深度学习在视觉问答任务中的探索与实现

项目背景

问题描述

VQA视觉问答，是计算机视觉领域的一个分支任务，涉及回答关于图像的问题。机器读取并理解图像的内容，并用自然语言去答案图片相关的问题。视觉问答有很多实现方式，但最常见的方式是使用深度学习模型，例如卷积神经网络（CNN）和循环神经网络（RNN），分别从图像和问题中提取特征。然后将这些特征组合并送入分类器中以预测答案。

VQA问题的难点在于结合了图像和文本的理解，需要系统理解图像内容以及问题的含义，并生成正确的答案。这对于计算机而言是一个具有挑战性的任务，因为它要求系统具备深入的视觉理解和语义推理能力。

相关工作

视觉问答任务在近年来受到了广泛的关注，并且已经有了一些重要的研究成果。其中，深度学习方法在视觉问答任务中取得了显著的进展。

一些常用的深度学习模型包括循环神经网络（Recurrent Neural Networks，RNN）和卷积神经网络（Convolutional Neural Networks，CNN）。这些模型通过学习图像和问题之间的复杂关系，能够对问题进行理解，并生成准确的答案。

此外，还有一些改进的模型结构，如注意力机制（Attention Mechanism）和外部知识的引入，进一步提高了VQA模型的性能和效果。

项目意义

视觉问答任务在实际应用中具有广泛的应用前景。它可以用于构建智能问答系统、辅助图像搜索和图像标注等任务。通过训练一个准确和

鲁棒的VQA模型，我们可以使计算机更好地理解和回答关于图像的问题，从而提高智能系统在视觉理解和自动问答方面的能力。

在本项目中，我们将探索视觉问答任务，并实现一个具有较高性能的VQA模型。通过该项目的实践和研究，我们将深入理解视觉问答任务的挑战和技术解决方案。

项目目标

- 本项目旨在通过训练一个神经网络模型，实现视觉问答任务。通过对大规模的图像和问题数据集进行训练，我们希望能够让模型学会从图像和问题中提取特征，并生成正确的答案。

- 本次大作业依照Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering (arxiv.org)模型，参考hengyuan-hu/bottom-up-attention-vqa代码并在Pytorch1.4版本上实现了该模型。我们通过构建一种bottom up的视觉注意力，与top down的上下文注意力，来实现对图像和问题特征的集中，以达到更好的回答效果。

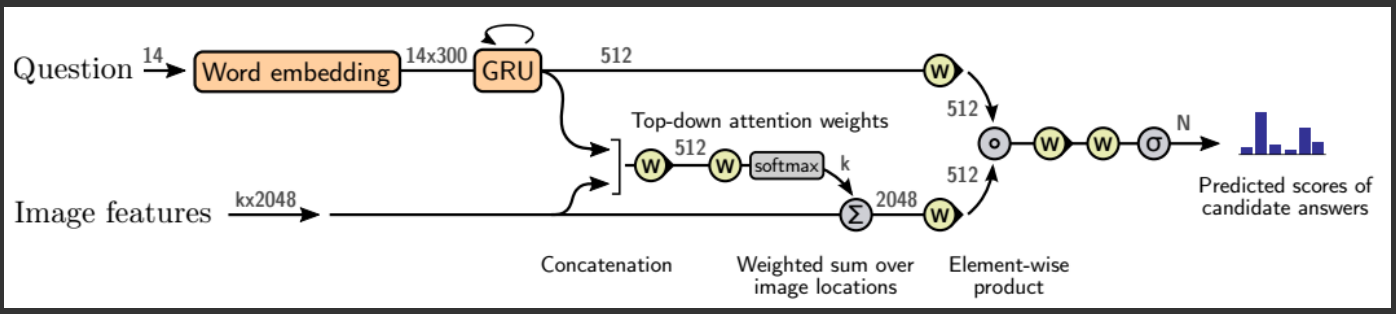
项目环境

- ModelArts Notebook 环境
- Pytorch1.4
- tqdm
- h5py

技术细节

VQA模型总览

以往的模型中，都是通过Top-Down Attention对图像的问题进行处理，并将这个注意力用在提取好的图片特征上面，最后使用分类器来预测答案。模型如下：



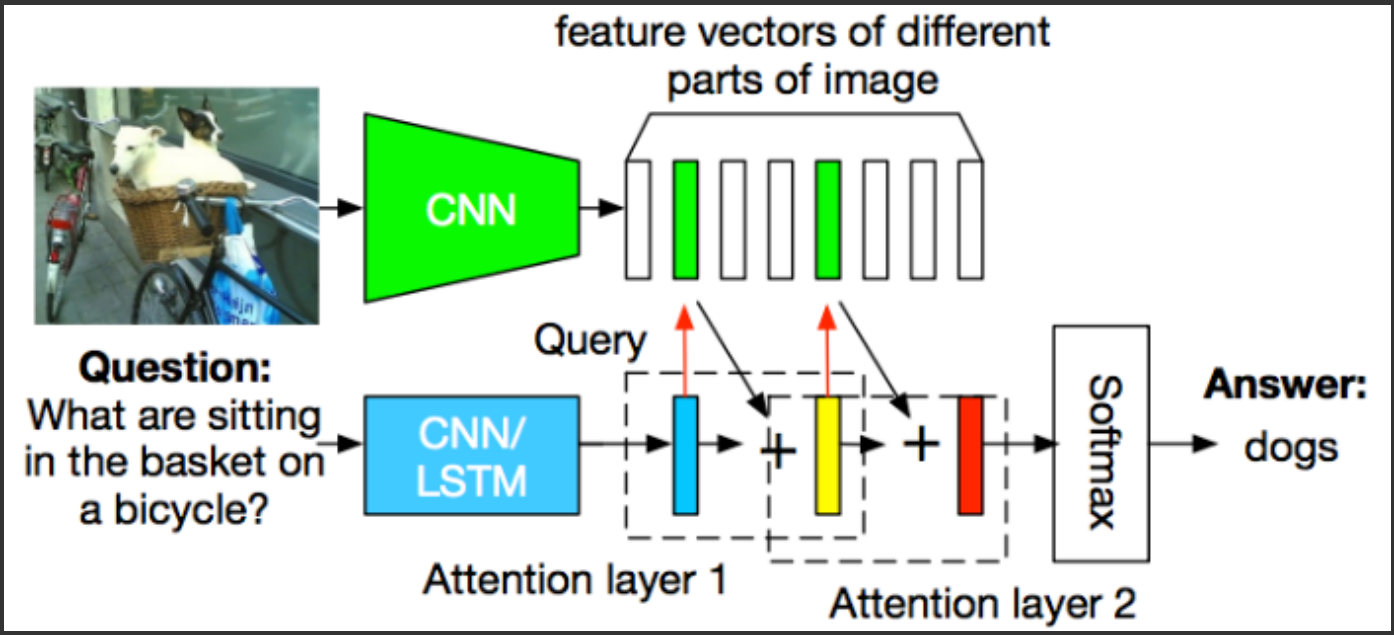
而我们抛弃了原来的CNN提取图像特征，使用Bottom-up attention对图片提取了特征，其他模型与以往的相似，整个模型构建伪代码如下：

```
class baseline_model(nn.cell):
  def forward():
    image_feature = bottom-upNet(image)
    Word = Wordembeding(question)
    question = LSTM/GRU(word)
    attention = Top-down(image_feature, question)
    answer = classifier(attention*image_feature,question)
    return answer
```

接下来简述下各个模块的原理。

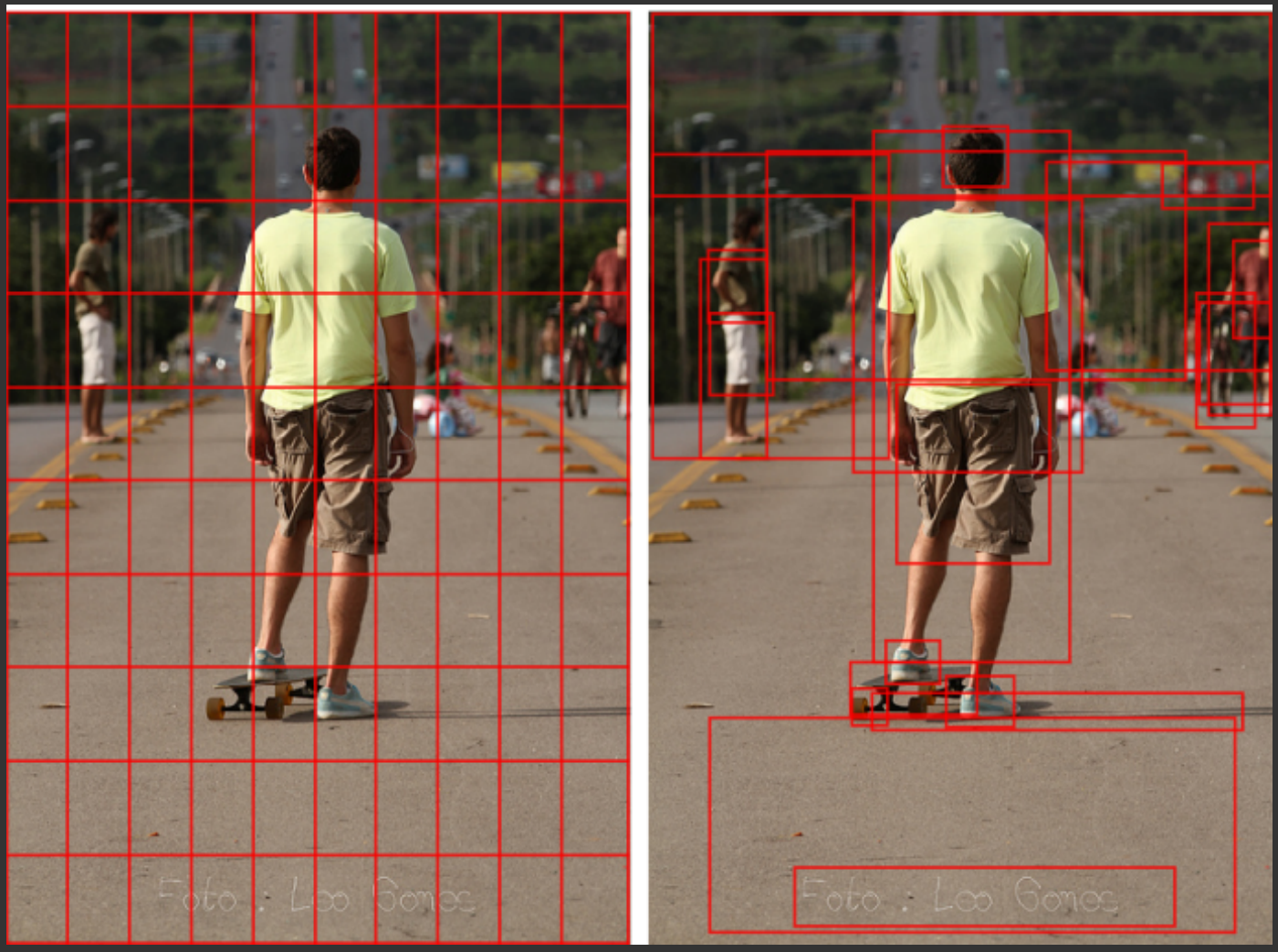
Bottom-up Attention

在以往的VQA模型中，仅仅使用Top-Down Attention对图像的问题进行处理，即取问题作为输入，建模注意力分布，并将其作用在CNN提取图像的特征上中。

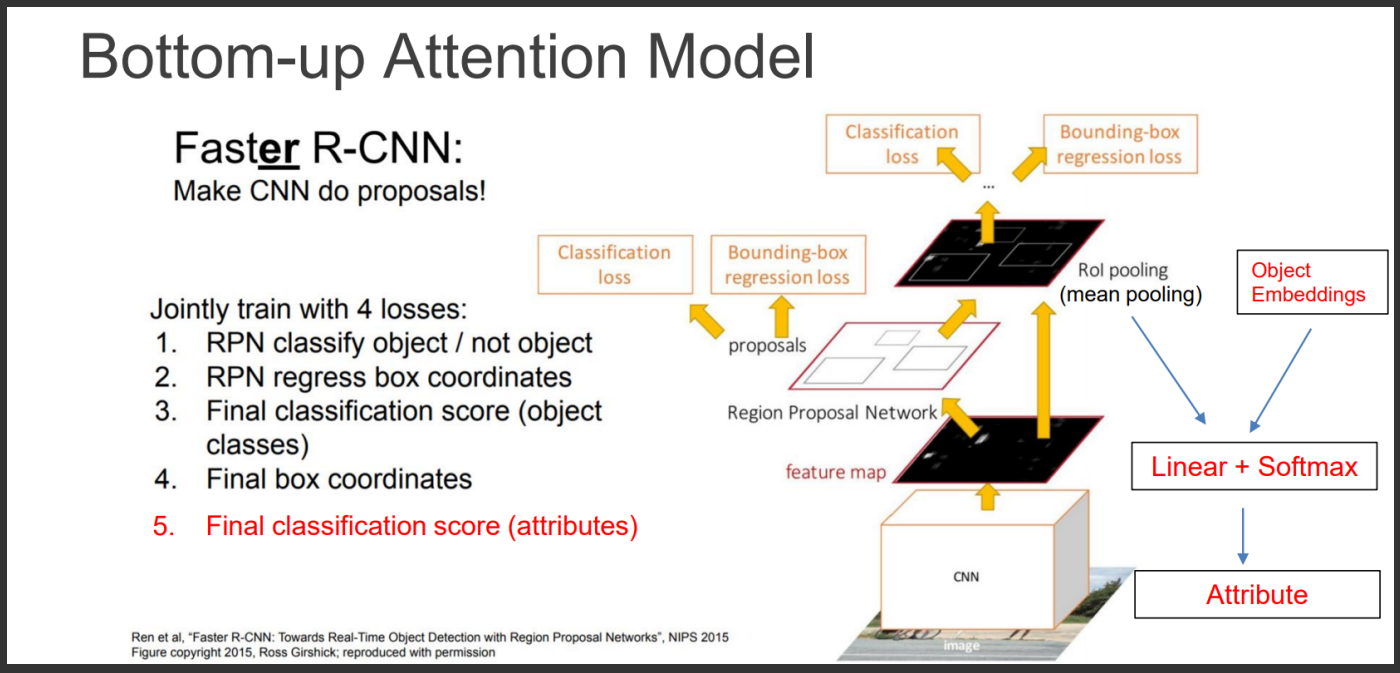


而这种方式处理其实忽略了CNN是怎样对图片进行特征提取的，在这种方式下，CNN提取的特征可能并不能很好地适应问题。下列左图就是Top-down Attention model中CNN提取的特征，可以看到这种方式下产生的输入都是一些与感受野大小、形状保持一致的小方格，与实际的图像并没有关联。

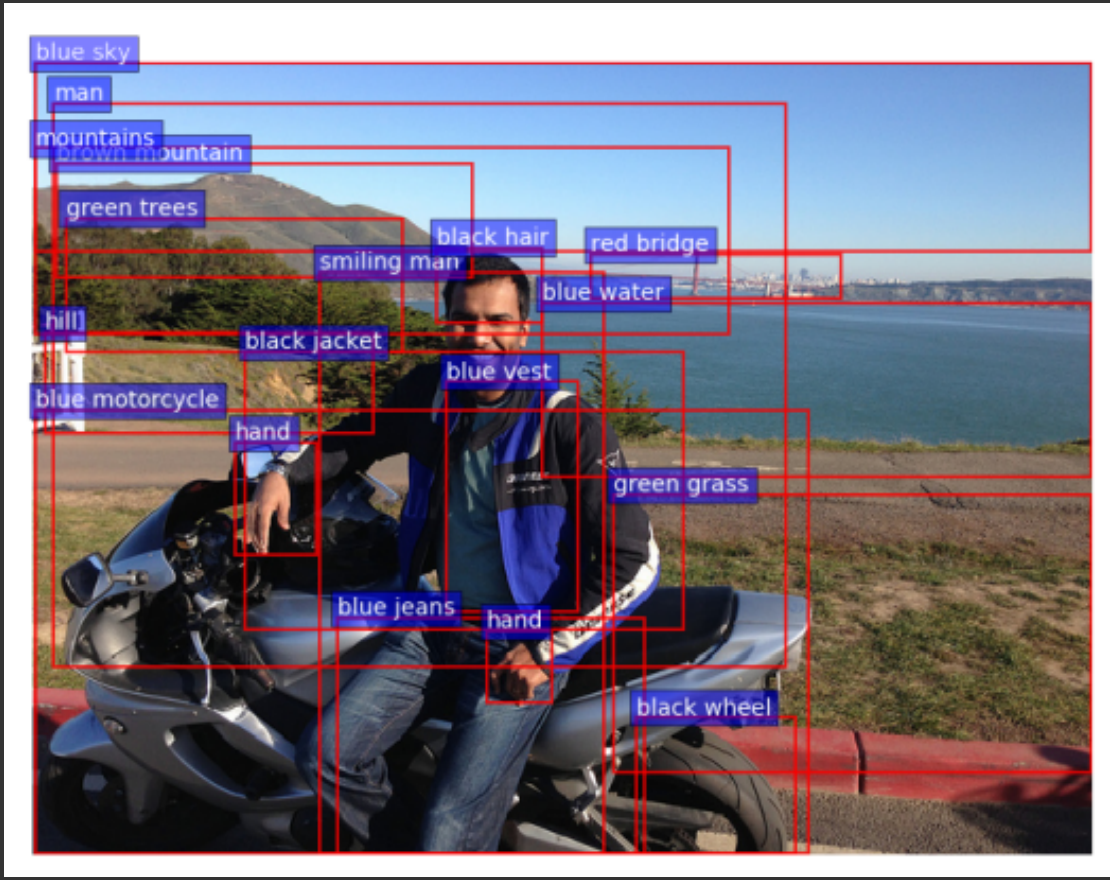
而Bottom-up attention则是通过针对图像的底层特征和区域来计算注意力，并提取显著图像的区域。具体地说，Bottom-up attention会提出图像中的候选区域，并为每个候选区域生成对应的特征向量，这些特征向量表示每个区域的视觉信息。如下列右图所示：



在这里我们使用Faster RCNN来实现对图片自下而上的注意力关注，并用边界框划定不同的特征空间，并对其进行分类。其模型如下图，其输出为一个边界框和分类结果：



经过Faster RCNN处理过后的图片如下：



Question embedding

我们首先对问题进行词向量的转换，转换的词向量W后送入到第一层LSTM/GRU中，得到处理好的向量W。

Top-down attention weights

在这里我们将在Question embedding中处理得到的W和图片特征V送入到模型中，首先将图像特征和图片特征投影到隐藏层维度上得到`v_proj`和`q_proj`。接着将`v_proj`和`q_proj`相乘，并送入到LSTM层中，联系上下文。最后将LSTM层处理后的结果使用Dorpout层并送入到线性层中，通过线性层得到注意力权重logits，并使用`Softmax`函数将其转换为概率分布。输出最后注意力选择的图片特征。模型图如下：

概率预测

参考Tips and Tricks for Visual Question Answering: Learnings from the 2017 Challenge (arxiv.org)

这篇文章中提到的一些在细节上提升VQA效果的方案：

- Using gated tanh activations in all non-linear layers
- Using soft scores as ground truth targets

对此，我们在预测时采用了回归预测，对答案进行概率预测，而不是传统的进行分类。这就需要我们在对数据集处理前需要预处理，计算每个问题下答案对应的分数，基本思路如下：对于答案数据集中的每个答案条目，统计每个答案出现的次数，并将其存储在answer_count字典中。接下来，我们遍历answer_count中的每个答案，如果该答案在ans2label字典中存在对应的标签，则将该标签添加到labels列表中，并计算该答案的得分，并将得分添加到scores列表中。

通过这样的处理，我们可以将任务视为候选答案分数的回归问题，而不是传统的分类问题。

实验过程

在实验中，由于时间原因，我们直接使用了peteanderson80/bottom-up-attention: Bottom-up attention model for image captioning and VQA, based on Faster R-CNN and Visual Genome (github.com)中提供的Faster RCNN预训练模型trainval_36来作为图片特征。在这基础之上，我们自己设计了Question Embedding、Top-down Attention的模型和分类器，实现了VQA的模型。

处理数据集

我们数据集中的一项应该包括问题，图片id，图片特征，答案这四部分。预训练模型trainval_36中包含了对图片id，以及预训练的图片特征和图片的位置。我们需要做的就是根据原始数据集中问题对应的图片id，加载对应的图片特征，并加入问题和答案这些数据。

Dataset的代码功能如下所示：

```
import torch
import h5py
import json
from torch.utils.data import Dataset

class VQADataset(Dataset):
    def __init__(self, split, image_features_path, dataroot):
        self.split = split
        self.image_features_path = image_features_path
        self.dataroot = dataroot
```

```
        # 加载trainval_label2ans.json和dictionary.json文件
        with open('trainval_label2ans.json', 'r') as f:
            self.label2ans = json.load(f)
        with open('dictionary.json', 'r') as f:
            self.dictionary = json.load(f)

        self.entries = []
        self._load_entries()
        self._create_img_id_to_idx()

    def _create_img_id_to_idx(self):
        # 创建从COCO图像ID到h5文件中对应索引的映射
        self.img_id_to_idx = {}
        # ...

    def _load_entries(self):
        # 加载数据集的条目
        # 加载问题数据
        # 根据问题ID对问题数据进行排序
        # 如果数据集不是测试集，加载答案数据并根据问题ID排序
        # 遍历问题和答案数据，创建条目并添加到entries列表中
        # 如果数据集是测试集，创建特殊条目
        # 打印entries列表的长度
        # ...

    def load_image(self, idx):
        # 加载图像特征
        # 检查是否已经加载了h5文件，如果没有则加载
        # 根据索引从h5文件中获取特征和空间信息
        # 转换为PyTorch张量并返回
        # ...

    def encode_question(self, question):
        # 将问题编码为张量
        # 使用dictionary对象对问题进行分词
        # 截断到指定的最大长度
        # 如果长度小于最大长度，在句子前面填充填充符号
        # 转换为长整型张量并返回
        # ...

    def encode_answer(self, answer):
        # 将答案编码为张量
        # 如果数据集是测试集，返回空张量
        # 否则，创建全零张量作为目标
        # 根据答案中的标签和分数对目标进行填充
        # 返回目标张量
        # ...

    def __getitem__(self, idx):
        # 获取指定索引的数据
        # 获取对应索引的条目
```

```
        # 调用load_image方法加载图像特征
        # 获取问题、问题ID和答案
        # 返回特征、空间信息、问题、答案和问题ID
        # ...

    def __len__(self):
        # 返回数据集的长度，即条目的数量
        return len(self.entries)
```

搭建网络

根据前文提到的神经网络原理，我们搭建了我们的网络模型。值得提到的一点是，由于LSTM是时间序列模型，时间t时刻要依赖时间 t-1时刻信息，不能并行执行，且LSTM相比GRU中的结构更为复杂，训练时间更长。因此我们在Question embedding中使用的是GRU，Top-down attention中用的是LSTM。

训练模型

最后则是进行我们的训练过程，我们通过求解每次迭代中的梯度，并用Adamax优化器对其进行优化，进行我们的训练。同时，我们还显式的打印每一次迭代中的模型准确率和损失。

```
# 创建数据加载器
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)

# 训练模型
for epoch in range(num_epochs):
    for batch in train_loader:
        # 前向传播和计算损失
        features, questions, targets = batch
        outputs = model(features, questions)
        loss = loss_fn(outputs, targets)

        # 反向传播和优化
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

经过测试，在数据集的batchsize=512的情况下一轮训练时间大概约为5分钟左右，我们设置epoch = 30，通过3个小时的训练，成功得到了我们的训练模型。

结果分析

在测试过程中，我们使用训练好的模型对测试集中的样本进行预测，并计算模型的准确率和其他评估指标。我们将模型的预测结果与真实答案进行比较，以评估模型的性能和泛化能力。这里我们定义了两个函数来计算损失以及准确率，代码如下：

```
def binary_cross_entropy_with_logits(input, target, mean=False):
    """
    Function that measures Binary Cross Entropy between target and output logits:
    """
    if not target.is_same_size(input):
        raise ValueError("Target size ({}) must be the same as input size
({})".format(target.size(), input.size()))
    max_val = (-input).clamp(min=0)
    loss = input - input * target + max_val + ((-max_val).exp() + (-input -
max_val).exp()).log()
    loss = loss.sum(dim=1)
    return loss.mean() if mean else loss

# 根据预测的概率计算得分
def compute_score_with_logits(logits, labels):
    # 选出最大的概率的答案
    logits = torch.max(logits, 1)[1].data
    # 映射到label维度上的one-hot向量
    one_hots = torch.zeros_like(labels).cuda()
    one_hots.scatter_(1, logits.view(-1, 1), 1)
    # 计算
    scores = (one_hots * labels)
    return scores.sum(dim=1)
```

测试结果如下所示：

项目总结

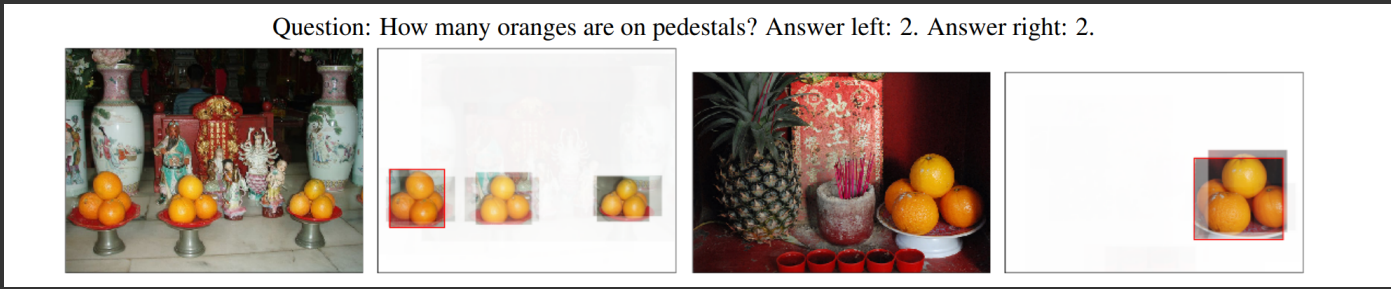
本项目旨在探索视觉问答（Visual Question Answering, VQA）任务，并设计并实现一个基于深度学习的VQA模型。通过对大规模的VQA数据集进行训练和评估，我们对模型的性能和效果进行了深入的分析和研究。

优越性

结合自下而上和自上而下的注意力机制的方法的优越性在于能够对图像进行更精细的分析，实现对图像和语言的理解。通过在图像的显著区域和对象层面计算注意力，并进行多步推理。在视觉问答任务中，该方法的准确率在63%左右，相比与其他的方法，该方法有较为优秀的效果 。

局限性

自下而上的注意力机制十分依赖对显著图像区域提取的好坏，如下面这个例子：



当Faster RCNN对于橘子图片的特征提取时没有细致的区分每个橘子，所以在回答时导致错误，这也提示我们可以通过改进提取方法来提高模型的表现。

总体而言，通过这个项目，我们深入理解了视觉问答任务的挑战和技术细节，并成功实现了一个基于深度学习的VQA模型。我们的实验结果表明，我们的模型在VQA任务上取得了有竞争力的性能。然而，我们也认识到VQA任务仍然存在许多挑战和限制，如对于复杂问题的处理和跨模态的推理能力等。因此，未来的研究可以继续探索和改进VQA模型的性能和泛化能力。