**Final Project – Visual Question Answering**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **学号** | **专业班级** | **姓名** | **性别** |
| 1 | 3170102728 | 计科1701 | 王晋开 | 男 |

## Project Introduction

**1.1 VQA任务介绍**

任务要求是输入一张图片和关于图片内容的一个自然语言形式的问题，要求输出正确的自然语言形式的答案。

对于这次的Final Project而言，我们不要求输出结果形式是Free-form的，而仅要求是从诸多候选答案中摘取一个作为结果，因此本质上最后是多分类问题。从我们设计的网络结构也能看出，Softmax+交叉熵的形式适用于解决多分类问题。

**1.2 数据集说明**

本次实验的数据集来源于VQA2.0数据集的一个子集。

Train：32077张图片、44375个问题

Validation：15682张图片、21435个问题

Test：15718张图片、21435个问题

数据集的图片均来自于MSCOCO 2014图片数据集[1]，关于图片内容设计的问题包含yes/no类型、number类型、other类型，每一个问题有十个相关答案，按照人类回答概率从高到低排序。

**1.3 任务难点**

VQA任务本身具有难点。理想中的VQA解决方案模型应当理解输入的问题以及图片，能够同时处理不同种类的问题比如查找型问题、计数问题等等，并且应该生成Free-form的答案也就是自然形成的语言，甚至答案不局限在自然语言域。然后目前的VQA领域的ML模型主要是通过分别提取文字序列和图片的feature然后用某种形式组合去作为第二阶段网络的输入，即隐式特征融合，另外生成的答案往往来自于预先定义的答案集合。总体评价来看，VQA现有阶段的解决方案是一种模式匹配模型，而不是推理模型。

我们在本次Project中参考的主要模型是2016的CVPR论文Stacked attention networks for image question answering[2]。经过我们的前期调研，关于Attention在VQA的应用应该追溯到2014的ECCV论文Where To Look: Focus Regions for Visual Question Answering[3]，该论文的主要工作是计算question feature到图像的每一个区域的feature的attention，然后question feature和每一个区域feature结合起来按照attention值embedding作为一个新的feature，最后输入到FC，这个方法暴力但是确实有attention的思想。2016年的论文提出的SAN模型思想类似，只是用到了多层attention，这是由于对于复杂的问题而言，需要关注的图像上的点有很多，因此一个attention层显然是不够的， 需要使用多个attention层迭代上述过程，每一次迭代都可以形成一个更加精确的视觉注意信息。

我们对2016 base model的优化点主要在两个方向。一是对图像特征的提取：base model采用的是用Imagenet数据集训练完成的VGG16作为提取图像特征的模型，具体而言是去除VGG16最后的三个FC，将最后一个pooling层的输出作为输入图像的feature。我们则参考了2018的CVPR论文Bottom-Up Top-Down attention[4]，采用了训练好的Fast R-CNN作为提取图像特征的模型，考虑的是提取object-box features的方式，提取出的特征更少但是对问题的解答更有效。第二个优化点是对文字序列特征的提取：base model采用的是CNN或者LSTM，我们则参考了经典论文Attention Is All You Need[5]，采用的是多层self-attention的方式，考虑的点在于保持对句子理解的同时提升网络的计算效率。

**1.4 开发环境**

华为ModelArts平台、华为云OBS对象储存服务

Mindspore 1.1.1

Ascend 96GB

Python 3.7.5

## Technical Details

**2.1 理论基础**

**2.1.1 Attention机制**

自2017年谷歌发布Attention Is All You Need一文后，各式各样的Transformer结构改造而成的模型开始在各大深度学习领域屠榜。而Transformer结构中最核心的部分就是attention机制。原始的Transformer是为了处理自然语言处理中的sequence-to-sequence问题而设计的，其中的self-attention操作大致可表述为以下几个步骤：

1. 对每个token序列，首先使用一个嵌入层获得其词嵌入。
2. 将每个词嵌入向量分别与Wq，Wk和Wv矩阵相乘。其中，q代表query，k代表key，v代表value。相乘后，我们会得到每个词的query、key和value向量。
3. 对于每个词，我们都用其query向量与key向量相乘以求内积。这样得到的内积可以被视为一种权重得分。求出内积后，我们再对所有的内积进行softmax操作以使其权重和为1。之后，我们再用以这个权重对句子中所有词的value向量进行加权和，以获得这个词最终的value向量。该向量可以被视为这个词考虑上下文后的综合特征。

因为这样的计算会对上下文中不同的词赋予不同的权重，就像人在阅读文字或者观察景象时会对特别之处赋予更高的注意力，所以这种计算机制被称为attention机制。

**2.1.2 SAN（stacked attention network）**

SAN模型是我们在本次程序设计中使用的基线模型。其大致计算步骤如下：

1. 使用VGG网络对输入图像进行特征提取。SAN模型会舍弃VGG网络末尾与图像分类相关的线性层和softmax层，而直接取出最末尾卷积层的输出作为图像特征。末尾卷积层的输出形状为512×14×14，可以被视为将图像分割成14×14个方块后，对每个方块分别提取512维特征的结果。
2. 使用LSTM或Text-CNN对输入文本进行特征提取。
3. 使用提取到的文本特征对图像特征进行attention，得到综合了图像语义和文本语义的特征向量。
4. 再使用上一步中提取到的特征向量对图像特征进行attention，得到新的特征向量。重复迭代数次，使得特征向量尽可能捕捉到图像中与文本相关的关键特征信息。
5. 使用最后得到的综合特征向量对不同答案进行分数计算，选出分数最高的答案作为判断结果。

SAN模型中的关键在于逐层attention，最终让模型的注意力聚焦到与问题最相关的图像部分。下图展示了当问题是“篮子中有什么”时，模型的注意力逐渐聚焦到图像中篮子和篮子中的狗的过程。

/

**2.1.3 RCNN和Fast-RCNN**

RCNN是一种常用的目标检测模型。在目标检测过程中，我们需要对图像中的物体进行检测和分类，并标明其在图像上的位置。而我们的基线模型SAN也应用了相似的图像分块和基于块的attention，所以我们认为可以使用RCNN来进行分块特征的提取。

RCNN的大致步骤如下：

1. 采用selective search的方法对图像进行区域提取，得到候选区域。
2. 使用卷积神经网络对每个区域的图像特征进行提取。
3. 对提取到的区域图像特征进行分类和边界回归，从而更加精确地勾勒目标物体的边框。

Fast-RCNN的大致做法也与之类似，但在一些具体的做法上会有提升。在本次程序设计中我们采用Fast-RCNN来进行数据预处理，通过提取Fast-RCNN中高层卷积层的输出来获取图像每个区域的特征。随后，类似地我们可以对分块图像特征和文本特征进行逐层attention，得到最终的综合特征信息。

**2.2 实现细节**

**2.2.1 数据预处理**

本次实验数据预处理工作主要为三方面：一是对图像的预处理，需要提取图像特征，并且保存为HDF5文件以便快速读取；二是对文字序列的处理，需要对问题集的每个词作为token生成字典以及对答案集的每个答案作为token生成字典，然后需要将每个问题转换为token idx表示的vector，将问题的答案转换为token idx表示的counter vector；三是对模型训练测试需要的dataset进行生成，需要定义单条训练数据的foramt，整合问题及答案及image id，然后储存为mindrecord格式文件。

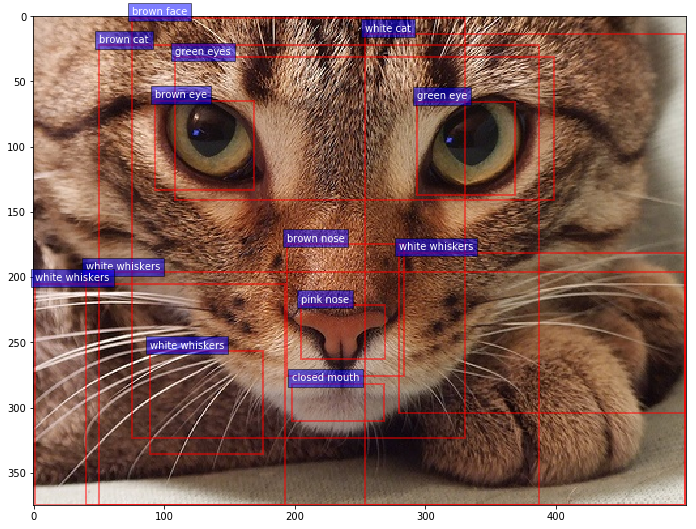
**图像处理**

首先我们下载在imagenet数据集上预训练好的Fast R-CNN模型（<https://storage.googleapis.com/up-down-attention/resnet101_faster_rcnn_final.caffemodel>）

接着我们加载该模型，然后对COCO2014数据集的图片作特征提取，核心的模型输入图片，推理得到features的代码为：

1. net = None
2. weights = 'data/faster\_rcnn\_models/resnet101\_faster\_rcnn\_final.caffemodel'
3. prototxt = 'models/vg/ResNet-101/faster\_rcnn\_end2end\_final/test.prototxt'
5. net = caffe.Net(prototxt, caffe.TEST, weights=weights)
6. conf\_thresh=0.4
7. min\_boxes=10
8. max\_boxes=20
10. im = cv2.imread(im\_file)
11. scores, boxes, attr\_scores, rel\_scores = im\_detect(net, im)
13. # Keep the original boxes, don't worry about the regression bbox outputs
14. rois = net.blobs['rois'].data.copy()
15. # unscale back to raw image space
16. blobs, im\_scales = \_get\_blobs(im, None)
18. cls\_boxes = rois[:, 1:5] / im\_scales[0]
19. cls\_prob = net.blobs['cls\_prob'].data
20. attr\_prob = net.blobs['attr\_prob'].data
21. pool5 = net.blobs['pool5\_flat'].data
23. # Keep only the best detections
24. max\_conf = np.zeros((rois.shape[0]))
25. **for** cls\_ind **in** range(1,cls\_prob.shape[1]):
26. cls\_scores = scores[:, cls\_ind]
27. dets = np.hstack((cls\_boxes, cls\_scores[:, np.newaxis])).astype(np.float32)
28. keep = np.array(nms(dets, cfg.TEST.NMS))
29. max\_conf[keep] = np.where(cls\_scores[keep] > max\_conf[keep], cls\_scores[keep], max\_conf[keep])
31. keep\_boxes = np.where(max\_conf >= conf\_thresh)[0]
32. **if** len(keep\_boxes) < min\_boxes:
33. keep\_boxes = np.argsort(max\_conf)[::-1][:min\_boxes]
34. **elif** len(keep\_boxes) > max\_boxes:
35. keep\_boxes = np.argsort(max\_conf)[::-1][:max\_boxes]

比如对以下这张图片的特征提取，我们自适应得到了13个box feature，但是后续为了方便进一步训练，我们固定box数量为36。提取完所有COCO数据集的图片特征后，我们保存为tsv文件。tsv文件地址（https://storage.googleapis.com/up-down-attention/trainval\_36.zip）



由于给定的数据集并不完全符合COCO2014，所以我们做了进一步缩减，提取所需的图片特征，并且按照给定数据集的划分分别保存到train.h5、val.h5以及test.h5中。核心代码的思路为从questions文件夹的各个数据集json文件中获取所需的image id，然后从上述tsv文件中读取整理为HDF5文件：

1. **def** get\_needed\_imageid(dataset="test"):
2. path = './questions/' + dataset + '.json'
3. **print**(path)
4. imageid\_set = set()
5. with open(path, 'r') as f:
6. t = json.load(f)
7. **print**(len(t['questions']))
8. **for** question **in** t['questions']:
9. imageid\_set.add(question['image\_id'])
10. **return** list(imageid\_set)
12. csv.field\_size\_limit(sys.maxsize)
14. **def** process\_image\_feature(dataset="test"):
15. h5FilePath = config["preprocessed\_%s\_path" % (dataset)]
16. num = config[('%s\_num' % (dataset))] # number of images in train or in val or in test
18. features\_shape = (
19. num,
20. config.output\_features,
21. config.output\_size,
22. )
23. boxes\_shape = (
24. num,
25. 4, # top, bottom, left, right
26. config.output\_size,
27. )
28. with h5py.File(h5FilePath, 'w', libver='latest') as fd:
29. features = fd.create\_dataset('features', shape=features\_shape, dtype='float32')
30. boxes = fd.create\_dataset('boxes', shape=boxes\_shape, dtype='float32')
31. coco\_ids = fd.create\_dataset('ids', shape=(num,), dtype='int32')
32. widths = fd.create\_dataset('widths', shape=(num,), dtype='int32')
33. heights = fd.create\_dataset('heights', shape=(num,), dtype='int32')
35. FIELDNAMES = ['image\_id', 'image\_w','image\_h','num\_boxes', 'boxes', 'features']
36. needed\_imageids = get\_needed\_imageid(dataset)
37. i = 0
39. **for** Tdataset **in** ['train', 'eval', 'test']:
40. tsvFilePath = "%s2014\_resnet101\_faster\_rcnn\_genome\_36.tsv" % (Tdataset)
41. with open(tsvFilePath, "r") as tsvF:
42. reader = csv.DictReader(tsvF, delimiter='\t', fieldnames=FIELDNAMES)
43. origin\_num = config[('origin\_%s\_num' % (Tdataset))]
44. **for** \_, item **in** enumerate(tqdm(reader, total=origin\_num)):
45. cur\_id = int(item['image\_id'])
46. **if** cur\_id **not** **in** needed\_imageids:
47. **continue**
48. coco\_ids[i] = int(item['image\_id'])
49. widths[i] = int(item['image\_w'])
50. heights[i] = int(item['image\_h'])
51. buf = base64.decodestring(item['features'].encode('utf8'))
52. array = np.frombuffer(buf, dtype='float32')
53. array = array.reshape((-1, config.output\_features)).transpose() # 36\*2048 -> T -> 2048\*36
54. features[i, :, :array.shape[1]] = array
55. buf = base64.decodestring(item['boxes'].encode('utf8'))
56. array = np.frombuffer(buf, dtype='float32')
57. array = array.reshape((-1, 4)).transpose() # 36\*4 -> T -> 4\*36
58. boxes[i, :, :array.shape[1]] = array
59. i = i+1

这里需要注意的是每一张图的特征矩阵为36\*2048，即36个box，每个box的feature为1\*2048。box的位置矩阵为36\*4，容易理解，不加赘述。

**文字序列处理**

首先我们对train，val，test集的所有question统一处理生成字典。代码的处理方式为question到sentence到token，然后对所有的token进行统计形成question token dict。注意我们需要顺便统计question的最长token数，以后需要用到。核心代码如下：

1. **def** prepare\_question(q):
2. q = q.lower()[:-1]
3. q = \_special\_chars.sub('', q)
4. **return** q.split(' ')
6. **def** prepare\_questions(questions\_json):
7. # Tokenize and normalize questions
8. questions = [q['question'] **for** q **in** questions\_json['questions']]
9. **for** question **in** questions:
10. **yield** prepare\_question(question)
12. **def** extract\_vocab(iterable, top\_k=None, start=0):
13. # Turns an iterable of list of tokens into a vocabulary.
14. all\_tokens = itertools.chain.from\_iterable(iterable)
15. counter = Counter(all\_tokens)
16. **if** top\_k:
17. most\_common = counter.most\_common(top\_k)
18. most\_common = (t **for** t, c **in** most\_common)
19. **else**:
20. most\_common = counter.keys()
21. # descending in count, then lexicographical order
22. tokens = sorted(most\_common, key=**lambda** x: (counter[x], x), reverse=True)
23. vocab = {t: i **for** i, t **in** enumerate(tokens, start=start)}
24. **return** vocab
26. **def** process\_questions():
27. questions = []
28. **for** dataset **in** ['train', 'val', 'test']:
29. path = './questions/' + dataset + '.json'
30. **print**(path)
31. with open(path, 'r') as f:
32. t = json.load(f)
33. questions.extend(list(prepare\_questions(t)))
35. question\_vocab = extract\_vocab(questions, start=1)
36. data\_max\_length = max(map(len, questions))
37. **return** question\_vocab, data\_max\_length

对answer的处理是相似的，但是需要注意在生成token的时候需要counter一遍根据most common截掉不需要的answer，比如我们规定最大答案数为6000，那么相当于是一个6000分类问题，most common排序大于6000的答案将被舍去不生成为token。上述思路的核心代码如下：

1. **def** process\_punctuation(s):
2. **if** \_punctuation.search(s) **is** None:
3. **return** s
4. s = \_punctuation\_with\_a\_space.sub('', s)
5. **if** re.search(\_comma\_strip, s) **is** **not** None:
6. s = s.replace(',', '')
7. s = \_punctuation.sub(' ', s)
8. s = \_period\_strip.sub('', s)
9. **return** s.strip()
11. **def** prepare\_answer(answer\_list):
12. **return** list(map(process\_punctuation, answer\_list))
14. **def** prepare\_answers(answers\_json):
15. # Normalize answers
16. answers = [[a['answer'] **for** a **in** ans\_dict['answers']] **for** ans\_dict **in** answers\_json['annotations']]
18. **for** answer\_list **in** answers:
19. **yield** prepare\_answer(answer\_list)
21. **def** process\_questions():
22. questions = []
23. **for** dataset **in** ['train', 'val', 'test']:
24. path = './questions/' + dataset + '.json'
25. **print**(path)
26. with open(path, 'r') as f:
27. t = json.load(f)
28. questions.extend(list(prepare\_questions(t)))
30. question\_vocab = extract\_vocab(questions, start=1)
31. data\_max\_length = max(map(len, questions))
32. **return** question\_vocab, data\_max\_length

**Dataset生成**

Dataset生成及mindrecord文件生成主要是为模型训练提供便利，规定Dataset格式同时能够保证数据在基线的一致性。参照base model论文，我们规定格式为：

question\_id,

image\_id: the index used to get image feature from h5,

question: the index of quesiton words,

answer,

answer\_counter,

answer\_label

我们首先读取了question集和对应answer集合，然后分别制作为dict以方便后续的查询。接着我们开始遍历question dict，根据第二步中生成的字典文件将question转换为token idx vector。同时根据question id从answer dict中取出对应的answers，然后处理得到answer idx vector、answer counter以及作为最终答案的answer label。在遍历question dict结束后，我们也就得到了一个符合前文所述dataset格式的列表作为最终的dataset。上述思路的基线代码为：

1. q\_file\_path = "questions/%s.json" % (dataset)
2. a\_file\_path = "annotations/%s.json" % (dataset)
3. q\_dict = dict()
4. a\_dict = dict()
5. # load questions and answers
6. with open(q\_file\_path, 'r') as f:
7. t = json.load(f)
8. t = t['questions']
9. **for** item **in** t:
10. q\_dict[item['question\_id']] = item
12. with open(a\_file\_path, 'r') as f:
13. t = json.load(f)
14. t = t['annotations']
15. **for** item **in** t:
16. a\_dict[item['question\_id']] = item
17. # load vocab
18. vocab = load\_vocab()
19. final\_sources = []
20. **for** qid, item **in** q\_dict.items():
21. source = dict()
22. source['question\_id'] = qid
23. source['image\_id'] = item['image\_id']
25. q\_vec = encode\_question(
26. item['question'],
27. vocab['question'],
28. min(config.max\_q\_length, vocab['q\_max\_len'])
29. )
30. source['question'] = q\_vec
32. # search relevant answer
33. a\_item = a\_dict.get(qid, -1)
34. **if** a\_item == -1:
35. **print**('cannot find qid %s in answer dict' % qid)
36. **continue**
38. a\_vecs, a\_counter, a\_label = encode\_answer(
39. a\_item['answers'],
40. vocab['answer']
41. )
42. source['answer'] = a\_vecs
43. source['answer\_counter'] = dict(a\_counter)
44. source['answer\_label'] = a\_label
46. final\_sources.append(source)

所述对question和answer的编码过程的相关代码为：

1. **def** encode\_question(q, vocab, max\_question\_length):
2. num\_tokens = len(vocab)
3. vec = np.zeros(max\_question\_length, dtype=int)
4. vec.fill(num\_tokens)
5. q = prepare\_question(q)
6. **for** i, token **in** enumerate(q):
7. **if** i >= max\_question\_length:
8. **break**
9. index = vocab.get(token, num\_tokens - 1)
10. vec[i] = index
11. **return** vec
12. **def** encode\_answer(answer\_list, vocab):
13. num\_tokens = len(vocab)
14. answer\_list = [t['answer'] **for** t **in** answer\_list]
15. answers = prepare\_answer(answer\_list)
16. a\_vecs = [vocab.get(t, num\_tokens-1) **for** t **in** answers]
17. a\_counter = Counter(a\_vecs)
18. a\_label = a\_counter.most\_common(1)[0][0]
19. **return** a\_vecs,a\_counter,a\_label

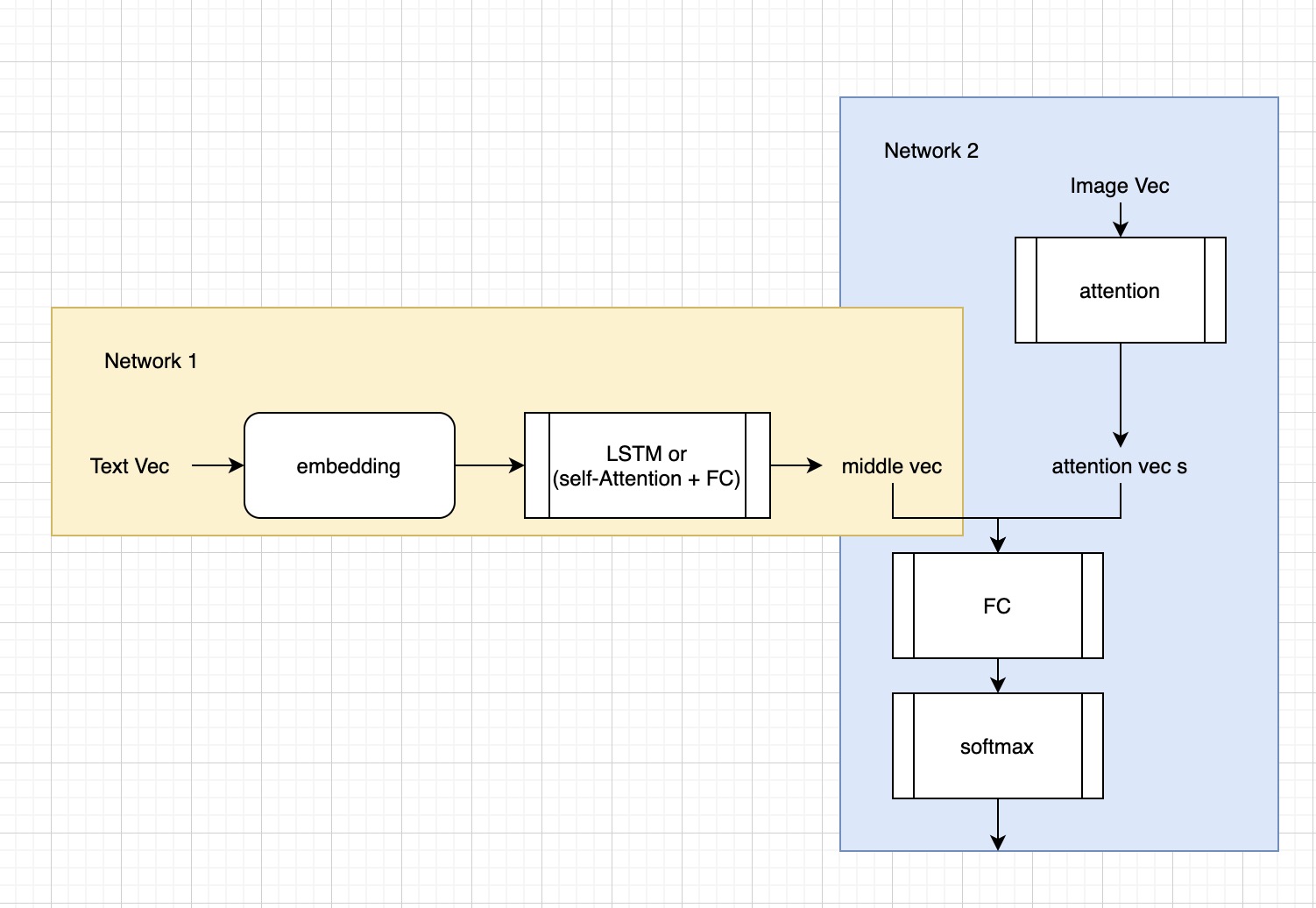
然后我们根据mindspore的文档制作dataset，以及根据需要储存为mindrecord格式的文件。

1. # create mindspore dataset
2. random.shuffle(final\_sources)
3. gen\_dataset = ds.GeneratorDataset(
4. source=Generator(input\_list=final\_sources),
5. column\_names=["data","label"],
6. shuffle=False
7. )
8. gen\_dataset=gen\_dataset.batch(batch\_size=batch\_size,drop\_remainder=True)
9. **if** dataset == 'train':
10. gen\_dataset = gen\_dataset.repeat(config.epoch\_size)
12. # save to a mindrecord file
13. **if** save\_mindrecord:
14. schema\_json = {
15. "question\_id": {"type": "int32"},
16. "image\_id": {"type": "int32"},
17. "question": {"type": "string"},
18. "answer": {"type": "string"},
19. "answer\_counter": {"type": "string"},
20. "answer\_label": {"type": "int32"},
21. }
22. **for** source **in** final\_sources:
23. source['question'] = base64.b64encode(source['question'])
24. source['answer'] = base64.b64encode(source['answer'])
25. source['answer\_counter'] = base64.b64encode(source['answer\_counter'])
27. # index for accelerate data load
28. indexes = ["question\_id", "image\_id", "answer\_label"]
30. **if** **not** os.path.exists('mindrecord'):
31. os.makedirs('mindrecord')
32. mr\_file\_path = 'mindrecord/%s.mindrecord' % (dataset)
33. writer = FileWriter(file\_name=mr\_file\_path, shard\_num=4)
34. writer.add\_schema(schema\_json, "data\_schema")
35. writer.add\_index(indexes)
36. writer.write\_raw\_data(final\_sources)
37. writer.commit()

**2.2.2 模型训练**

（截止到7.9，小组尚未完成模型训练）

根据讨论的想法，算法模型示意图为：



## Experiment Results

## 4. References

[1] Lin T Y, Maire M, Belongie S, et al. Microsoft coco: Common objects in context[C]//European conference on computer vision. Springer, Cham, 2014: 740-755.

[2] Yang, Zichao, et al. "Stacked attention networks for image question answering." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[3] Shih K J, Singh S, Hoiem D. Where to look: Focus regions for visual question answering[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 4613-4621.

[4] Anderson, Peter, et al. "Bottom-up and top-down attention for image captioning and visual question answering." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

[5] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.