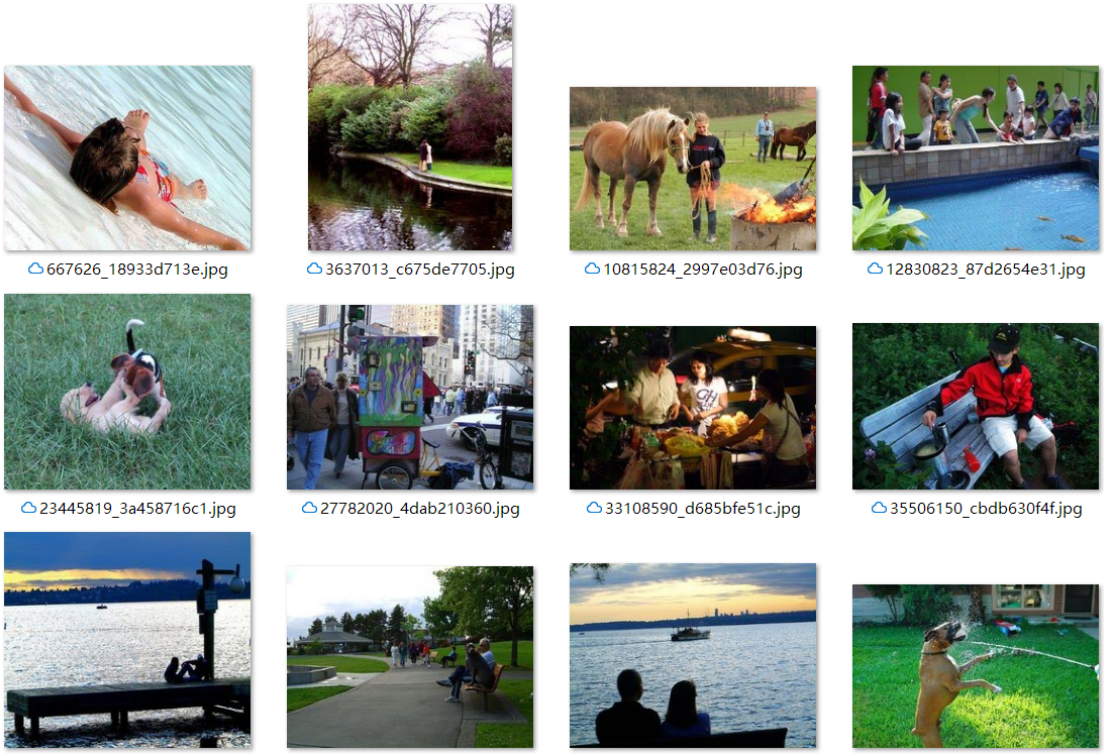
人工智能导论Final - 实验报告

梅佳奕 10165300206

- 数据集：Flickr8k

  共有8k张图片，每张图片带有5句标注。标注作为文本信息以"图片名- 句子"为一行的形式存储在Flickr8k.lemma.token.txt；图片内容不固定，角度不固定，如下：



1. 图片预处理

- 由于输入的图片尺寸不统一，将它们reshape成（224，224，3）大小，目的是能够符合提取特征值部分的网络VGG的输入尺寸；
- 在每张图片的矩阵与文件名之间建立映射关系，用*dictionary* **img_mapping**存储，以便后续用文件名调取图像信息。将**img_mappuing**存到本地。

2. 处理语句信息

- 读取得到 **img_ids**、**annots**、**imgs**三个列表，组织形式为：每项img_id对应一项img图像数据对应5个annotation组成的一个列表，即**annots** 每项是一个长度为5的list。
- shuffle打乱列表顺序，保持一一对应；
- 使用tokenizer将文字转化编码，同时可以得到词频、对应关系、word数等信息，可以选择是否使用onehot，只保留top-k的词用以训练，以提升训练效率；
- improve: 可考虑word embedding

3. 制作数据集

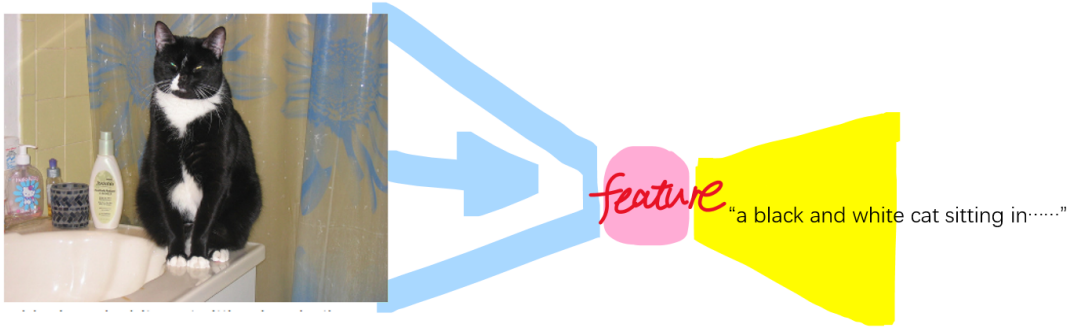
- 按6：2：2划分训练集、验证集、测试集

4. 搭建模型

- 模型结构：

- Encode：从（224，2244，3）大小的图像数据提取一维张量特征
- Decode：从特征张量生成语句（word序列），这个过程分为多步：
  - 特征——>word[0]
  - 特征+word[0]——>word[1]
  - 特征+word[0]+word[1]——>word[2]

    ……



- 网络的构建：
  - Encode：VGG19，由于无分类需求，可以去除含有大量参数的全连接层，（使训练效率更好）并采用预训练的模型，trainable=True
  - Decode：LSTM，对于每张图片，将5句话中的每一个词的生成都处理成一次 input - output，于是一张图片对应的训练所需次数激增，共 **sum(length(sentence[i]))**次。

```
def data_generator(annots_train_r,img_train, max_length):
    while 1:
        for k in range(6068):
            #retrieve photo features
            img = img_train[k]
            annots_5 = annot_train[k]
            input_image, input_sequence, output_word = create_sequences( max_length, annots_5, img)
            yield [[input_image, input_sequence], output_word]


def create_sequences(max_length, annots_5, img):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for annot in annots_5:
        # split one sequence into multiple X,y pairs
        for s in range(1, len(annot)):
            # split into input and output pair
            in_seq, out_seq = annot[:s], annot[s]
            # pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # store
            X1.append(img)
            X2.append(in_seq)
            y.append(out_seq)
```

```
27
28      return np.array(X1), np.array(X2), np.array(y)
```

- 实现：

  尝试了Tensorflow、Keras各种风格的代码与接口，遇到Keras中VGG封装不可add layer、用tf在较底层层面手写网络decoder部分实现易繁杂混乱等问题；最终选择在Keras层面以layer搭建网络，这样encoder与decoder的代码保持了一致性可以对接，同时代码较为简洁、逻辑清楚（对新手复盘友好）。

  需要注意的是，在encoder部分，是嵌入了VGG19的主体部分，可以对VGG19网络中的layer设置trainable参数，这样VGG可以我们的数据集上与decoder部分一起训练，提取的特征更符合decoder的理解。

```
1  def define_model(vocab_size, max_length):
2
3      pretrained_cnn = VGG19(weights='imagenet', include_top=False)
4      pretrained_cnn.layers.pop()
5      # pretrained_cnn.trainable = True
6      for layer in pretrained_cnn.layers[:]:
7          layer.trainable = True
8
9      # feature extractor model
10     inputs1 = Input(shape=(224,224,3))
11 #     Encoder = VGG19(weights='imagenet', include_top=True)
12 #     Encoder.layers.pop()
13     fe0 = pretrained_cnn(inputs1)
14     fe1 = GlobalAveragePooling2D()(fe0)
15     fe2 = Dropout(0.5)(fe1)
16     fe3 = Dense(256, activation='relu')(fe2)
17
18     # sequence model
19     inputs2 = Input(shape=(max_length,))
20     se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
21     se2 = Dropout(0.5)(se1)
22     se3 = LSTM(256)(se2)
23
24     # decoder model
25     decoder1 = add([fe3, se3])
26     decoder2 = Dense(256, activation='relu')(decoder1)
27     outputs = Dense(vocab_size, activation='softmax')(decoder2)
28
29     # tie it together [image, seq] [word]
30     model = Model(inputs=[inputs1, inputs2], outputs=outputs)
31     model.compile(loss='categorical_crossentropy', optimizer='adam')
32
33     # summarize model
34     print(model.summary())
35     plot_model(model, to_file='model.png', show_shapes=True)
36
37     return model
```

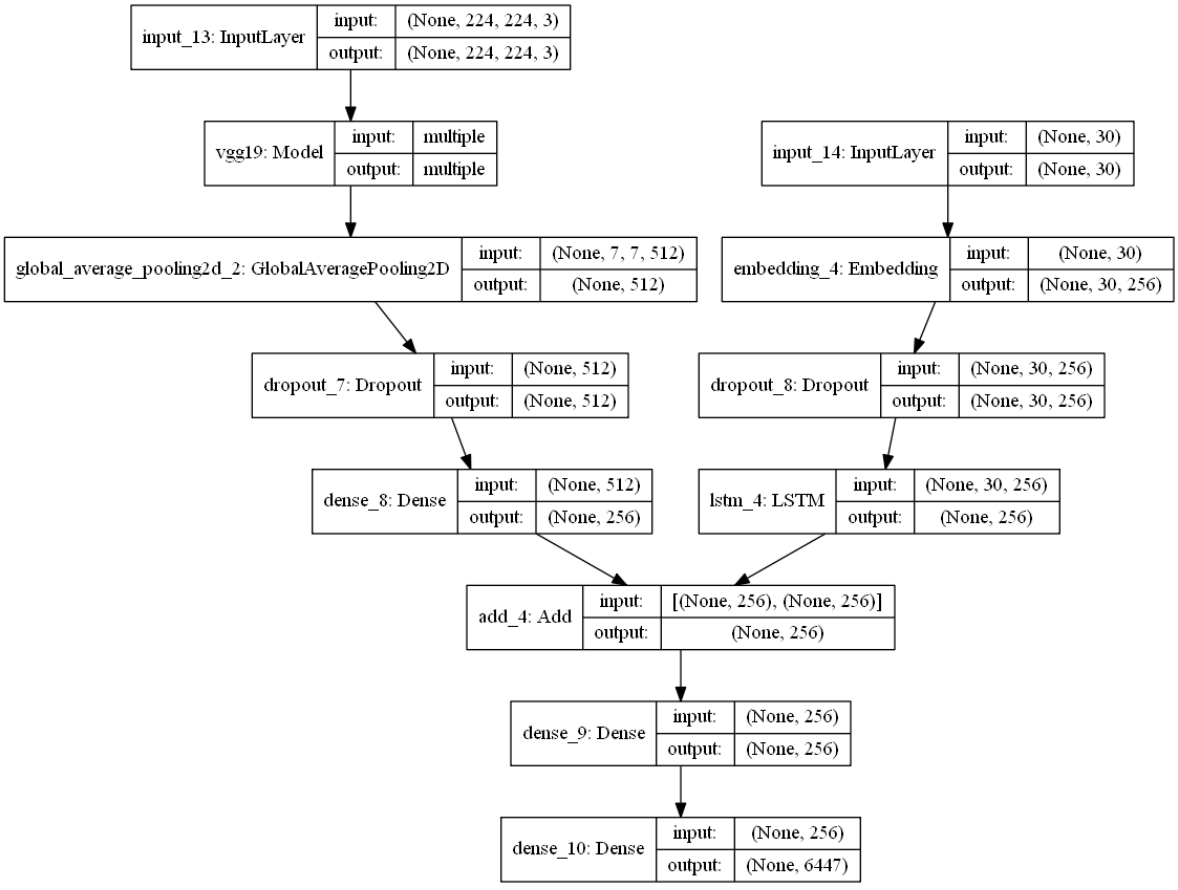
对比：查阅网上其他案例，都是通过一个预训练的提取特征模型来得到特征，作为input，只对decoder部分做训练。我认为这样会给整个"概要生成"带来不可突破的天花板。

模型summury：

Layer (type)	Output Shape	Param #	Connected to
input_13 (InputLayer)	(None, 224, 224, 3)	0	
vgg19 (Model)	multiple	20024384	input_13[0][0]
input_14 (InputLayer)	(None, 30)	0	
global_average_pooling2d_2 (Glo	(None, 512)	0	vgg19[1][0]
embedding_4 (Embedding)	(None, 30, 256)	1650432	input_14[0][0]
dropout_7 (Dropout)	(None, 512)	0	global_average_pooling2d_2[0][0]
dropout_8 (Dropout)	(None, 30, 256)	0	embedding_4[0][0]
dense_8 (Dense)	(None, 256)	131328	dropout_7[0][0]
lstm_4 (LSTM)	(None, 256)	525312	dropout_8[0][0]
add_4 (Add)	(None, 256)	0	dense_8[0][0] lstm_4[0][0]
dense_9 (Dense)	(None, 256)	65792	add_4[0][0]
dense_10 (Dense)	(None, 6447)	1656879	dense_9[0][0]

Total params: 24,054,127
Trainable params: 24,054,127
Non-trainable params: 0

None



5. 训练模型

- 设置

```
epochs = 20
steps = len(annot_train)#一步一图
max_length = 30 #每句长度
```

- fit

```
for i in range(epochs):
    generator = data_generator(annot_train, img_train , max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps,
verbose=1)
```

以generate的方式训练，更能灵活处理多input的情况。

6. Predict

比较遗憾的是，训练所需资源较大，目前在我自己的设备上还没能够完整训练结束。

```
Epoch 1/1
 859/6068 [===>..........................] - ETA: 53:12:33 - loss: 15.5502
```

一个epoch共需训练6068张图片，训练到800张左右时耗时约8小时，在个人电脑上容易中断；

思考：

我认为这与我的网络、数据处理方法都有关，想到的一些改进方法在上文已有提到，其中一些如去除VGG的全连接层以减少训练量已运用。

1. one-hot
2. word embedding
3. 调整模型：
    1. 对图片重点进行捕捉，再生成描述（参考文献《Show, Attend and Tell: Neural Image Caption Generation with Visual Attention》，2015：https://arxiv.org/abs/1411.4555）
    2. LSTM是否用GRU替代
4. 参数的调整（待学习）