 函数中对其进行了一点修改，可参考下图，看到修改了哪些地方

就是仅添加了两层神经网络，其他没有动

预测后通过 postprocess 函数提取出感兴趣的那个分类的 channel，我把该函数简化了一下，应该更容易理解一点

```
def postprocess(preds, cams, top_k=1):
    idxes = np.argsort(preds[0])[-top_k:][0]
    return cams[0, :, :, idxes]
```

文件最后的代码，我作了如下修改：

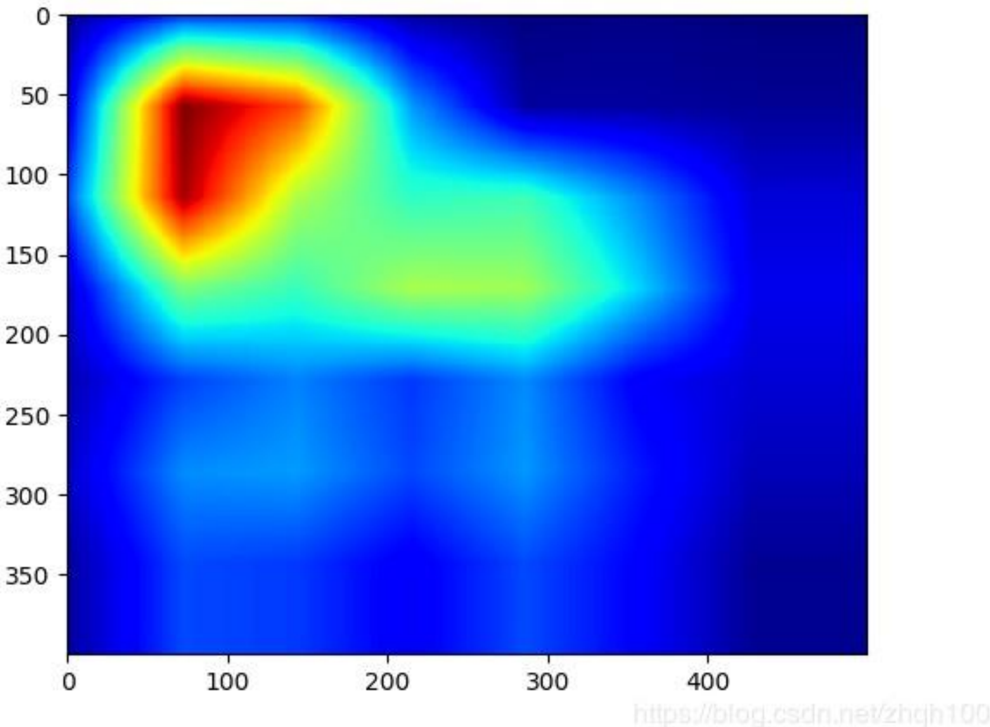
```
# 5. plot image+cam to original size
# plt.imshow(original_img, alpha=0.5)
plt.imshow(cv2.resize(class_activation_map,
                      original_size), cmap='jet')
plt.savefig("res.jpg")
plt.show()
```

看下效果图

原图

https://blog.csdn.net/zhqh100/article/details/105145986

预测后结果的颜色图谱

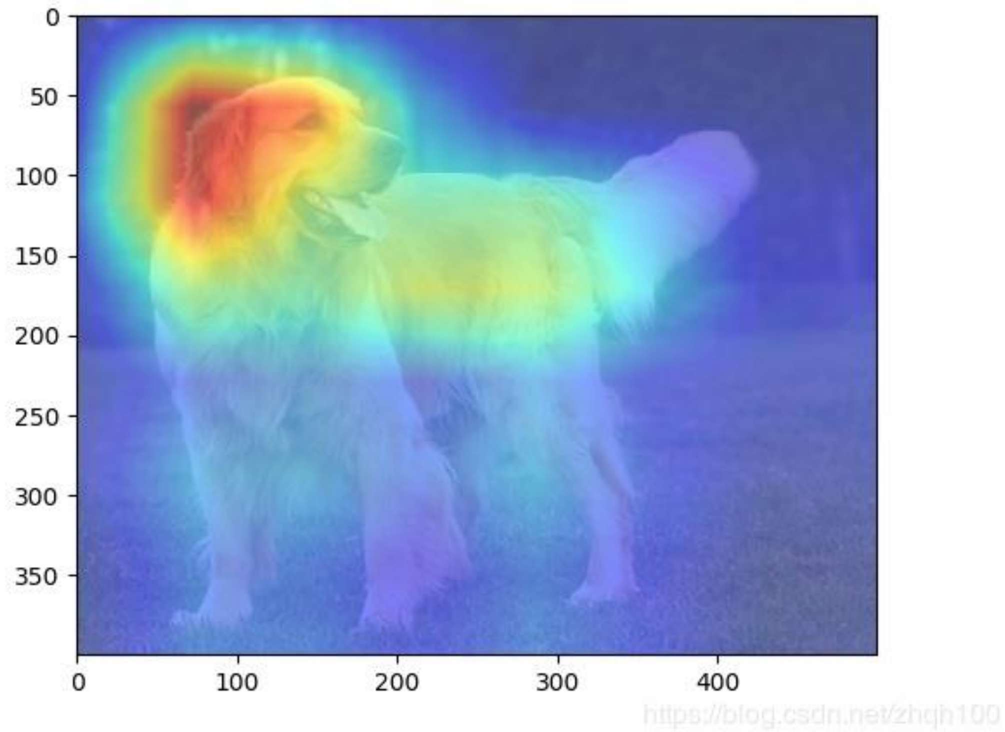


https://blog.csdn.net/zhqh100/article/details/105145986

也就是说，他是根据这部分区域预测出的结果，这是一只金毛犬，下面可以把分类名称打印出来

```
from keras.applications.imagenet_utils import decode_predictions
print(decode_predictions(preds))
[[('n02099601', 'golden_retriever', 0.82736635), ('n02111129', 'Leonberg', 0.08901908), ('n02111277', 'Newfoundland', 0.008406928), ('n02104029', 'kuvasz', 0.007829149), ('n02099712', 'Labrador_retriever', 0.004909024)]]
```

如果最后面代码没有修改的话，显示结果为：

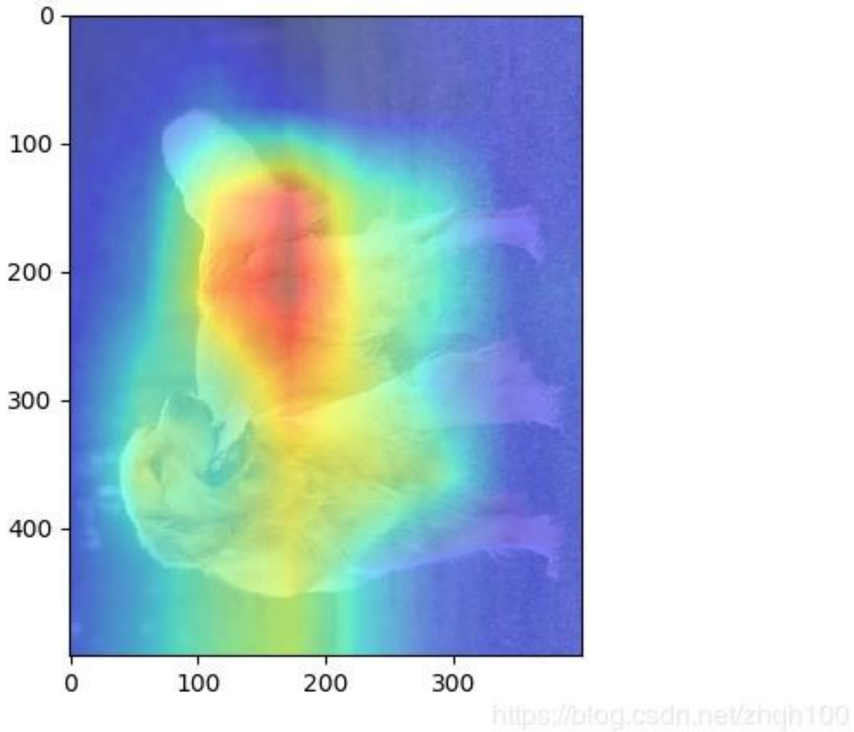


就是和原图进行了一个叠加，显示出，哪部分区域，是 CNN 得出最终预测结果的主要依据

————————————————————————————

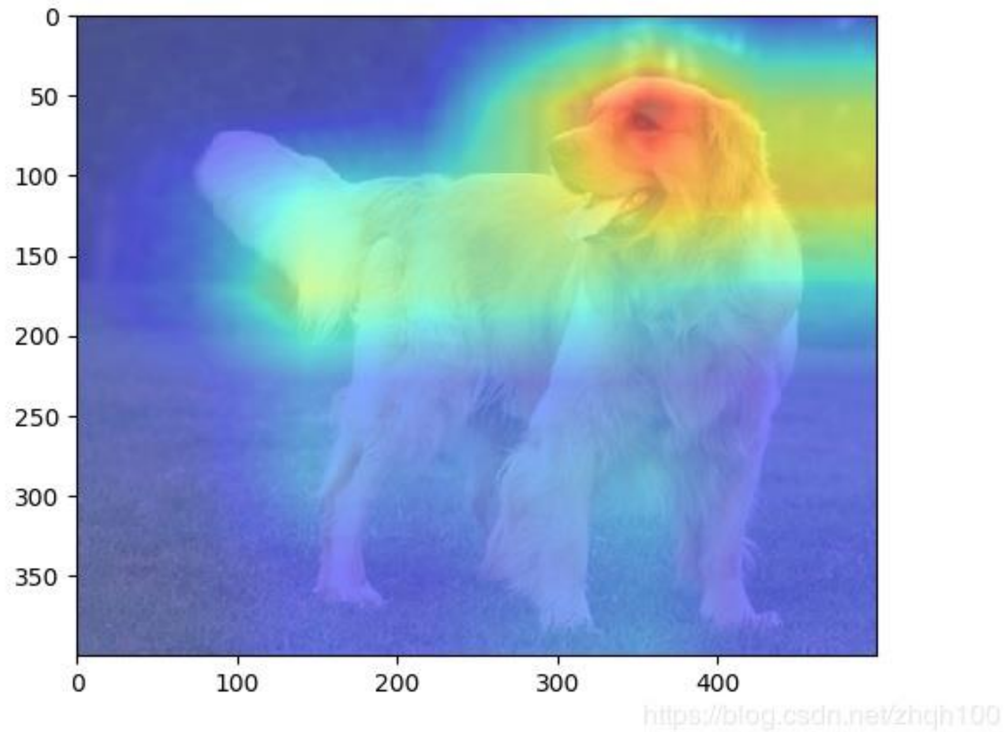
下面是自己折腾时刻，不属于正文

如果我把图片旋转一下，发现 resnet 几乎无法识别了



[[('n02099601', 'golden_retriever', 0.2694396), ('n02113023', 'Pembroke', 0.16751677), ('n01514859', 'hen', 0.10281576), ('n02105251', 'briard', 0.03323859), ('n02102318', 'cocker_spaniel', 0.03284181)]]

当然，如果水平翻转，还是能识别到的，而且关注点也可以正确的找到



[[('n02099601', 'golden_retriever', 0.82862425), ('n02111129', 'Leonberg', 0.07444748), ('n02104029', 'kuvasz', 0.01596525), ('n02111277', 'Newfoundland', 0.015939327), ('n02099267', 'flat-coated_retriever', 0.0076823044)]]

附修改后的神经网络

如需参考请访问： https://blog.csdn.net/zhqh100/article/details/105222377

keras 的 example 文件 cnn_seq2seq.py 解析

该代码是实现一个翻译功能，好像是英语翻译为法语，嗯，我看不懂法语

首先这个代码有一个 bug，本人提交了一个 pull request 来修复，

https://github.com/keras-team/keras/pull/13863/commits/fd44e03a9d17c05aaecc620f8d88ef0fd385254b

但由于官方长久不维护，所以至今尚未合并，

需要把第 68 行改为：

input_text, target_text, _ = line.split('\t')
然后根据训练数据，对字母进行编码，其中 target_token_index 中添加了两个字符，开始符号 '\t' 和结束符合 '\n'：

print(input_token_index)
{' ': 0, '!': 1, '$': 2, '%': 3, '&': 4, "'": 5, ',': 6, '-': 7, '.': 8, '0': 9, '1': 10, '2': 11, '3': 12, '5': 13, '6': 14, '7': 15, '8': 16, '9': 17, ':': 18, '?': 19, 'A': 20, 'B': 21, 'C': 22, 'D': 23, 'E': 24, 'F': 25, 'G': 26, 'H': 27, 'I': 28, 'J': 29, 'K': 30, 'L': 31, 'M': 32, 'N': 33, 'O': 34, 'P': 35, 'Q': 36, 'R': 37, 'S': 38, 'T': 39, 'U': 40, 'V': 41, 'W': 42, 'Y': 43, 'a': 44, 'b': 45, 'c': 46, 'd': 47, 'e': 48, 'f': 49, 'g': 50, 'h': 51, 'i': 52, 'j': 53, 'k': 54, 'l': 55, 'm': 56, 'n': 57, 'o': 58, 'p': 59, 'q': 60, 'r': 61, 's': 62, 't': 63, 'u': 64, 'v': 65, 'w': 66, 'x': 67, 'y': 68, 'z': 69}
print(target_token_index)
{'\t': 0, '\n': 1, ' ': 2, '!': 3, '$': 4, '%': 5, '&': 6, "'": 7, '(': 8, ')': 9, ',': 10, '-': 11, '.': 12, '0': 13, '1': 14, '2': 15, '3': 16, '5': 17, '8': 18, '9': 19, ':': 20, '?': 21, 'A': 22, 'B': 23, 'C': 24, 'D': 25, 'E': 26, 'F': 27, 'G': 28, 'H': 29, 'I': 30, 'J': 31, 'K': 32, 'L': 33, 'M': 34, 'N': 35, 'O': 36, 'P': 37, 'Q': 38, 'R': 39, 'S': 40, 'T': 41, 'U': 42, 'V': 43, 'Y': 44, 'a': 45, 'b': 46, 'c': 47, 'd': 48, 'e': 49, 'f': 50, 'g': 51, 'h': 52, 'i': 53, 'j': 54, 'k': 55, 'l': 56, 'm': 57, 'n': 58, 'o': 59, 'p': 60, 'q': 61, 'r': 62, 's': 63, 't': 64, 'u': 65, 'v': 66, 'x': 67, 'y': 68, 'z': 69, '\xa0': 70, '«': 71, '»': 72, 'À': 73, 'Ç': 74, 'É': 75, 'Ê': 76, 'à': 77, 'â': 78, 'ç': 79, 'è': 80, 'é': 81, 'ê': 82, 'ë': 83, 'î': 84, 'ï': 85, 'ô': 86, 'ù': 87, 'û': 88, 'œ': 89, '\u2009': 90, '’': 91, '\u202f': 92}
对，这个演示示例中不是对 word 进行编码，而是对字母进行编码，

至于原因，我分析应该是这样的，字母数量比较少，这个索引数也不过只有 70 个而已，但如果对单词进行编码，那随随便便就上千个，维度超大，后面再运算的时候，需要占用极大的内存和 GPU

然后对输入输出的句子手动进行 one-hot 编码：

在预处理中，target_text 的首位补了一个'\t'，代表句子开始了，末尾补了一个'\n'，代表句

子结束了

输入数据的尺寸为：

encoder_input_data.shape (10000, 16, 70)
decoder_input_data.shape (10000, 59, 93)
decoder_target_data.shape (10000, 59, 93)

而这个 decoder_input_data 和 decoder_target_data 都是翻译后的句子，只不过 decoder_target_data 比 decoder_input_data 提前一位，decoder_input_data 的第一位是 '\t'， 第二位才是真实内容，而 decoder_target_data 的第一位直接就是真实内容了。

为什么会把翻译的结果作为模型的输入？

因为在训练模型时，下一位的输出会依赖上一位的值，而在神经网络最开始的时候，如果预测的第一位错了，在预测第二位的时候，就会有一个错误的输入，我们这时候根据一个错误的输入去优化神经网络是走在了错误的方向，所以我们会辅助提供一个正确的值，这样神经网络才是向正确的方向优化

神经网络结构

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_2 (InputLayer)            (None, None, 93)     0
__________________________________________________________________________________________________
input_1 (InputLayer)            (None, None, 70)     0
__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, None, 256)    71680       input_2[0][0]
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, None, 256)    54016       input_1[0][0]
__________________________________________________________________________________________________
conv1d_5 (Conv1D)               (None, None, 256)    196864      conv1d_4[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, None, 256)    196864      conv1d_1[0][0]
```

```
____________________________________________________________________________________________________
conv1d_6 (Conv1D)               (None, None, 256)    196864      conv1d_5[0][0]
____________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, None, 256)    196864      conv1d_2[0][0]
____________________________________________________________________________________________________
dot_1 (Dot)                     (None, None, None)   0           conv1d_6[0][0]
                                                                 conv1d_3[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)       (None, None, None)   0           dot_1[0][0]
____________________________________________________________________________________________________
dot_2 (Dot)                     (None, None, 256)    0           activation_1[0][0]
                                                                 conv1d_3[0][0]
____________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, None, 512)    0           dot_2[0][0]
                                                                 conv1d_6[0][0]
____________________________________________________________________________________________________
conv1d_7 (Conv1D)               (None, None, 64)     98368       concatenate_1[0][0]
____________________________________________________________________________________________________
conv1d_8 (Conv1D)               (None, None, 64)     12352       conv1d_7[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                 (None, None, 93)     6045        conv1d_8[0][0]
====================================================================================================
Total params: 1,029,917
Trainable params: 1,029,917
Non-trainable params: 0
____________________________________________________________________________________________________
```

在预测的时候，encoder_input_data 就是输入的句子，decoder_input_data 是一个除第一位设置为开始符号'\t'外，其余位均为 0 的结构，在预测出第一位 decoder_target_data 后，把预测的字符追加到 decoder_input_data 后面一位，然后通过 for 循环预测下一位，以此类推，直到预期长度

因为预测出的结果为编号，需要反向索引为字符，而在反向索引时如果遇到结束符 '\n'，就表示句子结束，得到了完整的预测结果

https://blog.csdn.net/zhqh100/article/details/105145986

———————————————————————————————

代码 lstm_seq2seq.py 的数据预处理和上面一致，就不另外写一篇了，神经网络结构为：

lstm_seq2seq.py 神经网络结构

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape                     Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, None, 70)                 0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, None, 93)                 0
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 256), (None, 256), (None, 256)]    334848      input_1[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, None, 256), (None, 256), (None, 256)]    358400      input_2[0][0]
                                                                                             lstm_1[0][1]
                                                                                             lstm_1[0][2]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, None, 93)                 23901       lstm_2[0][0]
==================================================================================================
Total params: 717,149
Trainable params: 717,149
Non-trainable params: 0
__________________________________________________________________________________________________
```

https://blog.csdn.net/zhqh100/article/details/105145986

keras 的 example 文件 conv_lstm.py 解析

该文件演示了 ConvLSTM2D 和 Conv3D 的使用，

他的网络结构打印出来为

```
_________________________________________________________________________________
_____________________
Layer (type)                                                      Output Shape
Param #
=================================================================================
=====================
conv_lst_m2d_1 (ConvLSTM2D)                            (None, None, 40, 40, 40)
59200
_________________________________________________________________________________
_____________________
batch_normalization_1 (BatchNormalization)    (None, None, 40, 40, 40)          160
_________________________________________________________________________________
_____________________
conv_lst_m2d_2 (ConvLSTM2D)                            (None, None, 40, 40, 40)
115360
_________________________________________________________________________________
_____________________
batch_normalization_2 (BatchNormalization)    (None, None, 40, 40, 40)          160
_________________________________________________________________________________
_____________________
conv_lst_m2d_3 (ConvLSTM2D)                            (None, None, 40, 40, 40)
115360
_________________________________________________________________________________
_____________________
batch_normalization_3 (BatchNormalization)    (None, None, 40, 40, 40)          160
_________________________________________________________________________________
_____________________
conv_lst_m2d_4 (ConvLSTM2D)                            (None, None, 40, 40, 40)
115360
_________________________________________________________________________________
_____________________
batch_normalization_4 (BatchNormalization)    (None, None, 40, 40, 40)          160
_________________________________________________________________________________
_____________________
conv3d_1 (Conv3D)                                         (None, None, 40, 40, 1)
1081
=================================================================================
```

https://blog.csdn.net/zhqh100/article/details/105145986

====================
Total params: 407,001
Trainable params: 406,681
Non-trainable params: 320

____

____

其输入和输出分别为 noisy_movies 和 shifted_movies，也就是两段电影，影片内容是用代码生成的移动方框，如下

只要在代码中添加如下两行，即可保存一段影片：

```
import imageio
imageio.mimsave("my.gif", shifted_movies[3], 'GIF', duration=0.2)
```

而 noisy_movies 和 shifted_movies 的 shape 均为(1200, 15, 40, 40, 1)，

也就是包含 1200 个影片，每个影片有 15 帧，分辨率为 40*40，

noisy_movies 和 shifted_movies 影片内容有什么关系呢？

其实 shifted_movies 是 noisy_movies 的每一帧的下一帧，只不过有一点点噪音而已

如果把代码中的

```
            if np.random.randint(0, 2):
                noise_f = (-1)**np.random.randint(0, 2)
                noisy_movies[i, t,
                             x_shift - w - 1: x_shift + w + 1,
                             y_shift - w - 1: y_shift + w + 1,
                             0] += noise_f * 0.1
```

这段注释掉，然后在下面添加判断

```
for k in range(100):
    print(k)
    for i in range(1, 14):
        print((shifted_movies[k][i - 1].astype(np.uint8)==noisy_movies[k][i].astype(np.uint8)).all())
        # cv2.imshow("noisy", noisy_movies[k][i])
        # cv2.imshow("shift", shifted_movies[k][i - 1])
        # cv2.waitKey(1000)
```

我们就可以看到，这个判断永远为 True，所以该代码逻辑就是，

给一段影片，预测其下一帧，可能还带一点影片清晰度的修复(消除噪音)

https://blog.csdn.net/zhqh100/article/details/105145986

keras 的 example 文件 deep_dream.py 解析

这个程序大致就是让神经网络产生一些梦境般的效果，把实实在在的画面搞的花里胡哨，虚虚实实，

这里只是分析下代码，原理的话，可以参考下

https://blog.csdn.net/accepthjp/article/details/77882814

我用上次的那条狗，生成的效果大致是这样的，

《略》

抱歉，不是图丢了，是我把图删了，可能有人觉得生成的图片如梦如幻，但我看了有点起鸡皮疙瘩

基础神经网络使用的是 inception_v3
我们从代码

img = preprocess_image(base_image_path)
开始看，这里就是正常的读取一张图片，只不过为了 batch 处理，加了一个维度，其他没有变化

下面是计算 successive_shapes，就是很简单的计算出了两个尺寸，一个是原图尺寸的 1/1.4，另一个是原图尺寸的 1/(1.4**2)

如我原图尺寸为 400*500，这里就是计算出 285*356，再计算出 204*255，也就是

successive_shapes [(204, 255), (285, 357), (400, 500)]
然后下面的代码可以精简为：

```
for shape in successive_shapes:
    print('Processing image shape', shape)
    img = resize_img(img, shape)
    img = gradient_ascent(img,
                          iterations=iterations,
                          step=step,
                          max_loss=max_loss)
```

因为其余代码是计算一个 lost_detail，是为了对缩放产生的一点点细节丢失进行一点点补偿，也就是不加也能产生效果，只不过会有一点点模糊，所以我们先不关心

这段代码是先对图片缩放到最小的那个尺寸，如我这里是(204,255)，然后调用 gradient_ascent， gradient_ascent 会调用 eval_loss_and_grads，eval_loss_and_grads 会调

用 fetch_loss_and_grads， 总之就是计算一个 loss，和一个 loss 对应的梯度

loss 是 inception_v3 的 mixed2， mixed3， mixed4， mixed5 基层的输出的值(四周各去掉两个像素)的平方和再除以总数,再乘以一个系数，再相加；所以 loss 永远为正数；

梯度进行一个正则化(就是除以一个绝对值最大的那个数，缩放到 < 1.0)，然后再乘以一个 0.01，之后，就和原图加起来了，这样就对原图进行了一个修改；

而我们思考一下一个凸函数，或者凹函数，一个 y 值，对 x 在某个点的导数，如果再加回到 x 上去的话，那就是向 y 变得更大的方向移动；

我们这里的 x 就是 img，y 值就是 loss，所以 loss 会越来越大；

然后再 for 循环计算，缩放之后再进行几次计算

代码基本逻辑就是这样，估计看了之后还是不明白，这，有什么意义吗？

我想这里确实看不出什么意义，主要是这里的 loss 是一个制定的有点随心所欲的一个值，

而如果我们把 loss 值定为一个像狗的指标，而如果我们以此方法进行不停的反向计算的话，即便输入是一条鱼，我们也能够把原图迭代修改为一个像狗一样的鱼，有机会了可以试一下

我稍微改了一下，可以把原图识别为狗的图片，修改为识别为猫，图片基本变化不大，和原图一样，只是原先识别为 207，即金毛犬，但是在把原图修改后，识别为 283 了，即波斯猫：

```
from __future__ import print_function

from keras.preprocessing.image import load_img, save_img, img_to_array
import numpy as np
import scipy
import argparse

from keras.applications import inception_v3
from keras import backend as K

parser = argparse.ArgumentParser(description='Deep Dreams with Keras.')
parser.add_argument('base_image_path', metavar='base', type=str,
                    help='Path to the image to transform.')
parser.add_argument('result_prefix', metavar='res_prefix', type=str,
                    help='Prefix for the saved results.')
```

```
args = parser.parse_args()
base_image_path = args.base_image_path
result_prefix = args.result_prefix

# These are the names of the layers
# for which we try to maximize activation,
# as well as their weight in the final loss
# we try to maximize.
# You can tweak these setting to obtain new visual effects.
settings = {
    'features': {
        'mixed2': 0.2,
        'mixed3': 0.5,
        'mixed4': 2.,
        'mixed5': 1.5,
    },
}


def preprocess_image(image_path):
    # Util function to open, resize and format pictures
    # into appropriate tensors.
    img = load_img(image_path)
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = inception_v3.preprocess_input(img)
    return img


def deprocess_image(x):
    # Util function to convert a tensor into a valid image.
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, x.shape[2], x.shape[3]))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((x.shape[1], x.shape[2], 3))
    x /= 2.
    x += 0.5
    x *= 255.
    x = np.clip(x, 0, 255).astype('uint8')
    return x

K.set_learning_phase(0)
```

```
# Build the InceptionV3 network with our placeholder.
# The model will be loaded with pre-trained ImageNet weights.
model = inception_v3.InceptionV3()
dream = model.input

print('Model loaded.')

# Get the symbolic outputs of each "key" layer (we gave them unique names).
layer_dict = dict([(layer.name, layer) for layer in model.layers])

# Define the loss.
loss = layer_dict['predictions'].output[0][283]


# Compute the gradients of the dream wrt the loss.
grads = K.gradients(loss, dream)[0]
# Normalize gradients.
grads /= K.maximum(K.mean(K.abs(grads)), K.epsilon())

# Set up function to retrieve the value
# of the loss and gradients given an input image.
outputs = [loss, grads]
fetch_loss_and_grads = K.function([dream], outputs)


def eval_loss_and_grads(x):
    outs = fetch_loss_and_grads([x])
    loss_value = outs[0]
    grad_values = outs[1]
    return loss_value, grad_values


def resize_img(img, size):
    img = np.copy(img)
    if K.image_data_format() == 'channels_first':
        factors = (1, 1,
                   float(size[0]) / img.shape[2],
                   float(size[1]) / img.shape[3])
    else:
        factors = (1,
                   float(size[0]) / img.shape[1],
                   float(size[1]) / img.shape[2],
                   1)
```

```
    return scipy.ndimage.zoom(img, factors, order=1)


def gradient_ascent(x, iterations, step, max_loss=None):
    for i in range(iterations):
        loss_value, grad_values = eval_loss_and_grads(x)
        if max_loss is not None and loss_value > max_loss:
            break
        print('..Loss value at', i, ':', loss_value)
        x += step * grad_values
    return x

# Playing with these hyperparameters will also allow you to achieve new effects
step = 0.0001    # Gradient ascent step size
num_octave = 3    # Number of scales at which to run gradient ascent
octave_scale = 1.4    # Size ratio between scales
iterations = 2000    # Number of ascent steps per scale
max_loss = 0.9

img = preprocess_image(base_image_path)

shape=(299,299,3)

print('Processing image shape', shape)
img = resize_img(img, shape)
img = gradient_ascent(img,
                      iterations=iterations,
                      step=step,
                      max_loss=max_loss)

save_img(result_prefix + '.png', deprocess_image(np.copy(img)))
```

修改后图片为：

如果用 inception_v3 的默认参数进行预测的话，会把它预测为波斯猫

keras 的 example 文件 imdb_bidirectional_lstm.py 解析

imdb 是一个文本情感分析的数据集，通过评论来分析观众对电影是好评还是差评

其网络结构比较简单

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
bidirectional_1 (Bidirectional) (None, 128)            98816
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,658,945
Trainable params: 2,658,945
Non-trainable params: 0
_________________________________________________________________
```

对 imdb 数据集稍微分析一下，

通过函数 load_data 获取到的 x_train, y_train，是一堆编号，这个编号不太直接，可以通过下面代码解析出来：

```
word_index = imdb.get_word_index()

word_index = {k:(v+3) for k,v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2    # unknown
word_index["<UNUSED>"] = 3

reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

```
def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])

for i in range(10):
    print(decode_review(x_train[i]))
    print(y_train[i])
```
就可以看到评论的具体内容，而 y_train 打印出来的是 0 和 1，分别代表差评和好评

x_train 和 y_train 的 shape 分别为

```
(25000, 100)
(25000,)
```

——

不另开帖子了，把其他几个网络的结构也贴出来备忘：

imdb_cnn_lstm.py 的神经网络结构如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 128)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 96, 64)            41024
_________________________________________________________________
max_pooling1d_1 (MaxPooling1D) (None, 24, 64)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 70)                37800
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 71
_________________________________________________________________
```

activation_1 (Activation) (None, 1) 0
==
=
Total params: 2,638,895
Trainable params: 2,638,895
Non-trainable params: 0
____
_
imdb_cnn.py 的神经网络结构如下：

____
____
Layer (type) Output Shape
Param #
==
=====================
embedding_1 (Embedding) (None, 400, 50)
250000
____
____
dropout_1 (Dropout) (None, 400, 50)
0
____
____
conv1d_1 (Conv1D) (None, 398, 250)
37750
____
____
global_max_pooling1d_1 (GlobalMaxPooling1D) (None, 250)
0
____
____
dense_1 (Dense) (None, 250)
62750
____
____
dropout_2 (Dropout) (None, 250)
0
____
____
activation_1 (Activation) (None, 250)
0
____
____

dense_2 (Dense) (None, 1)
251

____

activation_2 (Activation) (None, 1)
0

==

Total params: 350,751

Trainable params: 350,751

Non-trainable params: 0

____

imdb_lstm.py 的神经网络结构为：

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 128)	2560000
lstm_1 (LSTM)	(None, 128)	131584
dense_1 (Dense)	(None, 1)	129

Total params: 2,691,713

Trainable params: 2,691,713

Non-trainable params: 0

____

keras 的 example 文件 imdb_bidirectional_lstm.py 解析

imdb 是一个文本情感分析的数据集，通过评论来分析观众对电影是好评还是差评

其网络结构比较简单

```
_________________________________________________________________
_
Layer (type)                 Output Shape              Param #
=================================================================
=
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
_
bidirectional_1 (Bidirectional) (None, 128)            98816
_________________________________________________________________
_
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
_
dense_1 (Dense)              (None, 1)                 129
=================================================================
=
Total params: 2,658,945
Trainable params: 2,658,945
Non-trainable params: 0
_________________________________________________________________
_
```

对 imdb 数据集稍微分析一下，

通过函数 load_data 获取到的 x_train, y_train，是一堆编号，这个编号不太直接，可以通过下面代码解析出来：

```
word_index = imdb.get_word_index()

word_index = {k:(v+3) for k,v in word_index.items()}
word_index["<PAD>"] = 0
word_index["<START>"] = 1
word_index["<UNK>"] = 2    # unknown
word_index["<UNUSED>"] = 3

reverse_word_index = dict([(value, key) for (key, value) in word_index.items()])
```

```
def decode_review(text):
    return ' '.join([reverse_word_index.get(i, '?') for i in text])

for i in range(10):
    print(decode_review(x_train[i]))
    print(y_train[i])
```
就可以看到评论的具体内容，而 y_train 打印出来的是 0 和 1，分别代表差评和好评

x_train 和 y_train 的 shape 分别为

(25000, 100)
(25000,)

——

不另开帖子了，把其他几个网络的结构也贴出来备忘：

imdb_cnn_lstm.py 的神经网络结构

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          2560000
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 128)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 96, 64)            41024
_________________________________________________________________
max_pooling1d_1 (MaxPooling1D) (None, 24, 64)          0
_________________________________________________________________
lstm_1 (LSTM)                (None, 70)                37800
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 71
```

```
________________________________________________________________________________
_
activation_1 (Activation)          (None, 1)                     0
================================================================================
=
Total params: 2,638,895
Trainable params: 2,638,895
Non-trainable params: 0
________________________________________________________________________________
_
```

imdb_cnn.py 的神经网络结构

```
________________________________________________________________________________
_____________________
Layer    (type)                                                  Output    Shape
Param #
================================================================================
=====================
embedding_1    (Embedding)                                 (None,    400,    50)
250000
________________________________________________________________________________
_____________________
dropout_1    (Dropout)                                     (None,    400,    50)
0
________________________________________________________________________________
_____________________
conv1d_1    (Conv1D)                                      (None,    398,    250)
37750
________________________________________________________________________________
_____________________
global_max_pooling1d_1          (GlobalMaxPooling1D)           (None,         250)
0
________________________________________________________________________________
_____________________
dense_1    (Dense)                                                (None,    250)
62750
________________________________________________________________________________
_____________________
dropout_2    (Dropout)                                            (None,    250)
0
________________________________________________________________________________
_____________________
```

activation_1 (Activation) (None, 250)
0

____

____

dense_2 (Dense) (None, 1)
251

____

____

activation_2 (Activation) (None, 1)
0

==
====================

Total params: 350,751

Trainable params: 350,751

Non-trainable params: 0

____

____

imdb_lstm.py 的神经网络结构

____

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, None, 128)	2560000
lstm_1 (LSTM)	(None, 128)	131584
dense_1 (Dense)	(None, 1)	129

===

Total params: 2,691,713

Trainable params: 2,691,713

Non-trainable params: 0

____

———

keras 的 example 文件 imdb_fasttext.py 解析

该文件功能上也是文本情感分类

默认的代码中 ngram_range = 1，这就差不多是常规的 NLP 处理，编号跟一个 Embedding，这就比较简单

所以我们还是分析一下 ngram_range > 1 的情况，我们先设置 ngram_range = 2，这样的话，x_train 中的第一个句子，首先会进行如下变换，原句：

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 19193, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 10311, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 12118, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32]

变换后：

{(16, 6), (18, 4), (4, 173), (39, 4), (469, 4), (4613, 469), (4536, 1111), (130, 12), (28, 224), (26, 400), (77, 52), (112, 167), (76, 15), (215, 28), (19, 14), (226, 65), (65, 458), (38, 76), (12, 215), (13, 1247), (317, 46), (381, 15), (16, 4472), (36, 256), (407, 16), (25, 100), (32, 15), (194, 7486), (5, 723), (88, 4), (141, 6), (385, 39), (15, 297), (26, 141), (2025, 19), (16, 38), (103, 32), (5952, 15), (447, 4), (22, 12), (670, 2), (33, 4), (5, 14), (82, 10311), (16, 283), (18, 51), (13, 104), (28, 77), (2071, 56), (15, 16), (12, 8), (973, 1622), (15, 13), (1247, 4), (400, 317), (46, 7), (3766, 5), (17, 546), (515, 17), (124, 51), (14, 22), (1385, 65), (1334, 88), (297, 98), (22, 16), (56, 26), (1622, 1385), (480, 5), (25, 104), (10311, 8), (112, 50), (71, 87), (224, 92), (178, 32), (106, 5), (147, 2025), (5, 4), (104, 88), (19, 178), (38, 619), (144, 30), (25, 1415), (150, 4), (4, 12118), (626, 18), (87, 12), (107, 117), (43, 530), (476, 26), (104, 4), (2, 7), (3785, 33), (117, 5952), (5, 144), (65, 16), (192, 50), (7, 4), (98, 32), (226, 22), (17, 12), (4, 226), (66, 3941), (16, 82), (25, 124), (32, 2071), (5, 62), (546, 38), (35, 480), (173, 36), (1029, 13), (16, 5345), (4, 22), (386, 12), (530, 476), (284, 5), (3941, 4), (36, 135), (5535, 18), (5244, 16), (1, 14), (12118, 1029), (172, 4536), (316, 8), (530, 38), (33, 6), (8, 316), (5, 150), (256, 4), (7, 3766), (7486, 18), (88, 12), (167, 2), (4472, 113), (22, 17), (4, 2223), (22, 4), (6, 194), (1920, 4613), (172, 112), (48, 25), (4, 2), (838, 112), (43, 838), (4, 192), (6, 147), (16, 626), (5, 16), (1111, 17), (19193, 5), (723, 36), (4, 172), (22, 71), (16, 480), (135, 48), (38, 1334), (8, 4), (4, 381), (2, 336), (530, 973), (18, 19193), (619, 5), (4468, 66), (62, 386), (51, 36), (8, 106), (17, 515), (30, 5535), (458, 4468), (38, 13), (12, 16), (21, 134), (13, 447), (480, 66), (4, 1920), (16, 43), (2, 9), (66, 3785), (1415, 33), (15, 256), (336, 385), (5, 25), (5345, 19), (6, 22), (283, 5), (71, 43), (50, 670), (14, 407), (92, 25), (22, 21), (113, 103), (134, 476), (50, 16), (256, 5), (4, 130), (100, 43), (36, 71), (36, 28), (26, 480), (9, 35), (2223, 5244), (52, 5), (4, 107), (480, 284)}

这个，直接看看不懂是吧，上面注释中有一个简单的示例，就是说当 ngram_range = 2 时，如果输入是

[1, 4, 9, 4, 1, 4]

那么输出就是

{(4, 9), (4, 1), (1, 4), (9, 4)}

就是按照顺序两两组合，并去掉重复项

如果 ngram_value=3 时，如果输入是

[1, 4, 9, 4, 1, 4]

那么输出就是

[(1, 4, 9), (4, 9, 4), (9, 4, 1), (4, 1, 4)]

而外面有一个 for 循环，所以，当 ngram_value=3 时，上面那个句子就会变为：

{(1, 14, 22), (16, 6), (4472, 113, 103), (66, 3941, 4), (16, 82, 10311), (18, 4), (4, 173), (4613, 469), (130, 12), (26, 141, 6), (26, 400), (316, 8, 106), (12, 16, 43), (38, 1334, 88), (226, 65), (147, 2025, 19), (16, 5345, 19), (135, 48, 25), (65, 458), (104, 4, 226), (381, 15), (16, 4472), (4, 1920, 4613), (36, 256), (18, 4, 226), (4468, 66, 3941), (14, 22, 4), (71, 87, 12), (194, 7486), (5, 723), (88, 4), (141, 6), (385, 39), (144, 30, 5535), (2025, 19), (5244, 16, 480), (16, 38), (2071, 56, 26), (5952, 15), (22, 12), (33, 4), (18, 51), (43, 530, 38), (12118, 1029, 13), (28, 77), (15, 16), (12, 8), (973, 1622), (15, 13), (16, 6, 147), (476, 26, 400), (1247, 4), (3766, 5), (172, 4536, 1111), (17, 546), (4, 107, 117), (56, 26, 141), (48, 25, 1415), (1385, 65, 458), (22, 71, 87), (13, 1247, 4), (28, 224, 92), (1334, 88), (22, 16), (56, 26), (71, 43, 530), (317, 46, 7), (5, 16, 4472), (25, 104), (10311, 8), (224, 92), (16, 626, 18), (106, 5), (38, 619), (144, 30), (17, 515, 17), (25, 1415), (50, 16, 6), (150, 4), (6, 22, 12), (626, 18), (87, 12), (43, 530), (2, 7), (4, 2, 7), (117, 5952), (5, 144), (150, 4, 172), (65, 16), (6, 147, 2025), (385, 39, 4), (5, 14, 407), (5, 144, 30), (1334, 88, 12), (7, 4), (16, 38, 1334), (297, 98, 32), (226, 22), (16, 283, 5), (2, 7, 3766), (17, 12), (66, 3941), (32, 15, 16), (25, 124), (32, 2071), (5, 62), (35, 480), (400, 317, 46), (4, 192, 50), (19, 14, 22), (8, 4, 107), (39, 4, 172), (107, 117, 5952), (141, 6, 194), (284, 5), (36, 135), (52, 5, 14), (21, 134, 476), (5244, 16), (9, 35, 480), (1, 14), (50, 670, 2), (12118, 1029), (172, 4536), (10311, 8, 4), (530, 38), (226, 65, 16), (8, 316), (5, 150), (88, 12), (7, 3766), (18, 19193, 5), (8, 316, 8), (167, 2), (4, 12118, 1029), (113, 103, 32), (6, 194, 7486), (112, 50, 670), (22, 17), (4, 2223), (51, 36, 28), (22, 4), (6, 194), (256, 5, 25), (172, 112), (480, 284, 5), (838, 112), (16, 626), (4, 192), (530, 973, 1622), (256, 4, 2), (2, 336, 385), (1111, 17), (19193, 5), (82, 10311, 8), (16, 480), (22, 71), (135, 48), (5535, 18, 51), (38, 1334), (336, 385, 39), (16, 38, 619), (12, 16, 38), (4, 381), (2, 336), (480, 66, 3785), (117, 5952, 15), (619, 5), (62, 386), (51, 36), (12, 8, 316), (36, 135, 48), (4536, 1111, 17), (30, 5535), (458, 4468), (723, 36, 71), (21, 134), (98, 32, 2071), (480, 66), (838, 112, 50), (626, 18, 19193), (530, 476, 26), (16, 43), (13, 104, 88), (66, 3785), (1415, 33), (92, 25, 104), (65, 16, 38), (15, 256), (65, 458, 4468), (28, 77, 52), (619, 5, 25), (283, 5), (71, 43), (2223, 5244, 16), (50, 670), (14, 407), (38, 13, 447), (167, 2, 336), (22, 21, 134), (7, 3766, 5), (4, 22, 71), (50, 16), (1111, 17, 546), (476, 26, 480), (4, 130), (3941, 4, 173), (22, 4, 1920), (25, 124, 51), (88, 12, 16), (36, 71), (26, 480), (9, 35), (5952, 15, 256), (4, 107), (4613, 469, 4), (973, 1622, 1385), (4, 172, 4536), (4, 226, 65), (39, 4), (12, 16, 626), (469, 4), (16, 480, 66), (4536, 1111), (28, 224), (173, 36, 256), (77, 52), (36, 256, 5), (112, 167), (76, 15), (16, 4472, 113), (215, 28), (19, 14), (43, 530, 973), (172, 112, 167), (458, 4468, 66), (317, 46), (38, 76), (12, 215), (13, 1247), (407, 16), (407, 16, 82), (25, 100), (32, 15), (194, 7486, 18), (4, 22, 17), (36, 28, 224), (88, 4, 381), (15, 297), (4, 226, 22), (130, 12, 16), (26, 141), (103, 32), (447, 4), (670, 2), (77, 52, 5), (5, 4, 2223), (5, 14), (82, 10311), (16, 283), (13, 104), (2071, 56), (2, 9, 35), (5345, 19, 178), (400, 317), (46, 7), (4, 172, 112), (66, 3785, 33), (33, 6, 22), (515, 17), (124, 51), (14, 22), (5, 150, 4), (19193, 5, 62), (19, 178, 32), (1385, 65), (297, 98), (546, 38, 13), (4, 2223, 5244), (32, 2071, 56), (1622, 1385), (12, 16, 283), (480, 5), (104, 88, 4), (2025, 19, 14), (112, 50), (71, 87), (13, 447, 4), (22, 12, 215), (178, 32), (5, 4), (147, 2025), (104, 88), (19, 178), (15, 13, 1247), (226, 22, 21), (22, 17, 515), (447, 4, 192), (4, 381, 15), (4, 12118), (107, 117), (476, 26), (25, 100, 43), (104, 4), (22, 16, 43), (3785, 33), (8, 106, 5), (192, 50), (98, 32), (4, 226), (26, 400, 317), (134, 476, 26), (18, 51, 36), (16, 82), (530, 38, 76), (15, 256, 4), (546, 38), (173, 36), (670, 2, 9), (1029, 13), (16, 5345), (4, 22), (386, 12), (530, 476), (386, 12, 8), (3941, 4), (5535, 18), (1247, 4, 22), (17, 12, 16), (469, 4, 22), (316, 8), (33, 6), (256, 4), (51, 36, 135), (7486, 18), (76, 15, 13), (25, 104, 4), (4472, 113), (224, 92, 25), (1920, 4613), (17, 546, 38), (38, 619, 5), (1622, 1385, 65), (48, 25), (4, 2), (62, 386, 12), (43, 838), (283, 5, 16), (6, 147), (15, 297, 98), (5, 16), (100, 43, 838), (723, 36), (215, 28, 77), (36, 71, 43), (4, 172), (5, 723, 36), (3785, 33, 4), (1029, 13, 104), (8, 4), (192, 50, 16), (530, 973), (26, 480, 5), (43, 838, 112), (18, 19193), (112, 167, 2), (515, 17, 12), (4468, 66), (103, 32, 15), (8, 106), (3766, 5, 723), (4, 130, 12), (17, 515), (38, 13), (12, 16), (13, 447), (43, 530, 476), (381, 15, 297), (4, 1920), (7486, 18, 4), (46, 7, 4), (7, 4, 12118), (2, 9), (480, 5, 144), (336, 385), (5, 25), (1920, 4613, 469), (5345, 19), (15, 16, 5345), (6, 22), (14, 22, 16), (38, 76, 15), (92, 25), (22, 21), (5, 25, 100), (4, 173, 36), (113, 103), (134, 476), (1415, 33, 6), (25, 1415, 33), (87, 12, 16), (256, 5), (284, 5, 150), (35, 480, 284), (30, 5535, 18), (124, 51, 36), (106, 5, 4), (100, 43), (14, 407, 16), (16, 43, 530), (36, 28), (5, 62, 386), (2223, 5244), (33, 4, 130), (52, 5), (5, 25, 124), (12, 215, 28), (480, 284)}

然后对这一堆拆分合并出来的东西进行编码，如

(15833, 395):20001
(217, 17, 10655):20002
(999, 55, 76):20003
(9805, 1031, 17419):20004
(424, 383, 139):20005
(6213, 139):20006
(190, 4, 1631):20007
(4, 1300, 20):20008

(181, 8, 1271):20009
(1818, 11, 2642):20010
(26, 11, 25):20011
(2159, 80, 376):20012
(171, 5392, 306):20013
(6, 1703, 56):20014
(25, 701):20015
(18, 85, 2851):20016
(3048, 23, 111):20017
(93, 35, 4843):20018
(569, 56):20019
(2876, 60):20020
(42, 110, 17):20021

然后对 x_train 重新进行编码：对上面那一行句子，变换为如下形式：

[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36, 256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 4536, 1111, 17, 546, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2025, 19, 14, 22, 4, 1920, 4613, 469, 4, 22, 71, 87, 12, 16, 43, 530, 38, 76, 15, 13, 1247, 4, 22, 17, 515, 17, 12, 16, 626, 18, 19193, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2223, 5244, 16, 480, 66, 3785, 33, 4, 130, 12, 16, 38, 619, 5, 25, 124, 51, 36, 135, 48, 25, 1415, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 10311, 8, 4, 107, 117, 5952, 15, 256, 4, 2, 7, 3766, 5, 723, 36, 71, 43, 530, 476, 26, 400, 317, 46, 7, 4, 12118, 1029, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2071, 56, 26, 141, 6, 194, 7486, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 5535, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 1334, 88, 12, 16, 283, 5, 16, 4472, 113, 103, 32, 15, 16, 5345, 19, 178, 32, 2072507, 1266161, 2923546, 1154584, 1857063, 1616068, 1714474, 4329328, 3275896, 444409, 4244999, 357021, 2204386, 4017303, 2436358, 2414342, 4564906, 3770144, 3500479, 872174, 1863798, 943035, 4384822, 3298316, 195462, 1705545, 3445315, 2679380, 3897218, 1369975, 1444853, 2587497, 1223247, 1834019, 1337290, 4361931, 3051937, 1919727, 4240690, 3591932, 3247084, 1834019, 2667860, 1494110, 1668106, 4598984, 3336776, 1875202, 805524, 414096, 3487212, 376682, 1203953, 3715275, 2906786, 663022, 405105, 462105, 1266161, 366345, 535987, 1350662, 1664509, 385648, 774138, 3219183, 4457427, 1608894, 175325, 1154584, 1857063, 1069587, 2481469, 3957733, 3541066, 1912467, 3279661, 774138, 3526297, 4601076, 633982, 4334780, 175325, 1945899, 805117, 2640072, 3052531, 4551342, 4317108, 3008452, 325845, 4599613, 554077, 2657378, 1074210, 1845337, 2295163, 585777, 1856528, 2955556, 3303299, 2983986, 3078909, 881581, 1234294, 1293338, 175325, 3110261, 4404168, 3227723, 3500479, 1324117, 1270770, 621300, 1546451, 3489999, 2370600, 2453465, 2706265, 4340821, 2465168, 215741, 781388, 2768420, 787354, 4580535, 3649558, 793926, 4284058, 2808043, 2214076, 37409, 1631135, 4313922, 2423293, 4546416, 1925874, 3957466, 1566896, 3167924, 3482887, 4649257, 1533431, 2710579, 1273401, 1371645, 339091, 2955020, 1857063, 1048333, 371041, 4632662, 4462444, 4184160, 3683844, 1130189, 4103711, 634353, 3859054, 4111666, 4652943, 280577, 3342187, 2548984, 512978, 2450927, 623789, 4333054, 422568, 2075762, 2843163, 4361077, 1399187, 2888161, 1212796, 1645243, 4086532, 2231317, 1269240, 3947951, 1087730, 371041, 270624, 1653006, 1535085, 4120352, 3588582, 4601017, 2544481, 621300, 1837573, 4555554, 2941959, 1858587, 3278478, 1338615, 4086532, 270907, 893670, 3110261, 4682638, 3198549, 431569, 175325, 1288546, 3081025, 3950486, 2776183, 288850, 3764569, 1166022, 562043, 2037223, 1079073, 394421, 1224674, 2967649, 3535840, 447226, 4241137, 4175770, 1241778, 497478, 4309649, 4621166, 955680, 2298045, 3548735, 4076356, 3233816, 951955, 2732147, 3275823, 1028990, 2957268, 1138575, 3921875, 3503889, 1136819, 4662389, 3643878, 2644868, 1738871, 2149326, 3878050, 3198969, 4573590, 844683, 3040121, 3424259, 4502983, 3380639, 3118615, 2634687, 4512611, 4413319, 2735167, 3312975, 789314, 3200807, 4325231, 3584612, 4136033, 845060, 1595397, 788821, 4229838, 4232459, 3930004, 1332077, 4354177, 4538725, 781097, 1121875, 3778939, 1483516, 758071, 2903423, 220146, 3444506, 1326694, 3389795, 2446126, 3737982, 4223093, 2303557, 4175770, 584750, 276981, 3810085, 3854999, 2580922, 3449499, 395999, 1056156, 3775697, 4578057, 1800179, 1537039, 3737664, 3690970, 316277, 1358314, 974070, 2642928, 1471662, 275848, 1221052, 1177149, 1380969, 3529334, 3540380, 3340025, 714322, 4284620, 678093, 822874, 517410, 3403412, 1294402, 4462099, 3216640, 3192564, 3209961, 1912923, 3187483, 519710, 621657, 3411325, 2154118, 247815, 4357275, 2942429, 3613380, 1139125, 3667706, 751415, 1782248, 4411849, 3876765, 4368462, 1276321, 3554934, 767011, 3559939, 2170613, 334046, 4072955, 2837364, 937534, 1290614, 3842601, 2188555, 2878704, 3218101, 3641236, 3562312, 3220767, 2439327, 1987147, 2672246, 3085759, 124840, 1185458, 3827304, 1361435, 2081509, 430292, 2887994, 4631142, 2464758, 3138060, 828823, 352647, 4431844, 2448218, 2374328, 2055547, 3297674, 4432172, 4354034, 1001325, 1245515, 3294590, 1131018, 1588305, 2727175, 972595, 3823373, 844324, 109698, 105837, 3951184, 2410181, 1097190, 1999140, 4499486, 1801256, 1985466, 4617078, 1036023, 452670, 3985467, 3014724, 3740459, 3028703, 4260887, 3758711, 1522965, 4456916, 3459929, 1894454, 3679016, 208032, 127063, 818206, 784645, 4472892, 2195399, 2486473, 754806, 1113581, 767330, 4647531, 2237173, 3330033, 2104257, 2367187, 1055957, 4402226, 2179797, 294426, 288755]

变换理由是：

(1, 14):2072507
(14, 22):1266161
(22, 16):2923546
(16, 43):1154584
(43, 530):1857063
(530, 973):1616068
(973, 1622):1714474
(1622, 1385):4329328
(1385, 65):3275896
(65, 458):444409
(458, 4468):4244999
(4468, 66):357021
(66, 3941):2204386
(3941, 4):4017303
(4, 173):2436358

(71, 43):2955020
(224, 92):2941959
(92, 25):1858587
(25, 104):3278478
(104, 4):1338615
(4, 226):4086532
(226, 65):270907
(65, 16):893670
(16, 38):3110261
(38, 1334):4682638
(1334, 88):3198549
(88, 12):431569
(12, 16):175325
(16, 283):1288546
(283, 5):3081025
(5, 16):3950486
(16, 4472):2776183
(4472, 113):288850
(113, 103):3764569
(103, 32):1166022
(32, 15):562043
(15, 16):2037223
(16, 5345):1079073
(5345, 19):394421
(19, 178):1224674
(178, 32):2967649
(1, 14, 22):3535840
(14, 22, 16):447226
(22, 16, 43):4241137
(16, 43, 530):4175770
(43, 530, 973):1241778
(530, 973, 1622):497478
(973, 1622, 1385):4309649
(1622, 1385, 65):4621166
(1385, 65, 458):955680
(65, 458, 4468):2298045
(458, 4468, 66):3548735
(4468, 66, 3941):4076356
(66, 3941, 4):3233816
(3941, 4, 173):951955
(4, 173, 36):2732147
(173, 36, 256):3275823
(36, 256, 5):1028990
(256, 5, 25):2957268

(5, 25, 100):1138575

然后也是进行 pad_sequences，pad 之后的 shape 也是

x_train shape: (25000, 400)
x_test shape: (25000, 400)

然后送入神经网络进行训练，ngram_range = 3 时，神经网络的结构为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 400, 50)           234148350
_________________________________________________________________
global_average_pooling1d_1 ( (None, 50)                0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 51
=================================================================
Total params: 234,148,401
Trainable params: 234,148,401
Non-trainable params: 0
_________________________________________________________________
```

如果 ngram_range = 1，神经网络结构为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 400, 50)           1000000
_________________________________________________________________
```

global_average_pooling1d_1 (GlobalAveragePooling1D) (None, 50)
0

____

dense_1 (Dense) (None, 1)
51

==

Total params: 1,000,051

Trainable params: 1,000,051

Non-trainable params: 0

____

可以看到参数量大了两百倍，因为原来 max_features = 20000，而加上 ngram 之后，max_features = 4682967，

如果 ngram_range = 1 时如果 GPU 空间还够的话，可能加上 ngram 之后，GPU 空间有可能就不足了；

看起来就是用 ngram 来代替原来简陋的 pad，提高一下识别效果；

keras 的 example 文件 lstm_stateful.py 解析

该程序要通过一个 LSTM 来实现拟合窗口平均数的功能

先看输入输出数据，

print(x_train[:10])

[[[-0.08453234]]

  [[0.02169589]]

  [[0.07949955]]

  [[0.00898136]]

  [[0.0405444]]

  [[-0.0227726]]

  [[0.03033169]]

  [[0.03801032]]

  [[0.04372695]]

  [[0.03803725]]]

print(y_train[:10])

```
[[-0.03537864]
 [-0.03141822]
 [ 0.05059772]
 [ 0.04424045]
 [ 0.02476288]
 [ 0.0088859 ]
 [ 0.00377955]
 [ 0.03417101]
 [ 0.04086864]
 [ 0.0408821 ]]
```

y_train 就是 x_train 两两数的平均值，不过 x_train 的最初的第一个数舍去了，看起来 y_train 的第一个数没什么道理似的，这个不必关心

x_train.shape: (800, 1, 1)
y_train.shape: (800, 1)
x_test.shape: (200, 1, 1)
y_test.shape: (200, 1)

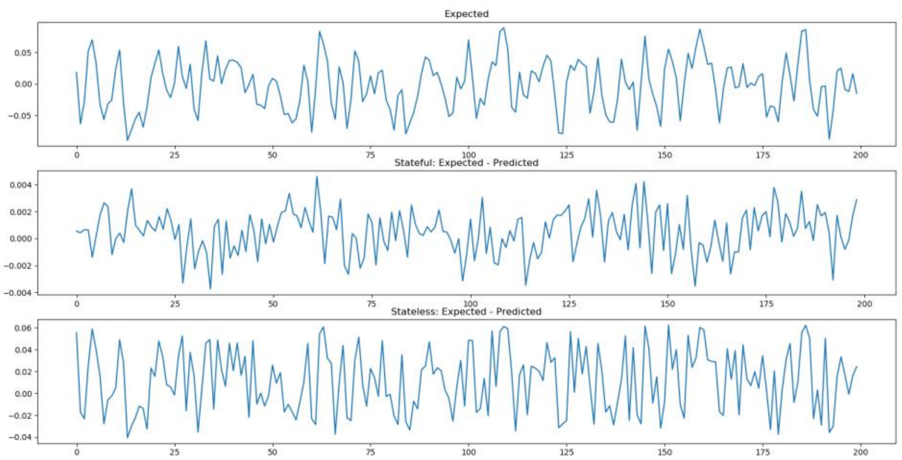
然后是神经网络结构，无论 stateful 是否为 True，结构都是一样的：

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(1, 20)	1760
dense_1 (Dense)	(1, 1)	21

Total params: 1,781
Trainable params: 1,781
Non-trainable params: 0

注意：在 stateful = True 时，我们要在 fit 中手动使得 shuffle = False

预测结果：



从训练时的打印来看：

stateful=True 时，

Epoch 10 / 10

https://blog.csdn.net/zhqh100/article/details/105145986

Train on 800 samples, validate on 200 samples
Epoch 1/1
800/800 [==============================] - 2s 2ms/step - loss: 4.9922e-06 - val_loss: 2.9957e-06

而 stateful=False 时，

Epoch 10/10
800/800 [==============================] - 2s 2ms/step - loss: 8.5024e-04 - val_loss: 9.6397e-04

原理部分介绍可以参考

https://blog.csdn.net/qq_27586341/article/details/88239404

keras 的 example 文件 lstm_text_generation.py 解析

该程序是学习现有的文章，然后学习预测下个字符，这样一个字符一个字符的学会写文章

先打印下 char_indices

{'\n': 0, ' ': 1, '!': 2, '"': 3, "'": 4, '(': 5, ')': 6, ',': 7, '-': 8, '.': 9, '0': 10, '1': 11, '2': 12, '3': 13, '4': 14, '5': 15, '6': 16, '7': 17, '8': 18, '9': 19, ':': 20, ';': 21, '=': 22, '?': 23, '[': 24, ']': 25, '_': 26, 'a': 27, 'b': 28, 'c': 29, 'd': 30, 'e': 31, 'f': 32, 'g': 33, 'h': 34, 'i': 35, 'j': 36, 'k': 37, 'l': 38, 'm': 39, 'n': 40, 'o': 41, 'p': 42, 'q': 43, 'r': 44, 's': 45, 't': 46, 'u': 47, 'v': 48, 'w': 49, 'x': 50, 'y': 51, 'z': 52, 'ä': 53, 'æ': 54, 'é': 55, 'ë': 56}

然后构造训练数据，输入是 sentences，输出是 next_chars，构造成如下结构，sentences 就是把句子拆分出来，next_chars，名字就看出来了，就是下一个字符

sentences	next_chars
preface\n\n\nsupposing that truth is a woma	n
face\n\n\nsupposing that truth is a woman--	w
e\n\n\nsupposing that truth is a woman--wha	t
\nsupposing that truth is a woman--what t	h
pposing that truth is a woman--what then	?
sing that truth is a woman--what then? i	s
g that truth is a woman--what then? is t	h
hat truth is a woman--what then? is ther	e
truth is a woman--what then? is there n	o
uth is a woman--what then? is there not	g
is a woman--what then? is there not gro	u
a woman--what then? is there not ground	\n
woman--what then? is there not ground\nfo	r
an--what then? is there not ground\nfor s	u
-what then? is there not ground\nfor susp	e
at then? is there not ground\nfor suspect	i
then? is there not ground\nfor suspecting	
n? is there not ground\nfor suspecting th	a
is there not ground\nfor suspecting that	a
there not ground\nfor suspecting that all	
re not ground\nfor suspecting that all ph	i
not ground\nfor suspecting that all philo	s
ground\nfor suspecting that all philosop	h

```
  ound\nfor suspecting that all philosopher          s
  d\nfor suspecting that all philosophers,           i
   or suspecting that all philosophers, in           s
   suspecting that all philosophers, in so           f
   pecting that all philosophers, in so far
   ting that all philosophers, in so far as
   g that all philosophers, in so far as th          e
   hat all philosophers, in so far as they           h
     all philosophers, in so far as they hav         e
   l philosophers, in so far as they have b          e
   hilosophers, in so far as they have been          \n
  osophers, in so far as they have been\ndo          g
  phers, in so far as they have been\ndogma          t
  rs, in so far as they have been\ndogmatis          t
   in so far as they have been\ndogmatists,
   so far as they have been\ndogmatists, ha          v
   far as they have been\ndogmatists, have           f
  r as they have been\ndogmatists, have fai          l
  s they have been\ndogmatists, have failed
  hey have been\ndogmatists, have failed to
   have been\ndogmatists, have failed to un          d
  ve been\ndogmatists, have failed to under          s
  been\ndogmatists, have failed to understa          n
  n\ndogmatists, have failed to understand           w
   ogmatists, have failed to understand wom          e
   atists, have failed to understand women-          -
   sts, have failed to understand women--th          a
```

啊，有一点，就是上面的 sentence，直接看起来好像不一样长，实际是一样长的，只不过前面三行，有两个\n，在打印的时候是两个字符，实际上\n 是一个字符，导致的看起来不整齐

然后进行 one-hot 编码，这都是 NLP 的常规操作，然后输入输出数据 shape 为：

x.shape   (200285, 40, 57)
y.shape   (200285, 57)

神经网络模型为

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 128)	95232
dense_1 (Dense)	(None, 57)	7353

```
=================================================================
Total params: 102,585
Trainable params: 102,585
Non-trainable params: 0
_________________________________________________________________
```

keras 的 example 文件 mnist_acgan.py 解析

这是一个 gan 网络，大致分为两个神经网络，一个是生成网络，另一个是判别网络

判别网络的结构大致如下：

```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
input_1 (InputLayer)             (None, 28, 28, 1)     0
____________________________________________________________________________________________________
sequential_1 (Sequential)        (None, 12544)         387840      input_1[0][0]
____________________________________________________________________________________________________
generation (Dense)               (None, 1)             12545       sequential_1[1][0]
____________________________________________________________________________________________________
auxiliary (Dense)                (None, 10)            125450      sequential_1[1][0]
====================================================================================================
Total params: 525,835
Trainable params: 525,835
Non-trainable params: 0
____________________________________________________________________________________________________
```

其中 Sequential1 的网络结构为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 14, 14, 32)        320
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 14, 14, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 14, 14, 64)        0
_________________________________________________________________
```

```
dropout_2 (Dropout)              (None, 14, 14, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)                (None, 7, 7, 128)           73856
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)        (None, 7, 7, 128)           0
_________________________________________________________________
dropout_3 (Dropout)              (None, 7, 7, 128)           0
_________________________________________________________________
conv2d_4 (Conv2D)                (None, 7, 7, 256)           295168
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)        (None, 7, 7, 256)           0
_________________________________________________________________
dropout_4 (Dropout)              (None, 7, 7, 256)           0
_________________________________________________________________
flatten_1 (Flatten)              (None, 12544)               0
=================================================================
Total params: 387,840
Trainable params: 387,840
Non-trainable params: 0
_________________________________________________________________
```

就是跟定一张图片，通过一堆卷积、激活、dropout 之后，最后拉伸生成一个 12544 维度的一个向量，然后跟两个 Dense，一个是判断是否为真图片(generation)，另一个是判断是哪个数字(auxiliary)

生成网络的结构大致如下：

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
input_2 (InputLayer)            (None, 100)          0
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, 1, 100)       1000        input_3[0][0]
__________________________________________________________________________________________________
multiply_1 (Multiply)           (None, 1, 100)       0           input_2[0][0]
                                                                 embedding_1[0][0]
__________________________________________________________________________________________________
```

____
sequential_2 (Sequential) (None, 28, 28, 1) 2656897 multiply_1[0][0]
==
Total params: 2,657,897
Trainable params: 2,657,321
Non-trainable params: 576
____
其中 Sequential1 的网络结构为：

____
Layer (type) Output Shape
Param #
===
dense_1 (Dense) (None, 3456)
349056
____
reshape_1 (Reshape) (None, 3, 3, 384)
0
____
conv2d_transpose_1 (Conv2DTranspose) (None, 7, 7, 192)
1843392
____
batch_normalization_1 (BatchNormalization) (None, 7, 7, 192) 768
____
conv2d_transpose_2 (Conv2DTranspose) (None, 14, 14, 96)
460896
____
batch_normalization_2 (BatchNormalization) (None, 14, 14, 96) 384
____
conv2d_transpose_3 (Conv2DTranspose) (None, 28, 28, 1)
2401
===
Total params: 2,656,897

Trainable params: 2,656,321
Non-trainable params: 576

____

也就是有两个输入，一个是随机数(input_2)，另一个是类别(input_3)，就是数字几

其中输入 input_3 经过一个 Embedding 之后和 和 input_2 相乘，这里是一个点乘，也叫内积，相乘之后 shape 不变，生成一个 100 维的向量，再经过 Dense、Reshape 和 Conv2DTranspose 之后，生成一张 28*28 的黑白图片

上面生成网络和判别网络合并起来，大致结构为：

```
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)            (None, 100)          0
__________________________________________________________________________________________________
input_5 (InputLayer)            (None, 1)            0
__________________________________________________________________________________________________
model_2 (Model)                 (None, 28, 28, 1)    2657897     input_4[0][0]
                                                                 input_5[0][0]
__________________________________________________________________________________________________
model_1 (Model)                 [(None, 1), (None, 10)]    525835      model_2[1][0]
==================================================================================================
Total params: 3,183,732
Trainable params: 2,657,321
Non-trainable params: 526,411
__________________________________________________________________________________________________
```

这里有一个 train_on_batch 加上参数 sample_weight ，这个 sample_weight 是对应 [y, aux_y] ，

```
print(len(disc_sample_weight))
print(len(disc_sample_weight[0]))
print(len(disc_sample_weight[1]))

tmp = [y, aux_y]
print(len(tmp))
print(len(tmp[0]))
print(len(tmp[1]))
```

大致就是这么个意思，y，也就是是否为真实，这个计算损失的结果就正常计算，稍微有一点就是真实图片的 y 的 label 值为 0.95

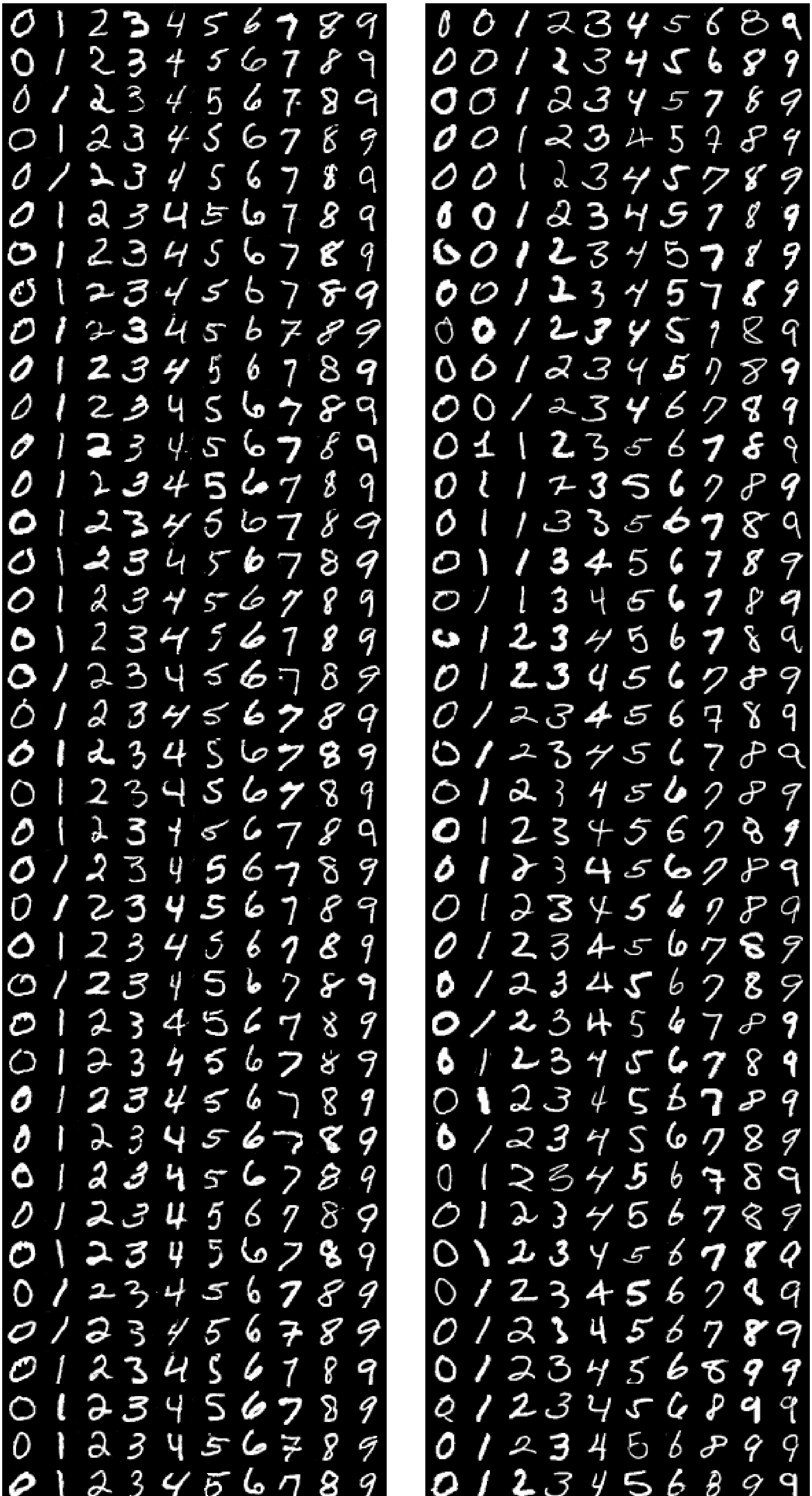
aux_y 的损失，由于对于新生成的图片，计算其分类没有啥意义，所以最初是把它的损失结果直接乘以 0，而对于 mnist 库中的图片，把分类的损失乘以 2，弥补一下

这种情况下，我们训练判别网络 discriminator 一次

然后我们再生成一堆图片，然后把是否为真图片的标签，全部设置为 0.95，然后训练一次 combined 网络，该网络中 discriminator.trainable = False，所以这里仅训练了生成网络

训练过程基本就是这些，其他代码就是计算测试的损失和保存生成图片

如下图，效果不错



keras 的 example 文件 mnist_cnn.py 解析

mnist_cnn.py 基本上就是最简单的一个卷积神经网络了，其结构如下：

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 128)	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0

不再过多解释

另一个更简单的网络结构为 mnist_mlp.py，即 多层感知器（MLP，Multilayer Perceptron）

https://blog.csdn.net/zhqh100/article/details/105145986

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 512)               401920
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 512)               262656
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                5130
=================================================================
Total params: 669,706
Trainable params: 669,706
Non-trainable params: 0
_________________________________________________________________
```

用全连接堆叠起来的图像识别

keras 的 example 文件 mnist_denoising_autoencoder.py 解析

mnist_denoising_autoencoder.py 是一个编解码神经网络，其意义就是如果图片中有噪点的话，可以去除噪点，还原图片

其编码网络为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
flatten_1 (Flatten)          (None, 3136)              0
_________________________________________________________________
latent_vector (Dense)        (None, 16)                50192
=================================================================
Total params: 69,008
Trainable params: 69,008
Non-trainable params: 0
_________________________________________________________________
```

就是输入一张图片，生成一个 16 维的向量

其解码网络为：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
decoder_input (InputLayer)   (None, 16)                0
_________________________________________________________________
```

https://blog.csdn.net/zhqh100/article/details/105145986

```
_________________________________________________________________
dense_1 (Dense)              (None, 3136)              53312
_________________________________________________________________
reshape_1 (Reshape)          (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTranspose) (None, 14, 14, 64)  36928
_________________________________________________________________
conv2d_transpose_2 (Conv2DTranspose) (None, 28, 28, 32)  18464
_________________________________________________________________
conv2d_transpose_3 (Conv2DTranspose) (None, 28, 28, 1)   289
_________________________________________________________________
decoder_output (Activation)  (None, 28, 28, 1)         0
=================================================================
Total params: 108,993
Trainable params: 108,993
Non-trainable params: 0
_________________________________________________________________
```

就是输入一个 16 维的向量，生成一个 28*28 的黑白图片

合并之后的网络结构就是

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_input (InputLayer)   (None, 28, 28, 1)         0
_________________________________________________________________
encoder (Model)              (None, 16)                69008
_________________________________________________________________
decoder (Model)              (None, 28, 28, 1)         108993
=================================================================
Total params: 178,001
Trainable params: 178,001
Non-trainable params: 0
_________________________________________________________________
```

输入就是有噪音的图片，输出是原图，损失函数是 mse，均方差

https://blog.csdn.net/zhqh100/article/details/105145986

效果如下：第一行是原图，第二行是加上噪点之后的图，第三行是解码出来的图

https://blog.csdn.net/zhqh100/article/details/105145986

keras 的 example 文件 mnist_hierarchical_rnn.py 解析

很显然，我没有看懂 HRNN 是啥意思，没有去看论文，应该就是一种 RNN 结构的变形吧

网络结构如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 28, 28, 1)         0
_________________________________________________________________
time_distributed_1 (TimeDistributed) (None, 28, 128)   66560
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 199,434
Trainable params: 199,434
Non-trainable params: 0
_________________________________________________________________
```

输入是图片，输出是分类

类似的，

mnist_irnn.py 的网络结构

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 100)               10200
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
```

https://blog.csdn.net/zhqh100/article/details/105145986

```
_________________________________________________________________
activation_1 (Activation)    (None, 10)                0
=================================================================
Total params: 11,210
Trainable params: 11,210
Non-trainable params: 0
_________________________________________________________________
————————————————————————————————————————
```

keras 的 example 文件 mnist_net2net.py 解析

该程序是介绍，如何把一个浅层的卷积神经网络，加深，加宽

如先建立一个简单的神经网络，结构如下：

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 28, 28, 64)        640
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 14, 14, 64)        36928
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
fc1 (Dense)                  (None, 64)                200768
_________________________________________________________________
fc2 (Dense)                  (None, 10)                650
=================================================================
Total params: 238,986
Trainable params: 238,986
Non-trainable params: 0
_________________________________________________________________
None
```

训练完成后，想办法把他加宽，成下面这样

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 28, 28, 128)       1280
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 14, 14, 128)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 14, 14, 64)        73792
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
```

```
fc1 (Dense)                  (None, 128)               401536
_________________________________________________________________
fc2 (Dense)                  (None, 10)                1290
=================================================================
Total params: 477,898
Trainable params: 477,898
Non-trainable params: 0
_________________________________________________________________
None
```

或者加深，变成下面这样

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1 (Conv2D)               (None, 28, 28, 64)        640
_________________________________________________________________
pool1 (MaxPooling2D)         (None, 14, 14, 64)        0
_________________________________________________________________
conv2 (Conv2D)               (None, 14, 14, 64)        36928
_________________________________________________________________
conv2-deeper (Conv2D)        (None, 14, 14, 64)        36928
_________________________________________________________________
pool2 (MaxPooling2D)         (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
fc1 (Dense)                  (None, 64)                200768
_________________________________________________________________
fc1-deeper (Dense)           (None, 64)                4160
_________________________________________________________________
fc2 (Dense)                  (None, 10)                650
=================================================================
Total params: 280,074
Trainable params: 280,074
Non-trainable params: 0
_________________________________________________________________
None
```

也就是介绍如何对神经网络参数进行增、改、查

首先是获取参数，获取卷积层参数和全连接层代码就是下面两行：

```
w_conv1, b_conv1 = teacher_model.get_layer('conv1').get_weights()
w_fc1, b_fc1 = teacher_model.get_layer('fc1').get_weights()
```

加宽的话，修改卷积层和全连接层参数是下面两行：

```
model.get_layer('conv1').set_weights([new_w_conv1, new_b_conv1])
model.get_layer('fc1').set_weights([new_w_fc1, new_b_fc1])
```

至于改成什么数据，那就自己可以自由发挥了，要么在原来的基础上，拼接随机的一些层，要么把原来的复制一份然后加一些噪音

加深的话，就是新建一个神经网络，把原有的层的参数获取重新拷贝过去就行了，新增加的层的参数，可以自由发挥如何初始化，

修改后的神经网络重新再进行训练