nlp-project-vqa

组员

杨浩峰 3190105301

翟智超 3190104555

孟楚天 3190105834

段皞一 3190105359

一. 研究任务描述

1.1 任务：Visual Question Answering

以一张图片和一个关于图片内容的自然语言形式的问题作为输入，要求输出正确的答案。

例如给出如下图片及问题，要求结合图片内容对于问题进行回答：

- 图片

- 问题：“图片中有几个人？”
- 预期回答：”3个“

1.2 数据集

- **数据集**URL：https://drive.google.com/open?id=1_VvBqqxPW_5HQxE6alZ7_-SGwbEt2_zn
- **数据集结构：**
  - $Train$：44375
  - $Validation$：21435
  - $Test$：21435

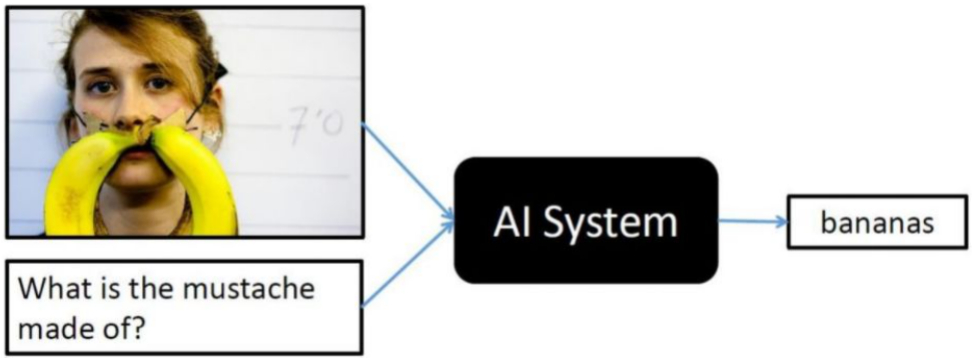
- $Directory$：其中$annotation$文件夹包含对于回答、问题、以及对应图像的注解；$images$文件夹中为测试、验证、训练集的图片；$questions$文件夹中为对应各个图片的问题。

```
data
├── annotations
|   ├── test.json
|   ├── train.json
|   └── val.json
├── images
|   ├── test
|   ├── train
|   └── val
├── questions
    ├── test.json
    ├── train.json
    └── val.json
```

二. 研究内容及挑战

2.1 VQA简介

VQA 介于图像理解（CV）和自然语言处理（NLP）的交集。VQA 任务的目的是开发出一种系统来回答有关输入图像的特定问题。答案可以采用以下任何形式：单词，短语，二元答案，多项选择答案或文本填空。

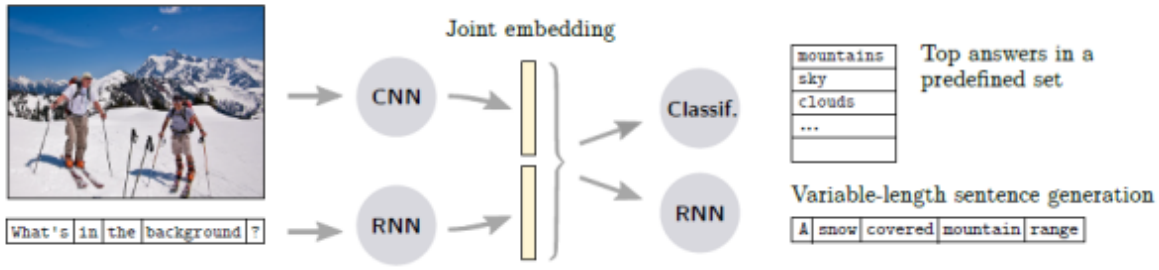


在 CV 领域，CNN 是当前非常重要的基础模型。进而产生了 $VGGNet$，$Inception$，$ResNet$等模型。类似的，NLP 领域，RNN 是之前主要的模型架构，因为$LSTM$的引入使得 RNN 有了重大突破。如$Vanilla\ VQA$ 模型使用了 $VGGNet$ 和 $LSTM$ 相结合的方法。后来在NLP领域的注意力机制也开始在CV领域开始得到应用，就有了 $Stacked\ Attention\ Network$等模型。2018年 $BERT$ 横空出世，在 NLP 领域掀起了革命。所以近两年，$BERT$ 也开始进入到VQA任务中，$BERT$一开始是用于替换 RNN 来处理文本。但是在2019，2020年开始，一些模型（如，$VL-BERT$）开始把简单有效的$Transformer$模型作为主干并进行拓展，视觉和语言嵌入特征可以同时作为输入，然后进行预训练以兼容下游的所有视觉-语言联合任务。

2.2 研究方法概述

2.2.1 $Joint\ embedding\ approaches$

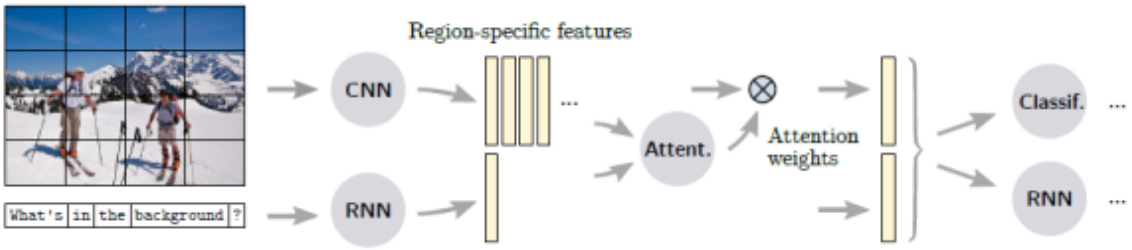
$Joint\ embedding$是处理多模态问题时的经典思路，在这里指对图像和问题进行联合编码。该方法的示意图为：



首先，图像和问题分别由CNN和RNN进行第一次编码得到各自的特征，随后共同输入到另一个编码器中得到$joint\ embedding$，最后通过解码器输出答案。值得注意的是，有的工作把VQA视为序列生成问题，而有的则把VQA简化为一个答案范围可预知的分类问题。在前者的设定下，解码器是一个RNN，输出长度不等的序列；后者的解码器则是一个分类器，从预定义的词汇表中选择答案。

2.2.2 $Attention\ mechanisms$

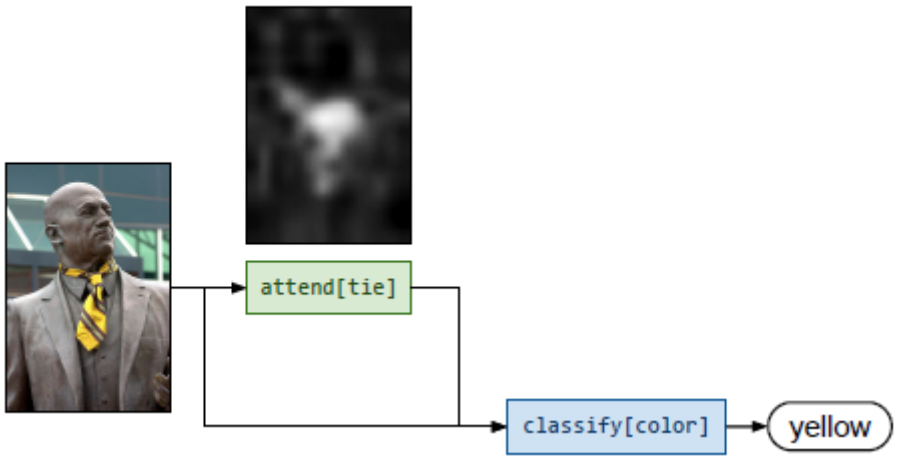
$Attention$机制起源于机器翻译问题，目的是让模型动态地调整对输入项各部分的关注度，从而提升模型的“专注力”。下面就是将$attention$机制应用到VQA问题的模型示意图。



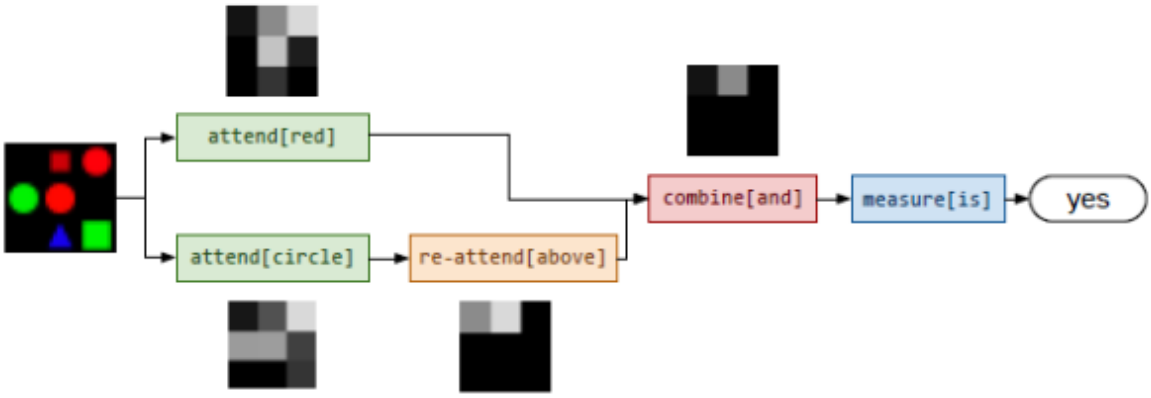
相关的工作表明，加入$attention$机制能获得明显的提升，从直观上也比较容易理解：在$attention$机制的作用下，模型在根据图像和问题进行推断时不得不强制判断“该往哪看”，比起原本盲目地全局搜索，模型能够更有效地捕捉关键图像部位。

2.2.3 $Compositional\ Models$

$CompositionalModels$的核心思想是设计一种模块化的模型。其最大的特点是根据问题的类型动态组装模块来产生答案。



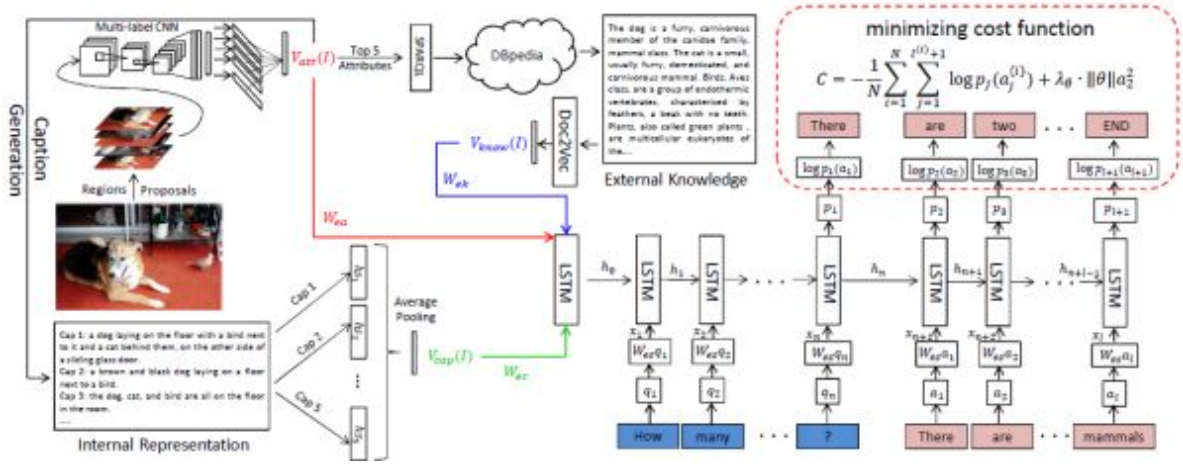
比如，在上面的例子中，当面对 *What color is his tie?* 这个问题时，模型首先利用$parser$对问题进行语法解析，接着判断出需要用到$attend$和$classify$这两个模块，然后判断出这两个模块的连接方式。最终，模型的推理过程是，先把注意力集中到 tie上，然后对其$color$进行分类，得出答案。



而在另一个例子中，当面对 *Is there a red shape above a circle?* 这种更为复杂的问题时，模型选择的模块也自动变得复杂了许多。

2.2.4 $Models\ using\ external\ knowledge\ base$

虽然VQA要解决的是看图回答问题的任务，但实际上，很多问题往往需要具备一定的先验知识才能回答。例如，为了回答"图上有多少只哺乳动物"这样的问题，模型必须得知道"哺乳动物"的定义，而不是单纯理解图像内容。因此，把知识库加入VQA模型中就成了一个很有前景的研究方向。大致模型结构如下：



- 红色部分表示，对图像进行多标签分类，得到图像标签（$attribute$）。
- 蓝色部分表示，把上述图像标签中最明显的5个标签输入知识库$DBpedia$中检索出相关内容，然后利用$Doc2Vec$进行编码。
- 绿色部分表示，利用上述图像标签生成多个图像描述（$caption$），将这一组图像描述编码。
- 以上三项同时输入到一个$Seq2Seq$模型中作为其初始状态，然后该$Seq2Seq$模型将问题进行编码，解码出最终答案，并用MLE的方法进行训练。

2.3 VQA问题难点与挑战

2.3.1 VQA问题与相似问题的对比

- VQA**与**QA：文本QA即纯文本的回答，计算机根据文本形式的材料回答问题。与之相比，VQA把材料换成了图片形式，从而引入了一系列新的问题：
  - 图像是更高维度的数据，比纯文本具有更多的噪声。
  - 文本是结构化的，也具备一定的语法规则，而图像则不然。
  - 文本本身即是对真实世界的高度抽象，而图像的抽象程度较低，可以展现更丰富的信息，同时也更难被计算机"理解"。
- VQA**与**$Image\ Captioning$：与$Image\ Captioning$这种看图说话的任务相比，VQA的难度也显得更大。

- $Image\ Captioning$更像是把图像“翻译”成文本，只需把图像内容映射成文本再加以结构化整理即可，而VQA需要更好地理解图像内容并进行一定的推理，有时甚至还需要借助外部的知识库。
- VQA的评估方法更为简单，因为答案往往是客观并简短的，很容易与$ground\ truth$对比判断是否准确，不像$Image\ Captioning$需要对长句子做评估。

2.3.2 VQA问题实验中的挑战

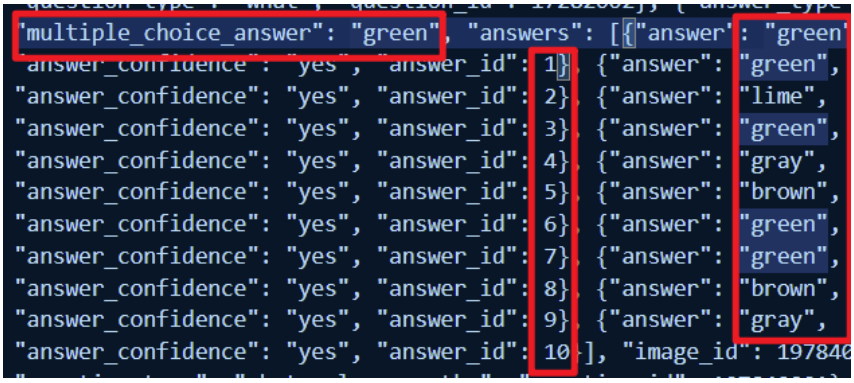
- **多词答案的处理**

  考虑到一个答案包含的词并不唯一，如`fly kite`，因此我们对答案不做任何分词，直接以答案为单元进行编码，在实现上以一个字典存储答案索引。

- **备选项的处理**

  $json$ 文件中提供了问题答案的备选项内容，实验的后期阶段我们尝试运用这个信息，以提高训练的正确率。

  以下是 $json$ 文件中备选项的展示：



  如何对备选项进行处理呢？我们实验中使用了注意力机制。具体来讲，就是把传入的$images$和$questions$ 作为 $query$, $options\_answer$ 作为 key。

$$h = tanh(W_q \cdot q + W_k \cdot k)$$

$$p = softmax(W_h \cdot h)$$

$$output = matMul(p, options)$$

  通过注意力机制，根据输入的问题和图片，关注备选答案中更符合正确答案的哪些答案，通过上述公式计算，得出的 output 继续进入模型中的后续处理。经过多轮训练，能够达到相比之前更好地效果。

- **预训练的特征**

  为了从图中提取到更优的信息，我们使用https://github.com/peteanderson80/bottom-up-attention 为整个$COCO$数据集提供的基于`Faster R-CNN`和`ResNet-101`的预训练特征文件，为每个图片创建36个特征的预训练特征文件，这些特征文件以tsv（制表符分隔值）格式存储。

  在之后我们读取文件并重建特征，这个过程需要我们把图片与图片的$feature\ map$对应起来。这里面会涉及到一个问题：由于单张图片的$feature\ map$是36×2048的类型为`float32`的向量，这使得整个$feature\ map$文件尺寸过大，超出了$ModelArts$的存储能力，这意味着我们不得不对预训练特征文件进行分割，进一步需要生成多个$mindrecord$、读取$mindrecord$生成$dataset$、读取$dataset$进行训练、验证和测试，这使得整个数据预处理过程变得异常复杂。

- **一张图片可能对应多个问题**

  在考虑使用预训练的特征后，正如上面提到的分割处理文件，我们只能通过分段遍历$40 + G$的feature_map.tsv文件来组合各种信息。因此我们需要以图片的id作为`key`来组织数据，随后我们意识到一张图片可能对应多个问题。因此我们字典的key不能够直接使用id，否则会造成大概`25%`数据的丢失(对于训练集中的图片而言)。对于这个问题的解决方案是我们的key最后采用了

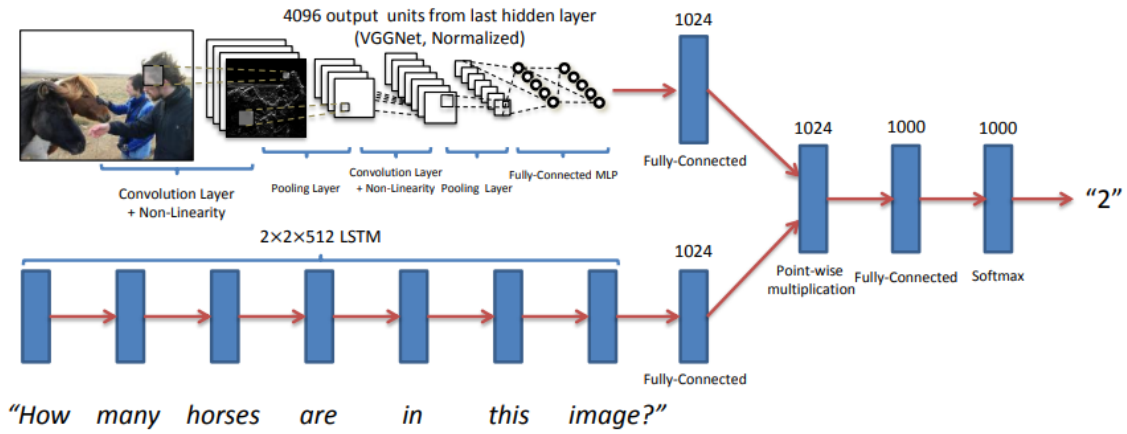
(id,idx)的方式，i在插入数据的时候从0开始查找直到在字典中找不到对应的idx，然后插入，最后$0-idx-1$代表同一张图片对应的一组问题。

三. 算法理论与技术细节

本实验实现并改进了三种模型：$Baseline$、$SAN(Stack\ Attention\ Network)$模型以及$topdown\ attention$模型。

3.1 $Baseline$

$Baseline$中采用的模型示意图如下：

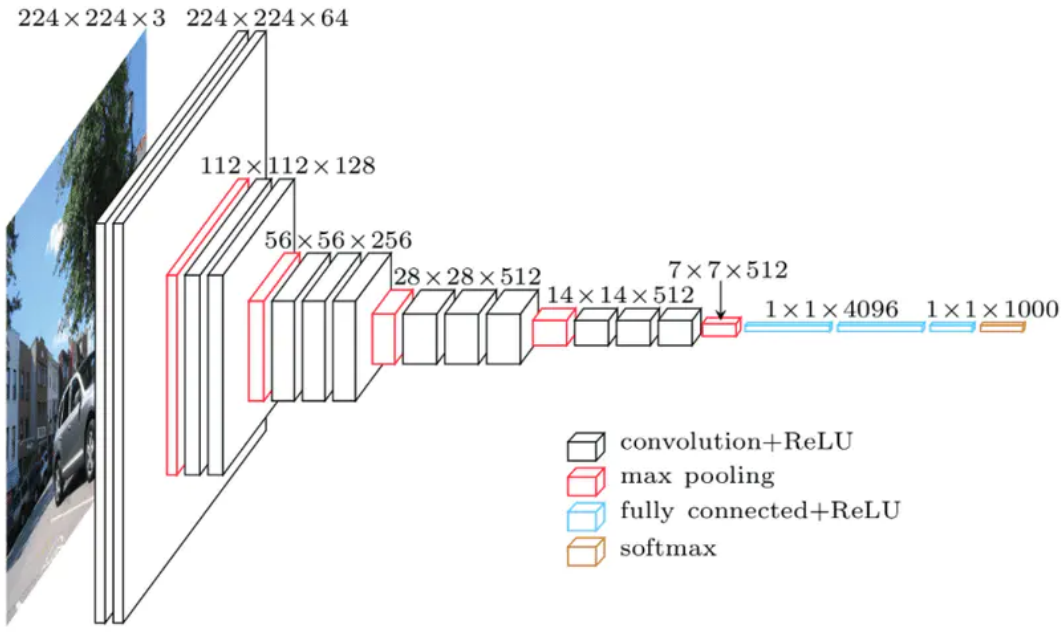


$Baseline$采用双通道的图像+文本模型，通过提取图像与文本的特征并进行联合后，传入一个全连接层并经过一个$Softmax$层获得问题对应各个答案的可能性大小。

- **图像特征提取**

  图像特征提取采用$VGGNet$，将图像特征信息转化为4096维的向量，并经过一个全连接层将信息压缩为1024维，便于与文本特征进行连接。

  $VGGNet$的具体结构如下：



- **文本特征提取**

使用一个具有一个隐藏层的$LSTM$来获得问题的1024维嵌入。从$LSTM$获得的嵌入是来自$LSTM$隐藏层的最后一个单元状态和最后一个隐藏状态表示（每个表示为512维的向量）的串联合并。每个问题词通过一个全连接层与一个$tanh$非线性层以300维的嵌入进行编码，然后将其输送给$LSTM$。嵌入层的输入词汇由训练数据集中看到的所有问题中的词语组成。

- **图像特征与文本特征连接**

  采用文本特征向量与图像特征向量的$Hadamard$积作为图片与问题信息的联合向量传入输出层。

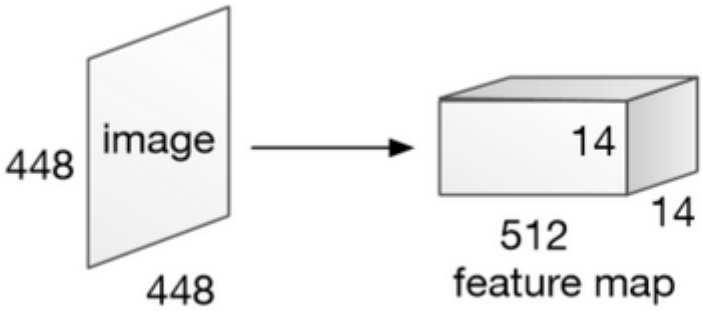
- 输出

  输出层由一个全连接层与一个$softmax$层构成，输出的维数为对应问题的候选答案个数。

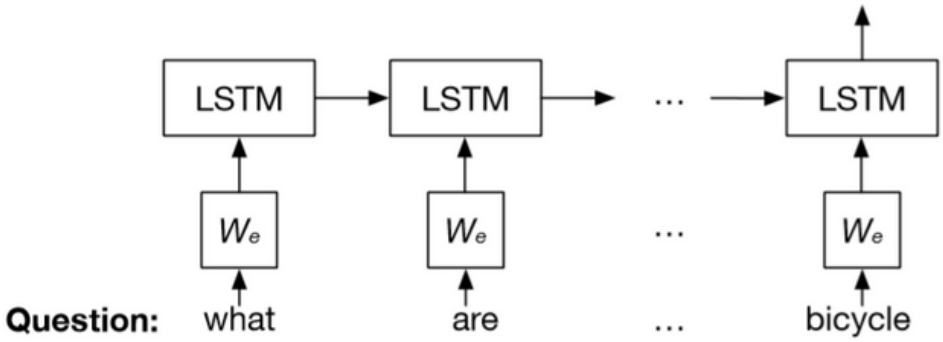
3.2 SAN模型

VQA问题的解决大多是层次性的，比如，当问题为：什么东西在车筐里？ 我们往往会先定义到自行车，再去注意车筐，再去看车筐里的东西。所以论文作者提出了注意力堆叠模型。 注意力的可视化如下图，白色高光部分为注意力集中区域，随着注意力堆叠层数增加，模型逐渐根据问题聚焦图片中对应的答案来源。

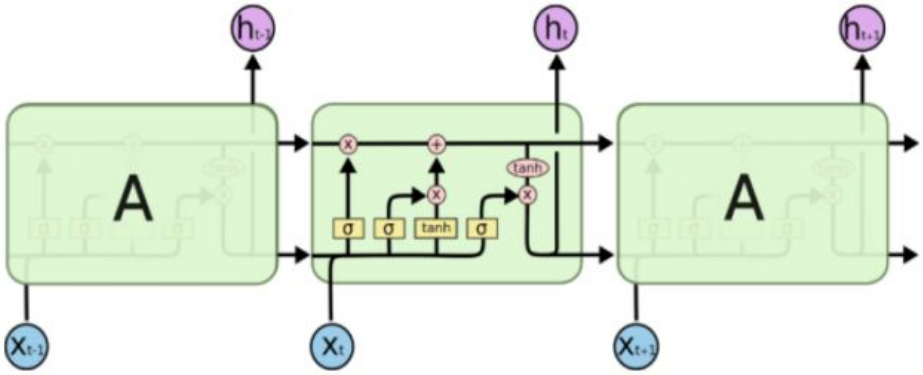
- **图片特征提取**



  - 选用$VGGNet$
  - 选取最后一个池化层作为特征输出14×14个512维的特征向量，每个特征向量代表原图32×32的区域。
  - 单层感知机将512维向量转化为与问题特征向量同维（方便注意力模型之后的操作）。
- **问题特征提取（LSTM或CNN）**
  - $LSTM$：$LSTM$是指长短期记忆网络，其是一种时间循环神经网络。$LSTM$主要用来解决RNN（循环神经网络）中存在的长期依赖问题。$LSTM$也是一种特殊的循环神经网络因此也具有链状结构，但是相比循环神经网络的重复模块有着不同的结构。它有四层神经网络层，各个网络层之间以特殊的方式相互作用，并非单个简单的神经网络层。下图是$LSTM$神经网络的基本结构。

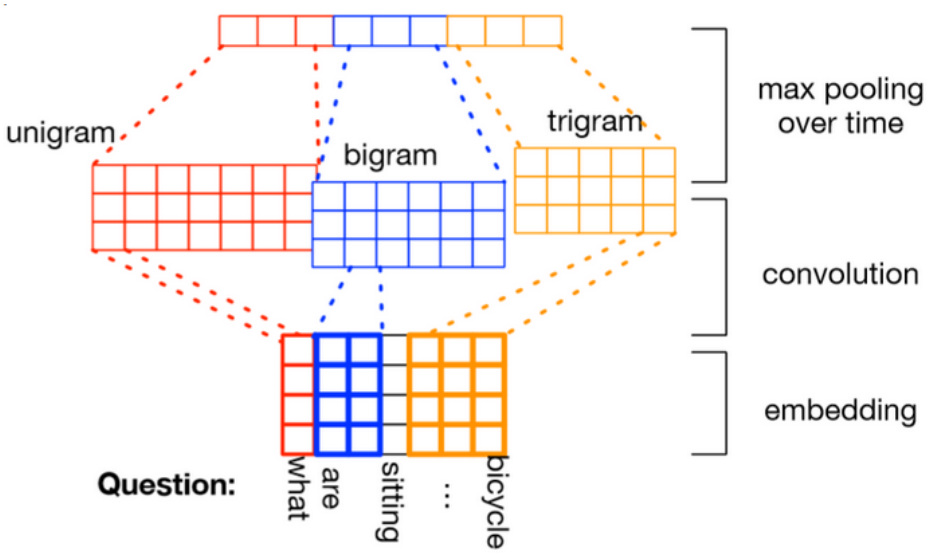


$LSTM$单元的具体结构如下：



在上图中，X表示缩放的信息，$+$表示添加的信息，σ表示$Sigmoid$层，$tanh$（双曲正切）表示$tanh$层，$h(t-1)$表示上一个$LSTM$单元的输出，$c(t-1)$表示上一个$LSTM$单元的记忆，$X(t)$表示输入，$c(t)$表示最新的记忆，$h(t)$表示输出。每个传输单元的状态是决定 $LSTM$ 网络的核心，就是穿过图中的每条水平线。一个单元状态就相当于一个传送带，它贯穿于整个结构，在这个过程中仅通过一些线性的作用保证了信息传输的不变性。$LSTM$还具有一个很好的性能就是可以增加和去除向单元状态中传输的信息，通过几个结构来管理信息的传输并将其称为门限，门限就是有选择地让信息通过。$LSTM$很好地改善了RNN中存在的信息长期以来的问题。

- CNN



- **堆叠注意力模型**

通常问题的解决只需要图中一小部分区域的信息，如果将所有区域完整的特征向量全部应用，会引入噪声影响结果。SAN能够逐步过滤噪声，指出与答案高度相关的区域。

如果我们现在得到了图像特征Vi和问题特征Vq，那么就可以利用SAN进行多步的推理过程，很多情况下，答案只与图像中的很小一部分有关，因此使用全局图像特征来预测将会导致结果变差（这是由于其他不相关区域作为噪声对答案产生了影响）。因此设计了SAN，通过多步过滤噪声，使得答案更具有针对性。

具体操作就是先将这两个特征传到一个单层神经网络，然后再用一个$softmax$层生成一个基于问题的图注意力分布：

$$h_A = tanh(W_{I,A}v_I \oplus (W_{Q,A}v_Q + b_A))$$
$$p_I = softmax(W_P h_A + b_P)$$

其中v_I为$d \times m$维的矩阵，d是为单个图片分区的特征向量维数，m为单个图片的分区总数。v_Q为d维的问题向量表示。$W_{I,A}, W_{Q,A}$均为$k \times d$维的矩阵，W_P为$1 \times k$维的向量，p_I为m维的向量与图片的分区数目对应。

基于注意力分布，我们可以重新计算图像向量的加权和，然后再将结果与问题向量相结合：

$$\tilde{v}_I = \sum_i p_i v_i$$
$$u = \tilde{v}_i + v_Q$$

相较于直接结合问题特征和图像特征，注意力模型能够更准确的定位到与答案有关的区域。然而对于复杂的问题，单层注意力层是不够的，因此采用多层注意力。第k层的注意力就可以表示为：

$$h_A^k = tanh(W_{I,A}^k v_I \oplus (W_{Q,A}^k u^{k-1} + b_A^k))$$
$$p_I^k = softmax(W_P^k h_A^k + b_P^k)$$

所以最终的向量就是与前一步的叠加：

$$\tilde{v}_I^k = \sum_i p_i^k v_i$$
$$u^k = \tilde{v}_I^k + u^{k-1}$$

重复上述步骤K次，那么最终用于推断答案的注意力则为：

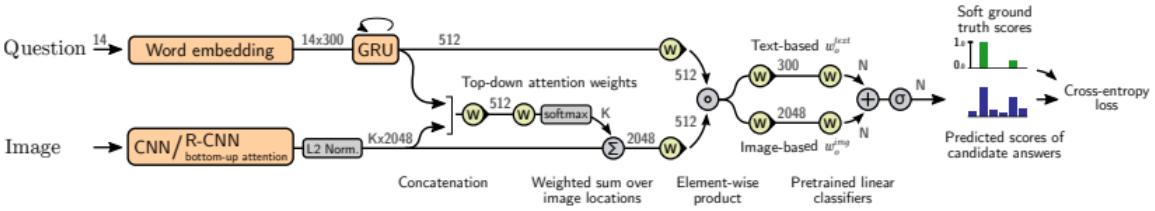
$$p_{ans} = softmax(W_u u^K + b_u)$$

- **论文中SAN可视化实例**

如上图可以观察到，模型较为正确地根据问题描述对于图片信息进行注意力聚焦。

3.3 $Topdown\ Attention$模型

该深度神经网络实现了输入图片及输入问题的联合嵌入，然后基于固定的候选答案集中实现多标签分类器。　灰色的数字表示层间的向量维数，黄色的单元表示网络中学习到的参数，其中圆形边框表示线性层，带尖角的圆形边框表示非线性层（$gated\ tanh$）。



- **问题特征提取**

  问题特征首先通过句子裁剪或补全将句子单词个数限定为14，并采用$GloVe$预训练模型将问题转化为14×300的嵌入表示。将$embedding$的问题作为出入传入内部维数为512的GRU单元，即在处理14个单词的嵌入后，作为我们的问题嵌入q。

- **图片特征提取**

  图片特征采用CNN进行提取，将图片转化为$K\times2048$的特征矩阵，其中K为图片分割的数量，各个分割后的部分的特征由一个2048维的向量表示。

- **Topdown Attention模型**

  模型实现了问题和图像的联合多模嵌入，并对一组候选答案的分数进行了回归的预测。

  网络内的学习非线性变换采用门控双曲正切（$gated\ tanh$）激活来实现的。我们的每个"门控$tanh$"层都实现了一个函数 f_a:

  $x\in\mathbb{R}^m\rightarrow y\in\mathbb{R}^n$，$a=\{W,W',b,b'\}$参数 定义如下:

  $$\tilde{y}=tanh(Wx+b)$$
  $$g=\sigma(W'x+b')$$
  $$y=\tilde{y}\circ g$$

  其中，σ 是 $sigmoid$ 激活函数，W,W'是可学习权重，$b,b'\in\mathbb{R}^n$是可学习偏差。$\circ$ 是逐元素乘积。向量 g 的乘积作用为充当中间激活 $\tilde{y}$ 的门。

  - **基于attention的图片特征(Image attention)**：首先将每个问题编码为门控循环单元（GRU）的隐藏状态 q，其中每个输入单词都使用学习单词嵌入来表示。 给定GRU的输出 q，我们为 k 个图像特征 v_i 中的每一个，生成未归一化的关注权重 a_i，如下所示:

    $$a_i=w_a^Tf_a([v_i,q])$$

    其中，w_a^T是一个可学习的参数向量。接着对于上述注意力权重进行$softmax$激活。然后，通过归一化值对所有位置的图像特征进行加权，并求和，以获得一个2048维向量，该向量表示所关注的图像。

    $$\alpha=softmax(a)$$
    $$\hat{v}=\sum_{i=1}^{K}\alpha_iv_i$$

  - **多模态融合(Multimodal fusion)**：之后将获得的问题的向量表示以及最终的图片向量表示传入非线性层并采用$Hadamard$乘积结合。其中，h 是问题和图像的联合表示。

    $$h=f_q(q)\circ f_v(\hat{v})$$

  - **分类器(Output classifier)**：h 作为输入传入输出分类器。首先将 h 传入一个非线性层并通过一个线性映射以预测各个候选答案的分数。其中$w_o\in\mathbb{R}^{N\times512}$为学习的权重矩阵。

    $$\hat{s}=\sigma(w_o,f_o(h))$$

-

3.4 技术细节

3.4.1 预训练参数和向量

预训练词向量：由于`Stack Attention`和`Topdown Attention`的文本特征提取部分都使用了预训练的词向量，我们采用了`Glove`200维的预训练词向量。对于那些不在`Glove`范围中的词，我们采用的策略是随机生成一个同样维度的向量，最后得到的映射表传入文本特征处理结构的嵌入层。

预训练参数：由于MindSpore并不提供可供本实验使用的用于特征提取的预训练模型，而像图片这样的复杂数据又需要比较好的预训练模型来提取特征，这使得完全不考虑引入预训练参数的模型实际分类效果非常糟糕。通过查找MindSpore的资源，我们在MindSpore Hub中查找到了`VGG 19`的`ckpt`文件。该文件是由`VGG 19 net`在`ImageNet`上处理分类而训练得到的。ckpt文件里面保存的是各层网络结构参数组织而成的字典，因此我们可以通过仔细观察字典结构的方式从而将参数手动加载到我们自己编写的`VGG net`结构中。

值得说明的是这些参数在训练过程中是冻结的，以及不会随训练而更新。

3.4.2 checkpoint的保存机制

为了能够得到比较好的模型参数，我们的ckpt保存机制比较复杂：在每一轮训练结束后，我们会进行一轮在验证集上的测试并得到对应的loss和accuracy。我们的ckpt保存机制就基于模型在验证集上的表现，只有当前这组模型参数对应的loss和accuracy都是当前最优情况时才更新ckpt文件。

此外我们还设置了**早退机制**，当模型在给定的步数中都没有能够超过先前的loss和accuracy时我们认为此时模型已经过拟合或收敛了，此时直接完全退出训练过程。

3.4.3 超参数设置

参数名	值
词典大小(vocab_size)	52083
分类数大小(output_size)	999 for stack_attention and baseline & 2958 for topdown_attention
批量数(batch_size)	32 for baseline & 100 for stack_attention & 128 for topdown_attention
训练轮数(epoch_size)	3
截断长度(max_length)	14
学习率(lr)	1e-4
动量(momentum)	0.9
权重衰减率(weight_decay)	3e-5
早退步数(early_stop)	100

四.实验内容及步骤

4.1 数据集特性观察

在实验开始前，我们首先对数据集整体特征进行了分析：

数据集特性	
最长问题词数	25（包含逗号）
最长回答词数	19（包含逗号）
问题+回答选项词典词数	52083（包含 标点+）
答案数量	8193
只取答案长度为1的vqa组合	
样本总数	80609
训练集样本数	40957
验证集样本数	19831
测试集样本数	19821
把答案当成一个整体，vqa的组合数就不会减少	
出现频率大于等于1	8193
出现频率大于等于2	2958
出现频率大于等于3	1935
出现频率大于等于4	1470
出现频率大于等于5	1184
出现频率大于等于6	999
出现频率大于等于7	866
出现频率大于等于8	782
出现频率大于等于9	713
出现频率大于等于10	654

作为一个多分类问题，每个答案都意味着一个分类，所以如果只是不做处理直接将答案作为分类，更难被正确分类。因此我们统计了答案的出现频率，根据论文中模型设计的分类数选取答案数最接近的出现频率，选取出现频率大于等于2(2958)的答案作为`Top-down Attention`的输入(论文中分类数3000)，出现频率大于等于6(999)的答案作为`baseline`以及`Stack Attention`的输入(论文中分类数1000)。

对于出现频率大于等于2，我们覆盖了93%的答案

对于出现频率大于等于6，我们覆盖了87%的答案

在预处理的时候，我们观察到了答案的类型分布如下：

答案类型	yes/no	number	other	all
分类数	2	327	7864	8193
答案频率	33063	10442	43740	43740

4.2 预处理

在我们的实现中单个样例包括四种数据：图片image，问题question，答案answer，答案备选项options，在读取json文件的时候完成对单个样例这四个属性的聚合。

- image

在生成MindRecord文件时图片按照原样进行存储，而在读取MindRecord文件生成dataset时根据指定的图片长宽调用MindSpore提供的Transform中的接口对图片大小进行重塑。对于baseline而言，重塑后的图片大小规格为（224，224，3），对于另外两个模型而言，该规格为（448，448，3）。

此外在使用feature map时存储的就直接是图片对应的（36，2048）的float32类型的向量，之后可以直接传入模型。

- question

同样地，在生成MindRecord文件时我们保留了全部的文本信息，采用了所有文本的最大长度作为文本存储的统一长度。而在读取MindRecord文件生成dataset时，则出于性能的考虑，对文本数据做了截断。如我们最后采用了14作为截断长度，而最长的文本含有25个词。

- answer

answer不进行分词，以整体为单位进行编码。最后将所有可能的answer映射到$0，1，2，....，size-1$以和分类器结果相对应。

- options

每个样例对应了十个备选答案，出于简化处理，我们对每个options的备选答案，我们仅取了第一个词（单个词的答案占了所有答案的90%）。options和question一起进行分词并构建词表。

4.3 实验步骤

- **步骤1 下载数据集到本地并将其连通源码上传到obs**

  我们按照json文件中的索引对数据进行预处理，注意到原有数据集中缺失部分图片导致和答案无法对应，于是我们从VQA: Visual Question Answering (visualqa.org)下载完整的COCO数据集作为我们后续的原始数据集。该数据集(18.6G)是原有数据集的超集(11G)，我们相信使用更大的数据集作为输入，也有助于提升我们模型的泛化能力。

- **步骤2 下载源码和数据集到本地容器**

  因为notebook是挂载在obs上，运行的容器实例不能直接读取操作obs上的文件，需下载至容器本地环境中。

```
import moxing as mox
mox.file.copy_parallel(src_url=obs_path+"mindrecord",
dst_url='./mindrecord')
mox.file.copy_parallel(src_url=obs_path+"preprocess",
dst_url='./preprocess')
mox.file.copy_parallel(src_url=obs_path+"utils",      dst_url='./utils')
mox.file.copy_parallel(src_url=obs_path+"model",      dst_url='./model')
mox.file.copy_parallel(src_url=obs_path+"ckpt",       dst_url='./ckpt')
mox.file.copy_parallel(src_url=obs_path+"pretrained/VGG.py",
dst_url='./pretrained/VGG.py')
mox.file.copy_parallel(src_url=obs_path+"pretrained/glove.6B.200d.word2vec.t
xt", dst_url='./pretrained/glove.6B.200d.word2vec.txt')
mox.file.copy_parallel(src_url=obs_path+"pretrained/embeddings.py",
dst_url='./pretrained/embeddings.py')
mox.file.copy_parallel(src_url=obs_path+"pretrained/__init__.py",
dst_url='./pretrained/__init__.py')
# mox.file.copy_parallel(src_url=obs_path+"data",       dst_url='./data')
```

- **步骤3 导入依赖库**

```
import mindspore
import numpy as np
import os
from easydict import EasyDict
from preprocess.preprocess import *
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"  # 允许重复载入lib文件
```

- **步骤4 预处理**
  - 预处理配置

```
padding = '<pad>'
config = EasyDict({
    'train_img_path': './data/images/train/COCO_train2014_',
    'train_ans_path': './data/annotations/train.json',
    'train_que_path': './data/questions/train.json',
    'valid_img_path': './data/images/val/COCO_val2014_',
    'valid_ans_path': './data/annotations/val.json',
    'valid_que_path': './data/questions/val.json',
    'test_img_path': './data/images/test/COCO_val2014_',
    'test_ans_path':  './data/annotations/test.json',
    'test_que_path':  './data/questions/test.json',
    'max_length': 25,
    'dict_path': './mindrecord/dict.npy',
    'idx_word_dict_path': './mindrecord/idx_word_dict.npy',
    'filter_set_path': './mindrecord/',
    'pretrained_path': './pretrained/',
    'num_splits': 1,
    'train_mindrecord_path': './mindrecord/train.mindrecord',
    'valid_mindrecord_path': './mindrecord/valid.mindrecord',
    'test_mindrecord_path':  './mindrecord/test.mindrecord',
    'train_featurerecord_path': './featurerecord/',
    'valid_featurerecord_path': './featurerecord/',
    'test_featurerecord_path': './featurerecord/'
})
```

- 读取数据

  注：只取那些答案长度为1的**vqa组合**

```
# get 3 types of input data
train_images, train_questions, train_answers, train_options =
get_list(config.train_que_path, config.train_ans_path)
valid_images, valid_questions, valid_answers, valid_options =
get_list(config.valid_que_path, config.valid_ans_path)
test_images,  test_questions,  test_answers, test_options  =
get_list(config.test_que_path,  config.test_ans_path)

total_questions = train_questions + valid_questions + test_questions
total_answers   = train_answers + valid_answers + test_answers
total_options   = train_options + valid_options + test_options
```

- 构建词典

```
# build word vocab
word_dict = dict({'<pad>': 0})
word_dict = add_word_into_dict(total_questions, word_dict)
word_dict = add_word_into_dict(total_options, word_dict)

answer_dict = dict()
answer_dict = add_answer_into_dict(total_answers, answer_dict)

# build revert dict
idx_word_dict = dict()
for item in word_dict.items():
    idx_word_dict[item[1]] = item[0]

# save dict
np.save(config.dict_path, word_dict)
np.save(config.idx_word_dict_path, idx_word_dict)

mox.file.copy_parallel(src_url="./mindrecord/dict.npy",
dst_url=obs_path+"mindrecord/dict.npy")
mox.file.copy_parallel(src_url="./mindrecord/idx_word_dict.npy",
dst_url=obs_path+"mindrecord/idx_word_dict.npy")
```

- 向量化和补齐长度

```
# word -> vector & padding
train_questions_vec = get_vec_and_pad(train_questions, word_dict,
config.max_length)
valid_questions_vec = get_vec_and_pad(valid_questions, word_dict,
config.max_length)
test_questions_vec = get_vec_and_pad(test_questions, word_dict,
config.max_length)

train_options_vec = get_option_vec_and_pad(train_options, word_dict, 1)
valid_options_vec = get_option_vec_and_pad(valid_options, word_dict, 1)
test_options_vec = get_option_vec_and_pad(test_options, word_dict, 1)

train_answers_vec, _, _, _ = get_answer_to_idx(train_answers, answer_dict)
valid_answers_vec, _, _, _ = get_answer_to_idx(valid_answers, answer_dict)
test_answers_vec, _, _, _ = get_answer_to_idx(test_answers, answer_dict)
_, bool_set, num_set, other_set = get_answer_to_idx(total_answers,
answer_dict)
```

```
# 保存集合
np.save(config.filter_set_path+'bool.npy', bool_set)
np.save(config.filter_set_path+'num.npy', num_set)
np.save(config.filter_set_path+'other.npy', other_set)
```

```
# 删除 data 文件夹下的文件，以节省空间
import os

for root,dirs,files in os.walk("./data"):
    for file in files:
        os.remove(root+"/"+file)

# 在服务器上创建 featurerecord 文件夹
! mkdir featurerecord


train_combine_dict = dict()
for i, q, a, o in zip(train_images, train_questions_vec, train_answers_vec,
train_options_vec):
    j=0
    while train_combine_dict.__contains__((i, j)):
        j += 1
    train_combine_dict[(i, j)] = (q, a, o)

valid_combine_dict = dict()
for i, q, a, o in zip(valid_images, valid_questions_vec, valid_answers_vec,
valid_options_vec):
    j=0
    while valid_combine_dict.__contains__((i, j)):
        j += 1
    valid_combine_dict[(i, j)] = (q, a, o)

test_combine_dict = dict()
for i, q, a, o in zip(test_images, test_questions_vec, test_answers_vec,
test_options_vec):
    j=0
    while test_combine_dict.__contains__((i, j)):
        j += 1
```

```
    test_combine_dict[(i, j)] = (q, a, o)
train_data_dict = dict()
valid_data_dict = dict()
test_data_dict = dict()

# id in range(1, 14)
id = 1
mox.file.copy_parallel(src_url=obs_path+"pretrained/feature_map" + str(id) +
".npy", dst_url='./pretrained/feature_map' + str(id) + '.npy')
feature_map_set = np.load('./pretrained/feature_map' + str(id) + '.npy',
allow_pickle=True).item()

for key, value in feature_map_set.items():
    j = 0
    while train_combine_dict.__contains__((key, j)):
        train_data_dict[(key, j)] = (value, train_combine_dict[(key, j)])
        j += 1
    j = 0
    while valid_combine_dict.__contains__((key, j)):
        valid_data_dict[(key, j)] = (value, valid_combine_dict[(key, j)])
        j += 1
    j = 0
    while test_combine_dict.__contains__((key, j)):
        test_data_dict[(key, j)] = (value, test_combine_dict[(key, j)])
        j += 1

try:
    os.remove('./pretrained/feature_map' + str(id) + '.npy')
    print("-- delete file successfully!")
except(FileNotFoundError):
    print("-- file not exists!")
```

- 取频率较高的那些词作为答案词集

```
total_answers_vec = train_answers_vec + valid_answers_vec + test_answers_vec
least_2_set = get_filtered_answer_set(total_answers_vec, 2) # 2958
least_6_set = get_filtered_answer_set(total_answers_vec, 6) # 999

# 保存集合
np.save(config.filter_set_path+'min2.npy', least_2_set)
np.save(config.filter_set_path+'min6.npy', least_6_set)
```

- **步骤5 生成MindRecord**
  - train

```
generate_mindrecord(config.train_mindrecord_path, config.train_img_path,
config.num_splits, train_images, train_questions_vec, train_answers_vec,
train_options_vec)

mox.file.copy_parallel(src_url="./mindrecord/train.mindrecord",
 dst_url=obs_path+"mindrecord/train.mindrecord")
mox.file.copy_parallel(src_url="./mindrecord/train.mindrecord.db",
dst_url=obs_path+"mindrecord/train.mindrecord.db")
```

  - valid

```
generate_mindrecord(config.valid_mindrecord_path, config.valid_img_path,
config.num_splits, valid_images, valid_questions_vec, valid_answers_vec,
valid_options_vec)

mox.file.copy_parallel(src_url="./mindrecord/valid.mindrecord",
 dst_url=obs_path+"mindrecord/valid.mindrecord")
mox.file.copy_parallel(src_url="./mindrecord/valid.mindrecord.db",
dst_url=obs_path+"mindrecord/valid.mindrecord.db")
```

- test

```
generate_mindrecord(config.test_mindrecord_path, config.test_img_path,
config.num_splits, test_images, test_questions_vec, test_answers_vec,
test_options_vec)

mox.file.copy_parallel(src_url="./mindrecord/test.mindrecord",
 dst_url=obs_path+"mindrecord/test.mindrecord")
mox.file.copy_parallel(src_url="./mindrecord/test.mindrecord.db",
dst_url=obs_path+"mindrecord/test.mindrecord.db")
```

- **步骤6 加载词典、集合**

```
# load dict
word_dict = np.load(config.dict_path, allow_pickle=True).item()
idx_word_dict = np.load(config.idx_word_dict_path, allow_pickle=True).item()

# load filter set
least_2_set = np.load(config.filter_set_path+'min2.npy',
allow_pickle=True).item()
least_6_set = np.load(config.filter_set_path+'min6.npy',
allow_pickle=True).item()
bool_set = np.load(config.filter_set_path+'bool.npy', allow_pickle=True).item()
num_set = np.load(config.filter_set_path+'num.npy', allow_pickle=True).item()
other_set = np.load(config.filter_set_path+'other.npy',
allow_pickle=True).item()

# set -> dict
least_2_dict = set_to_dict(least_2_set)
least_6_dict = set_to_dict(least_6_set)

bool_dict = set_to_dict(bool_set)
num_dict = set_to_dict(num_set)
other_dict = set_to_dict(other_set)
```

- **步骤七 训练配置**

  对于本实验，我们根据论文实现了三种模型，分别为`Baseline`，`Stack Attention`和`Topdown Attention`，实际训练时根据模型选择不同设置不同的`model_name`和`batch_size`

```
# model_name = 'baseline'
# model_name = 'stack_attention'
model_name = 'topdown_attention'


# batch_size = 32 # baseline
# batch_size = 100 # stacked attention
batch_size = 128 # top-down attention
```

```
train_config = EasyDict({
    'model': model_name,
    'vocab_size': 52083,
    'output_size': 999 if model_name == 'baseline' or model_name ==
'stack_attention' else 2958,
    'batch_size': batch_size,
    'epoch_size': 3,
    'max_length': 14,
    'use_op': False,
    'use_pretrained_feature_map': True,
    'hidden_size': 1024,
    'lr': 1e-4,
    'momentum': 0.9,
    'weight_decay': 3e-5,
    'early_stop': 100,
    'ckpt_save_path': './ckpt',
    'checkpoint_path': './ckpt/'+model_name+'.ckpt',
    'pretrained_path':
'./pretrained/vgg19_ascend_v130_imagenet2012_research_cv_top1acc74_top5acc91.97.
ckpt',
    'embedding_table_path': './pretrained/embedding_table_glove_200d.txt',
    'glove_vector_path': './pretrained/glove.6B.200d.txt',
    'glove_word2vec_path': './pretrained/glove.6B.200d.word2vec.txt',
    'train_featurerecord_path': './featurerecord/',
    'valid_featurerecord_path': './featurerecord/',
    'test_featurerecord_path': './featurerecord/'
})
```

- **步骤8 生成数据集**

```
# 受限于计算资源和额度，我们尝试使用多个训练平台完成训练
# platform = 'Local'
# platform = 'Colob'
platform = 'Ascend'
```

```
# frequency filter dict
filter_dict = least_6_dict if model_name == 'baseline' or model_name ==
'stack_attention' else least_2_dict
# image size
image_width = 224 if model_name == 'baseline' else 448
image_height = 224 if model_name == 'baseline' else 448
# parallel workers
num_parallel_workers = 4 if platform == 'Ascend' or platform == 'Local' else 2
print(num_parallel_workers)
# create dataset
```

```
train_dataset = generate_dataset(config.train_mindrecord_path,
train_config.batch_size, 1, train_config.max_length,
                                 filter_dict, image_height, image_width,
num_parallel_workers)
valid_dataset = generate_dataset(config.valid_mindrecord_path,
train_config.batch_size, 1, train_config.max_length,
                                 filter_dict, image_height, image_width,
num_parallel_workers)
test_dataset  = generate_dataset(config.test_mindrecord_path ,
train_config.batch_size, 1, train_config.max_length,
                                 filter_dict, image_height, image_width,
num_parallel_workers)
```

- **步骤9 创建训练模型**

```
import mindspore.nn as nn
import mindspore.ops.operations as P
from utils.metric_utils import *
from utils.wrapper_utils import *
from utils.callback_utils import *
```

```
from model.vqa_baseline import *
from model.stack_attention import *
from model.topdown_attention import *
```

```
# 创建网络
if model_name == 'baseline':
    if train_config.use_op:
        network = VQABasicOpAttn(train_config)
    else:
        network = VQABasic(train_config)
elif model_name == 'stack_attention':
    if train_config.use_op:
        network = StackedAttentionNetOpAttn(word_dict, train_config)
    else:
        network = StackedAttentionNet(word_dict, train_config)
elif model_name == 'topdown_attention':
    if train_config.use_op and train_config.use_pretrained_feature_map:
        network = TopDownAttentionNetOpAttn(word_dict, train_config)
    elif train_config.use_op:
        network = TopDownAttentionNetOpAttn(word_dict, train_config)
    elif train_config.use_pretrained_feature_map:
        network = TopDownAttentionNetFeature(word_dict, train_config)
    else:
        network = TopDownAttentionNet(word_dict, train_config)
```

```
# 创建训练、测试网络
loss_fn = nn.SoftmaxCrossEntropyWithLogits(sparse=True)
# train network
train_net = TrainNetworkWrapper(network, loss_fn, train_config)
train_net.set_train(True)
# valid network
valid_net = WithEvalCellWrapper(network, train_config)
valid_net.set_train(False)
```

- **步骤10 开始训练**

```
def train(train_net, valid_net, train_dataset, valid_dataset, train_config):
    # 创建文件夹
    if not os.path.exists(train_config.ckpt_save_path):
        os.mkdir(train_config.ckpt_save_path)

    current_step = 0
    valid_acc_max = 0.0
    valid_loss_min = np.inf
    valid_acc_model = 0
    valid_loss_model = np.inf
    for epoch_num in range(1, train_config.epoch_size+1):
        # train
        train_losses = []
        train_accs = []
        for i in train_dataset.create_dict_iterator():
            train_loss, train_acc = train_net(i['image'], i['question'],
i['answer'], i['options'])
            train_losses.append(train_loss.item(0).asnumpy().item())
            train_accs.append(train_acc.item(0).asnumpy().item())
        train_loss = sum(train_losses) / len(train_losses)
        train_acc = sum(train_accs) / len(train_accs)
        print('epoch:', epoch_num, ' train loss =', train_loss, 'acc =',
train_acc)

        # valid
        loss = []
        acc = []
        for j in valid_dataset.create_dict_iterator():
            step_loss, step_acc = valid_net(j['image'], j['question'],
j['answer'], j['options'])
            loss.append(step_loss.item(0).asnumpy().item())
            acc.append(step_acc.item(0).asnumpy().item())
        valid_loss = sum(loss) / len(loss)
        valid_acc = sum(acc) / len(acc)
        print('-- valid loss =', valid_loss, 'acc =', valid_acc)

        # save ckpt / early stop
        if valid_acc >= valid_acc_max or valid_loss < valid_loss_min:
            if valid_acc >= valid_acc_max and valid_loss < valid_loss_min:
                valid_acc_model = valid_acc
                valid_loss_model = valid_loss
                save_checkpoint(valid_net.network, train_config.checkpoint_path)
                valid_acc_max = np.max((valid_acc_max, valid_acc))
                valid_loss_min = np.min((valid_loss_min, valid_loss))
            current_step = 0
        else:
            current_step += 1
            if current_step == train_config.early_stop:
                print('early stop... min loss:', valid_loss_min, 'max acc:',
valid_acc_max, end='')
                print('; validation model loss:', valid_loss_model, 'acc:',
valid_acc_model)
```

```
import os
```

```
def train_feature_map(train_net, valid_net, train_config, filter_dict):
    # 创建文件夹
    if not os.path.exists(train_config.ckpt_save_path):
        os.mkdir(train_config.ckpt_save_path)

    current_step = 0
    valid_acc_max = 0.0
    valid_loss_min = np.inf
    valid_acc_model = 0
    valid_loss_model = np.inf
    for epoch_num in range(1, train_config.epoch_size+1):
        # train
        train_losses = []
        train_accs = []
        for id in range(1, 2):
            mox.file.copy_parallel(src_url=
obs_path+train_config.train_featurerecord_path + "train" + str(id) +
".mindrecord",    dst_url=train_config.train_featurerecord_path + "train" +
str(id) + ".mindrecord")
            train_dataset =
generate_feature_map_dataset(train_config.train_featurerecord_path +
'train'+str(id)+'.mindrecord', train_config.batch_size, 1,
train_config.max_length, filter_dict)
            for i in train_dataset.create_dict_iterator():
                train_loss, train_acc = train_net(i['image'], i['question'],
i['answer'], i['options'])
                train_losses.append(train_loss.item(0).asnumpy().item())
                train_accs.append(train_acc.item(0).asnumpy().item())
            # delete file
            try:
                os.remove(train_config.train_featurerecord_path + "train" +
str(id) + ".mindrecord")
                print("-- delete file successfully!")
            except(FileNotFoundError):
                print("-- file not exists!")

        train_loss = sum(train_losses) / len(train_losses)
        train_acc = sum(train_accs) / len(train_accs)
        print('epoch:', epoch_num, ' train loss =', train_loss, 'acc =',
train_acc)

        # valid
        loss = []
        acc = []
        for id in range(1, 2):
            mox.file.copy_parallel(src_url=
obs_path+train_config.train_featurerecord_path + "valid" + str(id) +
".mindrecord",    dst_url=train_config.train_featurerecord_path + "valid" +
str(id) + ".mindrecord")
            valid_dataset =
generate_feature_map_dataset(train_config.train_featurerecord_path +
'valid'+str(id)+'.mindrecord', train_config.batch_size, 1,
train_config.max_length, filter_dict)
            for j in valid_dataset.create_dict_iterator():
                step_loss, step_acc = valid_net(j['image'], j['question'],
j['answer'], j['options'])
                loss.append(step_loss.item(0).asnumpy().item())
```

```
                acc.append(step_acc.item(0).asnumpy().item())
            # delete file
            try:
                os.remove(train_config.train_featurerecord_path + "valid" +
str(id) + ".mindrecord")
                print("-- delete file successfully!")
            except(FileNotFoundError):
                print("-- file not exists!")

        valid_loss = sum(loss) / len(loss)
        valid_acc = sum(acc) / len(acc)
        print('-- valid loss =', valid_loss, 'acc =', valid_acc)

        # save ckpt / early stop
        if valid_acc >= valid_acc_max or valid_loss < valid_loss_min:
            if valid_acc >= valid_acc_max and valid_loss < valid_loss_min:
                valid_acc_model = valid_acc
                valid_loss_model = valid_loss
                save_checkpoint(valid_net.network, train_config.checkpoint_path)
                valid_acc_max = np.max((valid_acc_max, valid_acc))
                valid_loss_min = np.min((valid_loss_min, valid_loss))
            current_step = 0
        else:
            current_step += 1
            if current_step == train_config.early_stop:
                print('early stop... min loss:', valid_loss_min, 'max acc:',
valid_acc_max, end='')
                print('; validation model loss:', valid_loss_model, 'acc:',
valid_acc_model)
```

```
if train_config.use_pretrained_feature_map:
    train_feature_map(train_net, valid_net, train_config, filter_dict)
else:
    train(train_net, valid_net, train_dataset, valid_dataset, train_config)
```

```
mox.file.copy_parallel(src_url=train_config.checkpoint_path,
dst_url=obs_path+train_config.checkpoint_path)
```

- **步骤11 创建测试模型**

```
from mindspore import load_checkpoint
from model.vqa_baseline import *
from model.stack_attention import *
from model.topdown_attention import *
```

```
# 创建网络
if model_name == 'baseline':
    if train_config.use_op:
        network = VQABasicOpAttn(train_config)
    else:
        network = VQABasic(train_config)
elif model_name == 'stack_attention':
    if train_config.use_op:
        network = StackedAttentionNetOpAttn(word_dict, train_config)
    else:
```

```
        network = StackedAttentionNet(word_dict, train_config)
elif model_name == 'topdown_attention':
    if train_config.use_op and train_config.use_pretrained_feature_map:
        network = TopDownAttentionNetOpAttn(word_dict, train_config)
    elif train_config.use_op:
        network = TopDownAttentionNetOpAttn(word_dict, train_config)
    elif train_config.use_pretrained_feature_map:
        network = TopDownAttentionNetFeature(word_dict, train_config)
    else:
        network = TopDownAttentionNet(word_dict, train_config)
```

```
load_checkpoint(train_config.checkpoint_path, net=network)
test_net = WithEvalCellWrapper(network, train_config)
test_net.set_train(False)
```

- **步骤12 启动测试**

```
def test(test_net, test_dataset):
    # test
    loss = []
    acc = []
    for i in test_dataset.create_dict_iterator():
        step_loss, step_acc = test_net(i['image'], i['question'], i['answer'],
i['options'])
        loss.append(step_loss.item(0).asnumpy().item())
        acc.append(step_acc.item(0).asnumpy().item())
    test_loss = sum(loss) / len(loss)
    test_acc = sum(acc) / len(acc)
    print('test loss =', test_loss, 'acc =', test_acc)
```

```
def test_feature_map(test_net, train_config, filter_dict):
    # test
    loss = []
    acc = []
    for id in range(1, 14):
        mox.file.copy_parallel(src_url=
obs_path+train_config.train_featurerecord_path + "test" + str(id) +
".mindrecord",    dst_url=train_config.train_featurerecord_path + "test" +
str(id) + ".mindrecord")
        test_dataset =
generate_feature_map_dataset(train_config.train_featurerecord_path +
'test'+str(id)+'.mindrecord', train_config.batch_size, 1,
train_config.max_length, filter_dict)
        for i in test_dataset.create_dict_iterator():
            step_loss, step_acc = test_net(i['image'], i['question'],
i['answer'], i['options'])
            loss.append(step_loss.item(0).asnumpy().item())
            acc.append(step_acc.item(0).asnumpy().item())
        # delete file
        try:
            os.remove(train_config.train_featurerecord_path + "test" + str(id) +
".mindrecord")
            print("-- delete file successfully!")
        except(FileNotFoundError):
            print("-- file not exists!")
```

```
    test_loss = sum(loss) / len(loss)
    test_acc = sum(acc) / len(acc)
    print('test loss =', test_loss, 'acc =', test_acc)
```

```
if config.use_pretrained_feature_map:
    test_feature_map(test_net, test_dataset)
else:
    test(test_net, train_config, filter_dict)
```

五.评价指标及实验结果

5.1 评价指标

我们在本次实验中使用的评价指标为$loss$和$accuracy$，其中主要考虑模型在测试集上的$accuracy$。

具体实现上，我们在网络外层封装了一个计算并返回$loss$和$accuracy$的功能层，在网络得到预测值之后分别调用计算$loss$和$accuracy$的模块来得到相应的数值。

$loss$采用交叉熵损失函数，$accuracy$则直接比对标签值以统计预测值最大的分类的正确率。

5.2 实验结果

- **Without options**

	baseline	*stack attention*	*topdown attention*
acc	22.02%	27.63%	21.93%

- **Options**

	baseline	*stack attention*	*topdown attention*
acc	41.43%	62.18%	38.05%

- **baseline_optionAttn**
  - train

```
epoch: 1  train loss = 4.960002673824801 acc = 0.28249007936507936
-- valid loss = 4.782902529065028 acc = 0.2836322869955157
epoch: 2  train loss = 4.671070379096192 acc = 0.2875631313131313
-- valid loss = 4.62756761307139 acc = 0.31109865470852016
epoch: 3  train loss = 4.584912389387816 acc = 0.3567595598845599
-- valid loss = 4.585018190389613 acc = 0.42213191330343797
```

  - test

```
test loss = 4.598374321856307 acc = 0.41428437967115095
```

- **stack_attention_optionAttn**
  - train

```
epoch: 1  train loss = 5.193868891111077 acc = 0.6037246047247949
-- valid loss = 3.7597771292534943 acc = 0.5374299034337017
epoch: 2  train loss = 2.789524493045247 acc = 0.5313769724245265
-- valid loss = 2.264837041079441 acc = 0.5634579403656665
epoch: 3  train loss = 1.9963058284389246 acc = 0.5880135419540965
-- valid loss = 1.785006582736969 acc = 0.6201401896008821
```

  - test

```
test loss = 1.8071201499377456 acc = 0.6217757003886677
```

- **topdown_optionAttn**
  - train

```
epoch: 1  train loss = 7.478137924492015 acc = 0.3826544436416185
-- valid loss = 7.463242893447419 acc = 0.3710235778443114
epoch: 2  train loss = 7.375963361277057 acc = 0.3765354046242775
-- valid loss = 7.310383939457511 acc = 0.3808476796407186
epoch: 3  train loss = 7.192918788490957 acc = 0.37732568641618497
-- valid loss = 7.114494326585781 acc = 0.3810348053892216
```

  - test

```
test loss = 7.129115087543419 acc = 0.3805202095808383
```

六.总结与分析讨论

观察各种模型的训练结果，不难发现，引入$attention$机制的网络结构能够更加充分的挖掘图片和文字的信息从而在本任务下取得更好的结果。

由于MindSpore缺乏一些有效的预训练模型，因此我们实现的$topdown\ attention$网络结构在获得了$fusion\ vector$后无法进一步凭借预训练参数矩阵分化出更加充分的把握了任务信息的文本、图片向量。因此在结果上由于后续的操作实际上使得前面已经较好结合的信息在过多的网络结构中被部分丢失，因此训练结果会和$baseline$比较接近，甚至有时会出现更糟糕的结果。为此我们遵照论文引入了通过`faster R-CNN + visual genome + ResNet`得到的每张图片对应的36个2048维的$feature\ map$从而在一开始获得更精准丰富的图片特征以此实现比较好的分类效果。

而比对$stack\ attention\ network$和$baseline$可以看到通过两层$stack\ attention\ layer$模型有效地借助了文本信息，实现了对图片局部内容的更精准的定位，而这样的定位往往就是或者非常接近问题的答案，因而也就不难解释为什么$stack\ attention$ 网络在性能上会高出$baseline$不少。

引入$options$而取得的性能提升是非常直观且自然的：相当于是模型从完成填空题变成了完成选择题。这时综合了图片和文本的信息——也即模型对当前问题的理解就可以用于在备选集中做$attention$，也就是挑选出模型认为最有可能的答案。