


Visual Question Answering

1. Project Introduction


(1) 选题

VQA(Visual Question Answering),即视觉问题回答任务，VQA 系统将一个图像和一个关于图像的自由而开放式自然语言的问题作为输入，并生成一个自然语言的答案作为输出。




Q: How many giraffes walking near a hut in an enclosure ?

A: two



Q: What is the color of the bus ?

A: yellow



Q: What next to darkened display with telltale blue ?

A: keyboard

VQA 和其他 CV 任务的区别是，它回答的问题直到 run time 才确定，因此它更接近 general image understanding。VQA 比 image caption 更复杂，因为它往往需要图像以外的信息。另外 VQA 的答案往往更简短，因此也更容易和 ground truth 比较。VQA 的难点在于图像是高维信息，缺少语法规则和结构，也不能直接使用句法分析、正则表达式等 NLP 方法。此外，图像捕捉到的信息更接近现实世界，而语言本身就是一种抽象。

(2) 工作简介

在本任务中,我们使用 VQAv2.0 数据集的子集进行训练与测试,整体工作包括：数据处理、模型构建、训练与测试。在数据处理中，我们从网站下载数据集，提取 train、val、test 三部分的 image、question、annotation，并将它们进行一一对应，保存在 mindrecord 中。在模型构建中，我们构建了简单的 embedding 模型、Clip 预训练模型对 image 和 question 进行嵌入，并进行了点乘、MFB 模型、attention 机制等多种方式处理嵌入向量，得到输出。在训练与测试中，我们进行了 10 个 epoch 的训练，并调整了超参数如学习率、dropout 率等以获得最佳效果。最后，我们记录了准确率以对比实验效果。

(3) 开发环境及系统运行要求

开发环境：ModelArt Ascend Notebook

系统运行要求：Python3.7, Mindspore1.3.0

2. Technical Details

2.1 数据集简介

实验中使用的 VQA2.0 数据集主要分为 image, question 以及 annotation 三个部分，项目要求的数据集其数据量为：train:44375, validation:21435, test:21435。对于一张图片会有若干个问题，对于每一个问题会在 annotation 中有对应的解答，标明其 answer_type，以及 question_id，给出 10 个 answer，使用 answer_id:1-10 进行标号，并给出每个 answer 的 confidence。而 test 数据集没有对应的 annotation 文件。

Question 文件的 JSON 格式如下所示：

```
1 {"image_id": 131074,
2   "question": "Are the walls done in a summery color?",
3   "question_id": 131074002}
```

Annotation 文件的 JSON 格式如下所示：

```
1 {"answer_type": "other", #答案类型
2   "multiple_choice_answer": "looking in fridge", #答案
3   "answers": [...], #包含10个answer的列表
4   "image_id": 425226, #对应的img_id
5   "question_type": "what is the", #问题类型
6   "question_id": 425226003 #对应question_id}
7 {"answer": "looking in fridge",
8   "answer_confidence": "yes",
9   "answer_id": 1}
```

值得注意的是，在给出的数据集中，问题和答案，问题和图片并不是一一对应的，需要人为地去将他对齐，详细代码可查看我们的 preprocess_data.ipynb，其中对不存在于对应 image 中的 question，我们进行了移除。

为了保证 question_id 为顺序递增，而非像数据集中离散的，我们对 json 文件进行了遍历，将答案作为索引，label 序号作为键值，创建了两个索引，并创建了 trainval_questionid2label, test_questionid2label 两个字典用于建立 question_id 和 label 序号的映射，下面给出了部分建立 train_val 映射关系的伪代码：

```
1 # create labels
2 label_dict = {}
3 trainval_questionid2label = {}
```

```

4     i = 0
5     for item in trainval_json:
6         if(item not in label_dict):
7             add the item into the label_dict
8             i+=1
9         add the item into the trainval_questionid2label, with question_id as key

```

为了使得我们的模型能够顺利地用于预测和评估,我们需要根据华为云的要求创建对应的 mindrecord 文件,我们的 schema_json 文件共有三个成员,分别为问题,图片的 embedding 嵌入, 以及对应的 label:

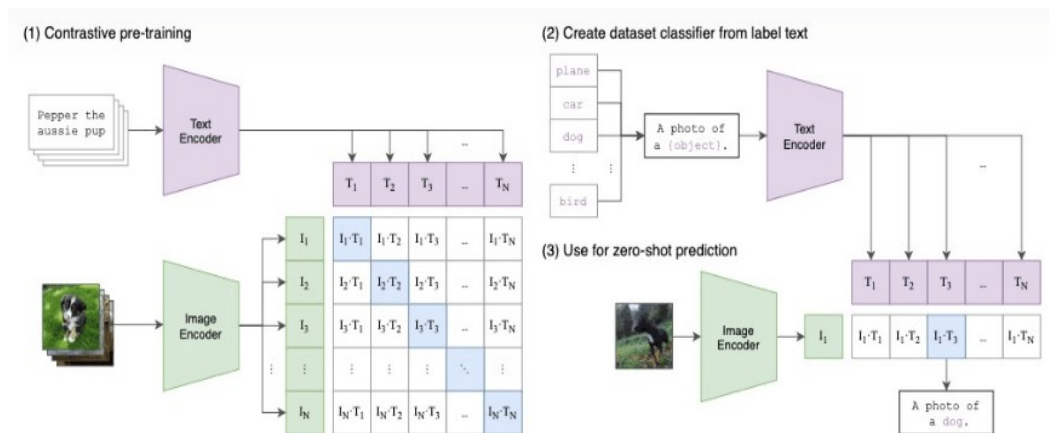
```

1. schema_json = {
2.     "ques_emb": {"type": "float32", "shape": [512]},
3.     "img_emb": {"type": "float32", "shape": [512]},
4.     "label": {"type": "int32"},
5. }

```

2.2 CLIP 简介

CLIP (Contrastive Language-Image Pre-Training, 以下简称 CLIP) 模型是 OpenAI 在 2021 年初发布的用于匹配图像和文本的预训练神经网络模型:



该模型直接使用大量的互联网数据进行预训练,在很多任务表现上达到了目前最佳表现 (SOTA)。模型采用对比学习, 预测 $N \times N$ 对图文数据, 将图片分类任务转换成匹配任务:

1. 双流, 2 个 encoder 分别处理文本和图片数据, text encoder 使用 Transformer, image encoder 用了 2 种模型, ResNet 和 Vision Transformer(ViT):

a. 5 种 ResNet: ResNet-50, ResNet-101, EfficientNet-style 的 ResNet, 包括 RN50x4, RN50x16, RN50x64;

b. 3 种 ViT: ViT-B/32, ViT-B/16, ViT-L/14;

2. encoder representation 直接线性投影到 multi-modal embedding space;

3. 计算 2 模态之间的 cosine similarity, 让 N 个匹配的图文对相似度最大, 不匹配的图文对相似度最小;

4. 对称的 cross-entropy loss;

5. 数据增强：对 resized 图片进行 random square crop。

其伪代码如下所示：

```
# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l] - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

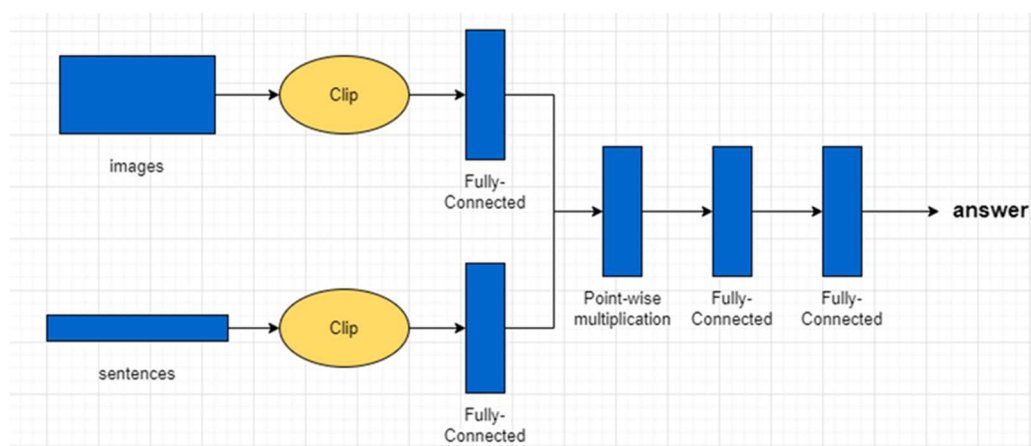
# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss = (loss_i + loss_t)/2
```

Figure 3. Numpy-like pseudocode for the core of an implementation of CLIP.

使用 clip 我们可以轻松地提取出 image feature, 并将其以及 question 等 text 转换为 embedding, 之后我们只需将这两个特征向量融合便可搭建我们的预测网络。

2.3 特征融合与预测网络(BaseLine)

在 baseline 模型中(如下图所示), 我们选用 Clip 预训练模型来生成图片的嵌入向量和问题的嵌入向量, 然后将二者通过一个全连接层进行映射, 之后将两个嵌入向量进行点积操作, 并通过两个全连接层进行输出:



其构建算法如下：

Baseline Model

1. 利用 clip 预处理模型做图像嵌入和问题句嵌入, 得到两个嵌入向量 `img_emb` 和 `ques_emb`。
2. 将 `img_emb` 和 `ques_emb` 经过 Fully-Connected 层和 relu 激活后得到两个向量: `img_out = relu(nn.Dense(img_emb))`, `ques_out = relu(nn.Dense(ques_emb))`。
3. 将 `img_out` 和 `ques_out` 做点乘操作, 并经过两层全连接层进行输出: `output = nn.Dense(nn.Dense(img_out * ques_out))`。

4. 利用 Softmax 交叉熵 (nn.SoftmaxCrossEntropyWithLogits) 计算损失。

在 VQA_baseline 中用到的函数说明如下：

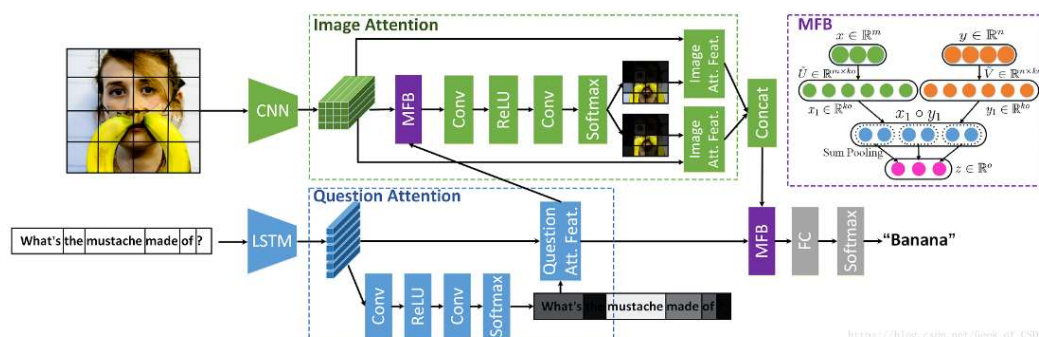
- nn.dense(): 这是 Mindspore 支持的全连接层 API。其接受的两个参数分别是输入向量的维度和输出向量的维度。
- nn.dropout(): 这是 Mindspore 支持的随机丢弃层 API。随机丢弃是一种正则化手段，该算子根据丢弃概率 $1 - \text{keep_prob} = 1 - \text{keep_prob}$ ，在训练过程中随机将一些神经元输出设置为 0，通过阻止神经元节点间的相关性来减少过拟合，在推理过程中，此层返回与输入维度相同的 Tensor。
- nn.Softmax(): 这是 Mindspore 支持的激活函数 API。它是二分类函数 mindspore.nn.Sigmoid 在多分类上的推广，目的是将多分类的结果以概率的形式展现出来。对输入 Tensor 在轴 axis 上的元素计算其指数函数值，然后归一化到 $[0, 1]$ 范围，总和为 1。
- nn.ReLU(): 这是 Mindspore 支持的修正线性单元激活函数 API。逐元素求 $\max(x, 0)$ 。特别说明，负数输出值会被修改为 0，正数输出不受影响。
- nn.SoftmaxCrossEntropyWithLogits: 这是 Mindspore 支持的损失函数 API。用于计算预测值与真实值之间的交叉熵。使用交叉熵损失函数计算出输入概率（使用 softmax 函数计算）和真实值之间的误差。对于每个实例 x_i ，i 的范围为 0 到 N-1，则可得损失为：

$$\ell(x_i, c) = -\log\left(\frac{\exp(x_i[c])}{\sum_j \exp(x_i[j])}\right) = -x_i[c] + \log\left(\sum_j \exp(x_i[j])\right)$$

其中 x_i 是一维的 Tensor，cc 为 one-hot 中等于 1 的位置。

2.4 MFB implementation

我们参考了 Multi-modal Factorized Bilinear Pooling with Co-Attention Learning for Visual Question Answering 这篇文章中的模型架构：



其中提出的 MFB(Multi-modal Factorized Bilinear Pooling)假设 $x \in R^m$, 就是假设 x 是图片的特征向量 $y \in R^n$, y 属于问题的特征向量。那么 bilinear pooling 就定义为

$$z_i = x^T W_i y$$

上面出现的 $w_i \in R^{m \times n}$ 就是一个投影矩阵, $z_i \in R$ 是 bilinear 模型的输出。上面公式里面的 W 里面包含了偏置项, 所以没有单独写出。需要通过训练 $W = [W_i, \dots, W_o] \in R^{m \times n \times o}$ 来获得相应 o 维的输出 z , 但是 bilinear pooling 有个问题就是因为引入了太多参数, 可能会导致运算量过大或过拟合。于是作者使用了本来在 uni-model data 上的矩阵分解将上面的公式变成了如下(就是将投影矩阵分解成 2 个低 rank 的矩阵):

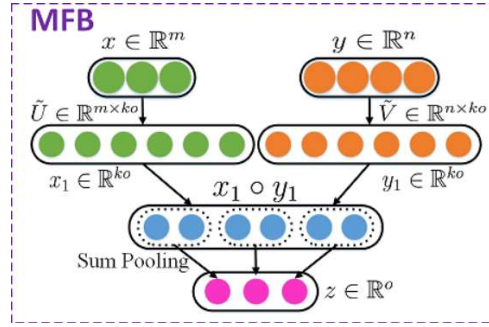
$$z_i = x^T U_i V_i^T y = \sum_{d=1}^k x^T u_d v_d^T y = 1^T (U_i^T x \circ V_i^T y)$$

K 就是 $U_i = [u_1, \dots, u_k] \in R^{m \times k}$ 和 $V_i = [v_1, \dots, v_k] \in R^{n \times k}$ 的维度, \circ 就是 Hadamard 积, 或者另一个向量在元素层面上一一对应的乘积, $1^T \in R^k$ 就是全 1 向量。

为了得到上面公式的输出向量 $z \in R^o$, y 要学习的是两个 3 阶的向量 U, V 对应的权重。可以在不影响鲁棒性的前提下将 U 和 V 通过简单的 reshape 直接变成 2 维的矩阵 $\tilde{U} \in R^{m \times ko}$ 和 $\tilde{V} \in R^{n \times ko}$, 这样上一条公式就可以重新写成:

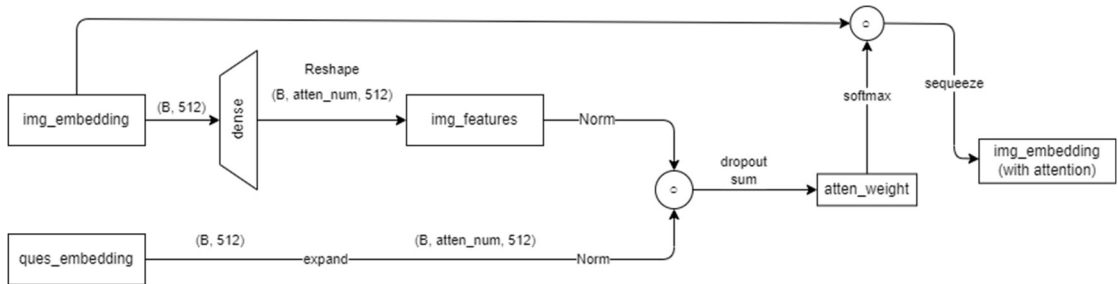
$$z = \text{SumPooling}(U^T x \circ \tilde{V}^T y, k)$$

上面这条公式里面的 $\text{SumPooling}(x, k)$ 表示用一个一维大小是 k 的没有重叠的窗口在输入的 x 上面进行 sum pooling。这个模型即被作者称为 MFB, 上面的内容可以用如下的图片表示



2.5 Attention implementation

我们还针对图像处理开发了一种方法, 将图像通过 Clip 处理后通过一个输出大小为 $\text{num} \times \text{emb_size}$ 的全连接层, 然后将其分为 num 个 embedding 向量, 与 ques_emb 做点乘操作, 模拟注意力机制, 然后将 num 个结果相加得到最终的预测值。其模型结构如下:



其构建算法如下:

Attention Model

1. 利用 clip 预处理模型做图像嵌入和问题句嵌入, 得到两个嵌入向量 img_emb 和 ques_emb 。

2. 通过一个全连接层将 `img_emb` 扩展到 `num*img_embsize` 大小,将 `ques_emb` 简单 repeat 到 `num*ques_embsize` 大小。
3. 将 `num` 个 `img_emb` 和 `num` 个 `ques_emb` 进行点乘操作, 然后经过 `softmax` 层得到输出 `out`
4. 将 `out` 的 `num` 个结果相加得到预测值。

2.6 Nonlinear Layer Implement

参考自 Tips and Tricks for Visual Question Answering: Learnings from the 2017 Challenge, 其中有一条利用 Nonlinear Layer 代替单纯的 dense 来变换向量维度, 因为某些特征可能不是线性变换就可以对齐的。非线性层的数学公式如下:

$$y_1 = \tanh(Wx + b)$$

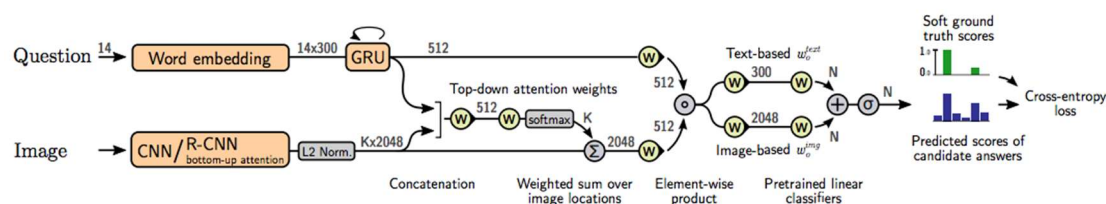
$$g = \sigma(W'x + b')$$

$$y = y_1 \circ g$$

使用中利用其替代 dense 进行向量维度的变换即可。

2.7 Faster_rcnn implementation

根据 Tips and tricks for visual question answering: Learnings from the 2017 challenge 文章的介绍:



对于 Image 部分我们可以采用 R-CNN 对图像进行特征提取, 这里使用了 trainval_resnet101_faster_rcnn_genome_36 得到了经过处理后的 train,val 数据集的 bottom-up attention 特征, tsv 文件中的数据格式如下所示:

```
1 FIELDNAMES = ['image_id', 'image_w', 'image_h', 'num_boxes', 'boxes', 'features']
```

其中 item['boxes']为对应检测框的位置信息 x,y,w,h; item['features']为对应检测框, item['num_boxes']为该张图片对应的检测框数量。

得到的特征向量如下图所示:

```
[19]: trainval_json["img_emb"][-1]
```

```
[19]: array([[0.0000000e+00, 4.9542804e+00, 9.0631904e-05, ..., 0.0000000e+00,
            0.0000000e+00, 0.0000000e+00],
            [3.9589169e+00, 0.0000000e+00, 3.0108869e-01, ..., 0.0000000e+00,
            4.8592129e+00, 4.8923711e-03],
            [1.1603643e+00, 5.8546145e-02, 2.3678014e-02, ..., 0.0000000e+00,
            5.8094821e+00, 6.1113114e+00],
            ...,
            [0.0000000e+00, 0.0000000e+00, 1.7784243e+00, ..., 0.0000000e+00,
            2.0203037e+00, 4.5792902e-01],
            [0.0000000e+00, 6.6212602e-02, 0.0000000e+00, ..., 0.0000000e+00,
            0.0000000e+00, 4.5068040e-01],
            [0.0000000e+00, 0.0000000e+00, 1.2360843e-01, ..., 0.0000000e+00,
            1.4913817e-02, 1.6268724e-01]], dtype=float32)
```

```
[25]: import numpy as np
      print(np.array(trainval_json["img_emb"]).shape)

(44506, 36, 2048)
```

但是对于这种方案，我们在尝试生成 mindrecord 文件以进行预测时遇到了一些障碍，由于我们采用的 image feature 是 36×2048 维度的，因此在生成 mindrecord 文件时出现内存不够用的情况，无论是在本地运行抑或是在华为云的 Ascend 平台上运行：

```
-----
MemoryError                                Traceback (most recent call last)
g:\trainval_36\preprocess_data.ipynb Cell 9' in <cell line: 10>()
    <a href='vscode-notebook-cell:/g%3A/trainval_36/preprocess_data.ipynb#ch0000000?line=7'>8</a> writer.add_schema(schema_json, "trainval_sc
hema")
    <a href='vscode-notebook-cell:/g%3A/trainval_36/preprocess_data.ipynb#ch0000000?line=8'>9</a> # 处理数据
--> <a href='vscode-notebook-cell:/g%3A/trainval_36/preprocess_data.ipynb#ch0000000?line=9'>10</a> writer.write_raw_data(new_json)
    <a href='vscode-notebook-cell:/g%3A/trainval_36/preprocess_data.ipynb#ch0000000?line=10'>11</a> writer.commit()
    <a href='vscode-notebook-cell:/g%3A/trainval_36/preprocess_data.ipynb#ch0000000?line=11'>12</a> print("finish", len(new_json))

File c:\Users\58382\AppData\Local\Programs\Python\Python39\lib\site-packages\mindspore\mindrecord\filewriter.py:323, in FileWriter.write_raw_da
ta(self, raw_data, parallel_writer)
    321     raise ParamTypeError('raw_data item', 'dict')
    322 self._verify_based_on_schema(raw_data)
--> 323 return self._writer.write_raw_data(raw_data, True, parallel_writer)

File c:\Users\58382\AppData\Local\Programs\Python\Python39\lib\site-packages\mindspore\mindrecord\shardwriter.py:171, in ShardWriter.write_raw_
data(self, data, validate, parallel_writer)
    169 row_blob = self._merge_blob({field: item[field] for field in self._header.blob_fields})
    170 if row_blob:
--> 171     blob_data.append(list(row_blob))
    172 # filter raw data according to schema
    173 row_raw = {field: self.convert_np_types(item[field])
    174             for field in self._header.schema.keys() - self._header.blob_fields if field in item}

MemoryError:
```

同时，如果我们需要将 image 特征的 embedding 同样存在 JSON 文件中，需要将 np.array 转换为 list 以支持可序列化，我们在 PC 端和华为云端都进行了尝试，但最后同样都会得到 Memory Error,并且由于华为云端不支持较大的文件 mox, trainval 的特征读入我们也是在 PC 端完成的，使用 wget 命令在华为云获取该文件最终也会卡死。

3. Experiment Results

Baseline model:

model	question_embdim	image_embdim	hiddendim	outdim	dropout	min_lr	max_lr	epochs	batch_size	MFB_k	MFB_o	acc
baseline	512	512	256	1024	0.2	0.001	0.01	4	256	2		0.2644
baseline	512	512	256	1024	0.2	0.005	0.03	6	256	2		0.23
baseline	512	512	256	1024	0.2	0.001	0.01	8	256	2		0.269484
baseline	512	512	256	1024	0.2	0.0001	0.003	6	256	2		0.27645
baseline	512	512	512	2048	0.2	0.0001	0.001	6	256	2		0.28765

根据同模型结果对比，可以发现，对于 baseline 模型，当 hidden_dim 较大时效果较好，而 learning_rate 介于 0.0001 到 0.001 之间 warm_up 时效果最好，准确率可以达到 28.8%左右。

MFB model:

model	question_embdim	image_embdim	hiddendim	outdim	dropout	min_lr	max_lr	epochs	batch_size	MFB_k	MFB_o	acc
MFB	512	512	256	1024	0.2	0.001	0.001	4	256	2	1024	0.23875
MFB	512	512	256	1024	0.2	0.005	0.005	4	256	2	1024	0.274
MFB	512	512	256	2048	0.2	0.005	0.005	4	256	2	1024	0.262
MFB	512	512	256	1024	0.2	0.0001	0.003	4	256	4	1024	0.251129
MFB	512	512	256	1024	0.1	0.0001	0.003	4	256	2	1024	0.24694

根据同模型结果对比，可以发现，对于 MFB 模型，当 learning_rate 为 0.005 时效果较好，准确率 accuracy 可以达到 27.4%。

Attention model:

model	question_embdim	image_embdim	hiddendim	outdim	dropout	min_lr	max_lr	epochs	batch_size	MFB_k	atten_num	acc
attn	512	512	256	1024	0.2	0.0001	0.001	2	256	2	4	0.23838
attn	512	512	256	1024	0.2	0.0001	0.001	4	128	2	16	0.25697
attn	512	512	256	1024	0.2	0.0001	0.003	4	256	2	8	0.2532
attn	512	512	256	1024	0.2	0.0001	0.003	4	256	2	32	0.261813
attn	512	512	256	2048	0.2	0.001	0.001	4	256	2	32	0.28214
attn	512	512	512	2048	0.2	0.0001	0.001	4	256	2	36	0.264543
attn	512	512	512	2048	0.2	0.0001	0.003	4	256	2	8	0.25174
attn	512	512	512	2048	0.2	0.0001	0.003	4	256	2	16	0.263789
attn	512	512	512	2048	0.2	0.0001	0.003	4	256	2	32	0.264072
attn	512	512	256	2048	0.2	0.0001	0.005	4	256	2	32	0.241152

根据同模型结果对比，可以发现，对于 Attention 模型，当 output_dim 较大时效果较好，而 learning_rate 介于 0.0001 到 0.001 之间 warm_up 时效果最好，同时，模型效果也随着 attention_num 在一定程度内的增大而变好，准确率可以达到 28.2%左右。

根据以上结果，可以看出参数 learning_rate 对结果有一定影响，同时对于 attention 模型，attention_num 对结果的影响也较大。三种模型中，baseline 模型和 Attention 模型效果相差不多，而 MFB 相对于 baseline 模型效果稍差一点，但三种模型的准确率都在相近的范围内，集中在 28%左右。

References:

- [1] VQA: Visual Question Answering (ICCV 2015).
- [2] Tips and tricks for visual question answering: Learnings from the 2017 challenge (CVPR 2018).
- [3] Zhou Yu, Jun Yu, Jianping Fan, Dacheng Tao, 2017. Multi-modal Factorized Bilinear Pooling with Co-Attention Learning for Visual Question Answering.
- [4] Tips and Tricks for Visual Question Answering: Learnings from the 2017 Challenge 论文解读:
<https://blog.csdn.net/u014248127/article/details/86091518?spm=1001.2014.3001.5502>
- [5] Multi-modal Factorized Bilinear Pooling with Co-Attention Learning for Visual Question Answering 笔记: https://blog.csdn.net/Geek_of_CSDN/article/details/81328487