Evaluating Compositionality of Vision and Language Models

Gregory M. McCord

Advisor: Professor Olga Russakovsky

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Computer Science

Princeton University

June 2020

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

_____

Gregory M. McCord

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Gregory M. McCord

Abstract

While research in computer vision has made many breakthroughs over the past two decades, probing even the most state-of-the-art models reveals glaring shortcomings. For example, the same Visual Question Answering (VQA) model that correctly answers the question "What is the largest measurement on the cup?" might incorrectly respond "yes" to the question "Is there a plate under the elephant?" when an elephant is not even in the picture. Similarly, a top-performing captioning model might incorrectly describe a blue shirt as white or a red flag as black.

To address some of these shortcomings, we first propose a vision and language evaluation framework that probes models trained on different tasks to better understand the performance of the models on different detailed concepts. The concepts we analyze are identifying the colors of objects, counting the numbers of objects, and identifying the object types in a scene. We evaluate along each of these dimensions by using distractors (related but incorrect statements) to test along individual dimensions. Further, we also train the models with different levels of supervision along the tested dimension to demonstrate that performance on these tests improves solely with the dimension tested while not inhibiting the models' performance on other dimensions. With this scheme, we quantify the types of queries the models accurately answer and those on which they fail. We develop baselines for our framework using the standard Show and Tell captioning model, Bottom-Up Top-Down captioning model, and Pythia VQA model.

Next, we address two reasons for the surprising inconsistencies observed in current VQA models. Namely, responding solely based on the language of the question and putting inappropriate attention on different objects referenced in the question can lead to this undesirable behavior. We generate complex binary questions for VQA models using the scene graphs from the GQA dataset to evaluate how effectively the models are able to align the questions with the corresponding images. Additionally,

we test a verification scheme for VQA models to correct for their internal consistency by creating a set of entailed questions and correcting for non-verified answers. On the state-of-the-art Pythia VQA model, our verification improves performance on VQA accuracy from 64.95% to 65.99%. Finally, we take a look at broader extensions of this research into financial applications.

Acknowledgements

I would like to first express my sincere gratitude to my advisor, Professor Olga Russakovsky, for her support, guidance, and encouragement throughout the year. Through her, I have learned not only how to do research effectively but also how to critically evaluate the questions both I and others are asking towards the problems we face. I would like to thank my other professors as well, in particular Professor Ahmadi, Professor Petras, and Professor Boumal, whose teaching and advice remind me of the joy of learning. I would also like to thank Professor Narasimhan, Zeyu Wang, and Zhiwei Deng for their help and mentoring throughout the EVAL project. I would also like to thank the other members of the VisualAI Lab for constantly introducing me to new ideas and challenging me to approach new problems. I would additionally like to thank Utsav Popat for offering advice on the finance portion of my thesis.

I would like to thank my friends and rugby teammates for all of their support over the last few years. In particular, I would like to thank Owen Tedford, Alex Rogers, and Hugues Martin Dit Neuville for their friendship on and off the field. I would also like to thank our PURFC coaches Richard Lopacki, Marshall Boyd, Dylan McGagh, Alan Zondagh, Taylor Hurff, and Dan Ostberg for their professional mentorship and personal development - they have broken me down and built me back up into the man I am today. I'd also like to thank the never-ending support of the close friends I've made at Cannon including my fellow Delawarean, Marshall Saunders, and track friends Heide Baron and Hadley Wilhoite. I would also like to thank Athletes in Action and Reverend Busch for their help in continuing to grow in my faith. I would like to thank the first roommate I ever had, Dovid Braverman, and zee group-mate, Grace Guan, for their support since our first days on campus.

I would like to thank Archmere Academy as a whole and in particular, Father McLaughlin, Coach Booker, Coach Ambroji, Coach Niumataiwalu, and the lifelong friends made there. They have been instrumental in the development and progression

of my academic, personal, and faith journeys. I would also like to thank St. Mary's of the Assumption Church and Father Dillingham for his constant support in my faith growth.

I would like to thank Emily Hilliard-Arce and George Lin for being the first friends I grew close to on campus. I would like to thank my roommate, Jose Yanez, for constantly encouraging me to expand my horizons, try new things, and continue to grow as a person. I would like to thank Ashley Willingham for her insightful comments on my drafts and her support in all areas of my life, helping me to continue to grow and learn everyday.

Finally, I would like to thank my brother, Chris, and parents, Kathy and Mike, for supporting me every single day, good and bad. They have been there for me to rely on and learn from since day one and have always driven me forward, supporting me in my faith in God and academic and athletic pursuits. Without their love, support, and guidance, I would not have grown and developed as I have.

To my brother, mother, and father, for helping me become who I am today

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background Information

In order to understand the current motivations for computer vision, the dependence on large datasets and powerful Graphics Processing Units (GPUs), and growth in variety of tasks, we must first understand how the current research landscape came to be. After providing a brief history, we present a swath of popular tasks currently being researched in computer vision and natural language processing. We will conclude with an overview of tasks that sit at the intersection of both vision and language, including the two principle components of my study, VQA and captioning.

1.1.1 Brief History of Computer Vision

Computer Vision has developed significantly over the past few decades. Algorithms designed to process and interpret images have been around since the 1970s. However, in the early days of these algorithms, researchers lacked both the data and computing power to leverage these revolutionary models. It was not until the early 2000s that significant steps could be taken in computer vision that resulted from access both to expansive datasets and sufficiently powerful computers to process the data. In 2003,

Fei-Fei Li created the Caltech 101 dataset consisting of 101 image categories and 9,146 images [15]. Soon after, the first large-scale datasets were deployed, including PASCAL [14] and ImageNet [11]. For comparison of scale, ImageNet contained over 1.2 million images with 1,000 object classes. While there had been challenges associated with datasets before (including one for PASCAL), the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) marked a turning point in computer vision [46].

ILSVRC began in 2010, and in the first two years, top-5 error rate (percentage of images where the true label was not in the top 5 predictions) exceeded 25%. Additionally, the dominant models in these competitions used recognition pipelines featuring Scale Invariant Feature Transform (SIFT) [36] and Spatial Pyramid Matching (SPM) [29] for object recognition while preserving spatial information. However, ILSVRC 2012 saw the advent of AlexNet, a deep Convolutional Neural Network (CNN), which achieved a top-5 error rate of 15.3% [28]. While this groundbreaking discovery occurred in 2012, the concept of neural networks (and in particular CNNs) had been around since at least the 1980s when Fukushima proposed convolutional and downsampling layers in a network architecture [17]. Further, LeCun, et al. demonstrated the efficacy of such networks with the proposed LeNet-5 architecture for optical character recognition (OCR) [30]. It is not that CNNs hadn't been attempted on computer vision tasks in the 21st century. Rather, initial results on Caltech 101 indicated that CNNs were not suited for the task of image recognition due to their poor performance on that task. This discrepancy in performance of CNNs on the different datasets demonstrated the need for large amounts of data in order to effectively train CNNs and achieve high results. A comprehensive history of computer vision, specifically on the developments in the 20th century that are less relevant to this work, may be found in Szeliski's introduction [48].

1.1.2 Survey of Current Computer Vision Tasks

Current computer vision research divides (with a degree of overlap) among several areas. Traditional computer vision is often thought of as image processing. The first influential task (potentially the most influential in all of computer vision) was image classification and object recognition that spawned the ImageNet dataset and competition mentioned prior, which presents images with the focus on certain objects and requires the model to correctly identify the image. A task that naturally followed this was object detection - identifying multiple objects of different types in the same image. The MSCOCO dataset [34] was one of the first large datasets to present complete human-drawn annotations for different object classes. More advanced tasks often rely on object detection algorithms such as Faster R-CNN [45] or You Only Look Once (YOLO) [44] to feed object features into the models. From this perspective, image processing can often be thought of as one of the building blocks of other popular computer vision tasks.

Video processing is highly related to image processing except that it includes the temporal element in order to track objects over time. Beyond basic classification of objects seen in videos, works such as Karpathy et al. [25] extend the task to categorizing actions as well. Today, video processing is commonly used in self-driving cars, although much of this research is proprietary.

Bias is perhaps the most independent (but also the most encountered) topic in computer vision since it appears both at both the dataset and model levels. Datasets can present bias in all sorts of ways ranging from language priors [18] to gender and racial bias [5]. These biases introduced in datasets then become apparent in models that are trained on these datasets. This has led to an effort to balance and correct biases in datasets while enforcing constraints on models to make them more fair [6].

Finally, there are vision application tasks, which heavily depend on simpler, more contained tasks such as video processing or object detection. Self-driving cars,

robotics, and simultaneous localization and mapping (SLAM) are all hot topics of research currently. However, many higher level tasks also rely on computer vision at the intersection of language.

1.1.3 Survey of Current Language Tasks

Current Natural Language Processing (NLP) research broadly falls into two major categories. The first major category is Natural Language Generation, where models train on a text corpus in order to generate new sentences. Older models would use Markov Chains in order to predict the probability of the next word given the previous k terms. More recently, the trend has moved to using recurrent neural networks (RNNs) with Long-Short Term Memory (LSTMs) [20]. These often serve as the basis for popular artificial intelligence language interfaces such as chatbots.

The second major category is Natural Language Understanding. Sentiment analysis is one of the most popular tasks under this umbrella where a model is fed a sentence and is tasked with responding with a sentiment score describing the emotions of the speaker or writer. Question entailment is a second popular task that deals with identifying entailed properties of a body of text. In Chapter 5 of our work, we utilize a rudimentary entailment scheme in the analysis of a model. Question answering is a third popular task that involves a model responding to a question. While this area has been extensively studied, it is a difficult research area due to the large amounts of unlabeled text available but relative dearth of corresponding labels. However, a 2018 hallmark paper from OpenAI aimed to combat this by proposing a Generative Pre-Training (GPT) model that helped to dramatically improve performance of such language understanding models using unlabeled text corpus [42]. Another hallmark paper published in 2018 detailed a second pre-training scheme put forth by Google AI, which performs Bidirectional Encoder Representations from Transformers (BERT) [13]. BERT pre-trains bidirectional encoders by taking into account context

on both the left and right sides of each word. A recent 2019 pre-training model called RoBERTa [35] builds off of and significantly improves over BERT by optimizing the model's design and fine-tuning the performance of the model.

Of the categories of language research mentioned, natural language understanding, is of particular interest to our research as understanding language is incredibly important when interacting with visual data. Herein, we see the rise of vision and language models. For instance, the pre-training efforts led by BERT and GPT gave rise to vision and language pre-training papers such as the 2019 pre-training vision and language BERT (ViLBERT) [37] paper. This model extends the BERT architecture by applying it to a multi-modal stream that processes vision and language input through co-attentional layers.

An additional, more fledgling area of NLP research is in evaluation frameworks for language models. The Benchmark of Linguistic Minimal Pairs (BLiMP) [55] proposes a new dataset for evaluating language models on the basis of minimal pairs - a series of similar questions that vary slightly and help to identify which particular syntaxes trouble individual language models. We utilize a similar methodology in our analysis in Chapter 4. On the other hand, oLMpics [49] devises an evaluation scheme to specifically understand what language model pre-training captures, the results of which will continue to help guide the development and training of pre-trained language models. These results can further help improve performance in vision and language models as well.

1.1.4 Survey of Vision and Language

Unified vision and language models attempt a difficult task - not only must a model semantically understand an image and, independently, the important parts of a sentence, it must also understand how the attentions of the sentence relate to the image. While the unification of vision and language can be accomplished in a multitude of

ways, the major two avenues examined in this paper are captioning and visual question answering (VQA). VQA models take as input an image and a question, and the models output an answer to the question. On the other side of the task, captioning takes an image and returns a textual summary of the image. For VQA, some basic questions such as "Is there a table?" reduce to object detection. These questions could be answered simply (and accurately) by running an object detector and returning 'True' if the object is found in the image and 'False' otherwise. Similarly, a captioning model attempting to say simple and correct things might list the objects in the image. However, harder questions and more complex captioning models require more intelligent reasoning in order to address interactions between objects in the image. For example, one might ask "What object has the same color as the car?" to a VQA model while a captioning model could summarize the same image by saying "The woman drives the red car past the red fire hydrant." In this paper, we focus primarily on analyzing and applying our research to captioning and VQA models. However, we hope to extend these results to related vision and language tasks in future work.

Vision and language models have many practical applications in real life that motivate their study. Captioning models can help in the automated categorization of images by automatically generating easily filtered and searched text to accompany each image. Similarly, VQA models can help vision impaired users navigate the internet by answering questions regarding images on the web page.

1.1.5 Motivation

While current models outperform expectations from the last decade on loss rates on a variety of tasks, it takes a minimal amount of probing into a state-of-the-art model to realize they lack the understanding that current evaluation metrics might suggest. For instance, even common captioning metrics have been shown to correlate poorly

with human responses [8]. Further, the same Visual Question Answering (VQA) model that correctly answers the question "What is the largest measurement on the cup?" might incorrectly respond "yes" to the question "Is there a plate under the elephant?" when an elephant is not even in the picture. Similarly, a top-performing captioning model might incorrectly describe a blue shirt as white or a red flag as black. Additionally, there is currently no unified understanding of how vision and language models trained on different tasks compare to one another. While captioning models capture a high level view of the image, it might be that VQA models possess a more fine-grained understanding.

Therefore, we aim to evaluate vision and language models' ability to compose information and determine not only how well they do, but also in what particular ways they do well. There are several dimensions along which we probe the models in order to test different qualities. Among the tested dimensions are unary questions (about a specific aspect of the image), relational questions (relationships between different objects), and counting questions (how many of a certain object there are). Furthermore, we introduce a set of compositional questions that test the language part of the model by inverting object order and introducing false information. For example, consider an image of a chair with nothing else in the image. For some models, the questions "Are there a chair and desk?" and "Are there a desk and chair?" might return different answers even though a human cognitively understands that these are equivalent questions. Relatedly, some models perform poorly when there are multiple objects mentioned in the question, only a subset of which are actually in the image. Current captioning evaluation is performed using a variety of metrics, but none of them allow for a breakdown of the score into different categories, meaning that there is currently no way to understand what aspects of a captioning model perform well and which underperform. VQA evaluation on the other hand allows for a breakdown of questions by category type where the question categories fall more into the type

of question it is (e.g. "Is there X?" or "What is Y?"). This scheme provides greater insight into the model's performance on more complex questions and intentionally probes a broader class of aspects than the standard VQA categories cover. See Section 1.3.2 for a summary of current evaluation metrics.

I am currently working with Zeyu Wang, Professor Olga Russakovsky, Professor Karthik Narasimhan, and Zhiwei Deng on a publication (targeted for NeurIPS 2020) for work relating to this. Current evaluation metrics used for captioning include BLEU [40], CIDEr [53], METEOR [12], ROUGE [32], and SPICE [1], but these metrics don't do a great job of evaluating the compositionality of the models. Similar to our work in VQA, the paper will aim to create a new evaluation scheme for captioning that helps evaluate the compositionality of models.

1.2 Contributions and Structure

In this work, we detail our contributions made in two primary areas. First, we describe our proposed vision and language evaluation framework, the goal of which is to give greater insight into black box vision and language models - specifically captioning and VQA models. Specifically, we demonstrate how our evaluation framework is sensitive to a model's understanding of detailed concepts such as color, specific object classes, and counting. We further demonstrate that our evaluation scheme operates independently of other dimensions. That is, our metric can be used to test a model's understanding along an isolated dimension, removing confounding variables. In Chapter 2, we detail the principles of the scheme, models tested, and the datasets used in the evaluation. In Chapter 3, we detail the process used to set up the evaluation framework and the results for baseline comparison.

Second, we describe our low-level analysis using the Pythia VQA model, aiming to further understand these black box systems. Specifically, in Chapter 4, we address

several model failures that motivate our analysis before introducing a scheme designed to test the robustness of the Pythia model by querying it with a number of binary questions that require composing information from multiple sources in the image and question. In Chapter 5, we introduce a method to correct for model inconsistencies by devising a scheme to verify the internal-consistency of the model. That is, we verify if the model responds with contradictory answers to different questions on the same image and, if so, correct the prediction.

Finally, in Chapter 6 we discuss extensions of this research into the broader scope of machine learning and interpretability, specifically into the realm of finance. We give a high level overview of the history of computer vision in finance before delving into instances of poor interpretability for black box models, helping to motivate our evaluation scheme as it aims to provide a lens into the performance of such models. Finally, we address practical applications for VQA and captioning models in the use of insurance companies and hedge funds.

1.3 Related Work

1.3.1 Visual Question Answering

The Visual Question Answering Challenge was first created when the original VQA paper was published in 2015 [3]. The VQA dataset contains 204,721 images from the MSCOCO dataset [34] along with over 760K questions. However, this dataset had notable shortcomings that made models trained on the dataset perform poorly on real-life, general questions. The VQA 2.0 dataset [18] attempted to combat this by balancing the question priors in the original dataset and creating lists of complementary pairs (which will be integral in the evaluation of our evaluation metric). These complementary pairs are similar images that have different answers to the same question.

Current VQA evaluation is performed using the VQA metric [3], which tests whether the proposed answer is shared by at least 3 human annotators. While this metric does a good job of determining if the model correctly answered the question, as a scheme, it doesn't probe how well (or poorly) the model composes information in the image.

Many datasets have attempted to address tasks similar to VQA, most notably Visual Genome (VG) [27] and VisDial (Visual Dialog) [10]. The VG dataset is particularly interesting because it contains 108,077 images (many of which come from MSCOCO) and scene graphs for each image, which create a mapping of relationships between objects in the images. Because some of these images came from MSCOCO, it means that a significant portion is also already in the VQA dataset, allowing researchers to leverage different data splits created there. GQA [21] built off of VG to create a dataset with 22M balanced open-ended questions generated from the scene graphs from VG. An added benefit of the GQA dataset is that the scene graphs and images contained within have been further refined since the original VG dataset, making the data easier to work with. In some experiments, we leverage the GQA scene graphs to manually create our own question datasets.

In addition to datasets adding/revising questions to help improve VQA metrics and understanding of relationships between objects in the images, there has also been a big push to improve the consistency of VQA models. For example, the same model shouldn't return different answers for equivalent questions. In the GQA paper [21], Hudson and Manning propose new metrics to evaluate consistency between entailed questions. Similarly, Ray, et al. [43] created a new question dataset based on MSCOCO images that contains entailed questions to heavily penalize models that usually fail when answering entailed questions. While both of these datasets test the consistency of VQA models, they do so on the dataset level. In our work, we will impose consistency constraints within the model itself so as to be dataset-agnostic.

Finally, since the competition spawned in 2015, a number of papers have made great strides in improving performance on the VQA challenge. The Bottom-Up Top Down (BUTD) [2] architecture that won the 2017 challenge is one of the most prevalent baseline models for the task and has influenced many other successful models along the way. The Pythia model [23] that won the 2018 challenge also employed a top down architecture but made some enhancements to the baseline BUTD model.

1.3.2 Captioning

Current evaluation metrics used for captioning include BLEU [40], CIDEr [53], METEOR [12], ROUGE [32], and SPICE [1]. While each of these scores correlates with human-created captions as demonstrated in their respective papers, we assert that these metrics fail to accurately capture the ability of models to reason along the dimensions of individual detailed concepts. In fact, not only do these metrics output a single value for the score, we will show that they are agnostic to the detailed concepts learned in the dataset. Similar to our work in VQA, this paper will aim to create a new evaluation scheme for captioning that helps evaluate the compositionality of models.

There are a number of top performing captioning models currently available. However, with the plethora of evaluation metrics used to measure captioning, comparing between different models can often be seen as a trade-off as each metric weights different qualities differently. We present a number of models considered state-of-the-art upon release that introduced novel ideas to the task. The Show and Tell captioning model [54] is a standard baseline model that maximizes the likelihood of a caption given an image. The BUTD captioning model [2] shares the bulk of an implementation with the VQA model of the same name as discussed in Section 1.3.1 and introduced the novel concept of initializing focus on the image features. LxMERT [50] is a recent top performing pre-trained cross-modality vision and language model that focuses on

understanding the alignment between the text and image. Based off of the BERT [13] architecture for language understanding, LxMERT combines a language encoder and object-relationship encoder into a cross-modality (involving two methods of perception - language and vision) encoder to generate a joint representation of the vision and language context. Similarly, ViLBERT [37], another pre-trained vision and language model based off of the BERT architecture, places an emphasis on visual grounding.

Chapter 2

Evaluation Framework: Methodology

Along with Chapter 3, this chapter focuses primarily on work done with Zeyu Wang, Professor Olga Russakovsky, Professor Karthik Narasimhan, and Zhiwei Deng for a submission to NeurIPS 2020. We developed an evaluation framework for vision and language models. This chapter outlines the datasets and models that we evaluated using our framework as well as an overview of our contributions in the principles of the framework in Section 2.2.

2.1 Overview of Evaluation Framework

In this chapter, we propose a task-agnostic, unified evaluation framework for vision and language models. This analysis is motivated by the idea of understanding what a black box vision and language system knows, helping to improve interpretability and aid in the development of new models and schemes to overcome the limitations of current systems. Currently, there are two major problems in vision and language models that we address in this chapter. First is that there are many such vision and language models and tasks but no way to compare them. Second is that current met-

rics are very difficult to interpret (particularly in the case of captioning as referenced in Section 1.3.2), making understanding these systems on a low-level impossible.

Taking a look into captioning and VQA in particular, we will further motivate this scheme. When a captioning model publishes with a METEOR [12], CIDEr [53], or SPICE [1] score, readers understand that a higher score correlates with a better model. These papers all presented human studies that indicated their metrics correlate well with human-created captions. However, these scores do not give an understanding into what the captioning model has a strong grasp. Each of these metrics outputs a single score that captures a holistic representation of a model that is useful in comparing different captioning models. Now, consider the VQA metric [3]:

$$\text{Acc}(ans) = \min\left\{\frac{\#\text{ humans that said } ans}{3}, 1\right\}$$

While the VQA metric is simpler to analyze than captioning metrics since it already is an accuracy, the score can further be subdivided by question category in order to calculate the accuracy on different subsets, thereby addressing some of our previous concerns with captioning scores. However, these results do not show whether or not the VQA metric itself is sensitive to detailed concepts. Similarly, it does not demonstrate what causes these models to perform poorly under some circumstances, which we will analyze in Chapters 4 and 5. Throughout this chapter, we aim to highlight the shortcomings of current metrics and propose a new framework that can better accomplish these goals. Our goal is to present an evaluation framework that is sufficient for determining if a model has a strong grasp of a topic or not. We further analyze the model's performance along desired dimensions, thereby giving us greater insight into the detailed concepts of which a model may or may not have a strong grasp.

2.2 Dimensions of Analysis

We want to test our framework by applying it to several different vision and language models to demonstrate that our metric presents a sufficient condition for understanding these detailed concepts. In particular, we will analyze the performance of several models along three dimensions: attributes, object categories, and counting. We want to be able to understand the particular attributes that a model has a grasp of, and the particular attribute we will analyze is the most common attribute found in vision and language datasets - color. We also want to know how well a model can differentiate different objects. And finally, we want to understand how well a model understands counting, which is notoriously difficult in all types of vision and language tasks. Without going into specifics on the implementation of the scheme (see Section 3.1), we will provide an overview of why we chose these particular dimensions, the insight we hope to gain from them, and the forms that questions and captions will take in our dataset.

2.2.1 Choosing Dimensions

A strong evaluation framework should give you deep insight into the model. While these are black box systems and we cannot know exactly how the models make their decisions or generate their responses, we ideally would want to know what to expect from our system. Interpretability of black box models is often what has held back the, practical usage of such models since businesses cannot afford to be wrong for unknown reasons. Several examples of real world failures along these lines that have caused significant financial upheaval can be found in Section 6.2. However, by understanding how vision and language models reason about information on a detailed level, we can gain deeper insight into these systems. We will address this motivation shortly through the Theory of Mind. These ideas hit at the concept of visual grounding.

While we do not propose that well-grounded models are necessary to perform well on vision and language tasks, we assert that grounded models are easier to interpret and reason about and perform well on current benchmarks.

There are several main ways to consider how the vision and language systems reason about information. The first is unary reasoning - how a system reasons about individual objects or entities. Under the idea of language cues in the visual items, this is the fundamental building block of how a model can interact between the different input modalities. In our experiments, we use object classes as a broad category to investigate a model's understanding of unary reasoning. The second form of reasoning is about attributes - the adjectives that describe unary entities. If a model understands unary concepts and their attributes, then it has a rich understanding of the image and associated language and thereby a strong basis for visual grounding [27]. The third category is counting - understanding how many objects of the same or different class exist in an image. While it's very important to understand information about individual objects in an image, it is also important to have a concept of what objects are in the image because if a model knows how many objects are in the image and the locations of them, it can further understand the attributes specific to each one. Finally, the fourth category is relationships between these different objects in the image, which helps to tie the image together. By understanding interactions in the image, models can perform better on the task. In essence, this concept of grounding amounts to generating a scene graph of the image and answering questions based on that. As demonstrated by Anderson et al. [2] among other results, the performance of vision and language models increases significantly when models are pre-trained using scene graphs. Due to difficulty in obtaining and cleaning sufficient data for comparing object relationships, we ultimately decided to forego the experiment and focus on gaining a baseline understanding using the other three methods of reasoning.

Theory of Mind

In the process of designing our evaluation scheme, we realized that there was a deep connection to research into Artificial Intelligence and the Theory of Mind, which aims at reasoning about machine learning systems. Specifically, in Chandrasekaran et al. [7], we see a human study that demonstrates that humans can easily be taught to predict the output of a VQA model given some input. This study shows that, even in spite of the inconsistencies in the responses output by vision and language systems, humans can understand and interact with these systems on a low level. In particular, they can understand the types of questions that the models do not excel at either and therefore account for those errors when predicting the model's response. In Nematzadeh et al. [39], we similarly recognize the language related task of understanding the beliefs of a model in order to help it track inconsistencies in the world. We reference these papers as we aim to do something similar in a vision and language setting by probing these systems to understand how they reason about detailed concepts.

2.3 Datasets

We now present the datasets used for testing our evaluation framework both for captioning and for VQA. However, we note that in Chapters 4 and 5, we will introduce additional datasets as well as continue the use of several detailed in this section.

2.3.1 VQA dataset

The Visual Question Answering dataset takes two forms. The initial dataset was VQA 1.0 [3] that took the MSCOCO [34] images and generates over 1M questions about the images, each with 10 ground truth answers to accompany them. However, as this dataset had an imbalanced language prior, VQA 2.0 [18] soon arrived and

corrected for many of the issues in the original dataset.

MSCOCO Images

COCO [34] is a large scale image dataset published by Microsoft in 2015 that has become the premier computer vision image dataset for higher order tasks such as object detection. The dataset contains 328K images, 80 object categories, and over 2.5M labeled instances. Further, these images contain pixel-level annotations of all object instances for a total of 1.5M object instances. This dataset further serves as the starting point for many more advanced datasets as well due to the vast number of images [8].

2.3.2 VG and GQA datasets

In order to gain access to attributes, object classes, and the counts of objects in the image, we need to access the scene graphs of images. While some image annotations are considered complete (for instance, MSCOCO contains pixel-level annotations), they do not contain the scene graphs that robustly characterize the relationships between objects in the image. Fortunately, Visual Genome and GQA present scene graphs that we leverage throughout our experiments, including those in Chapters 4 and 5. While we ultimately utilize GQA scene graphs, we first introduce Visual Genome as it is the foundation of GQA.

Visual Genome

First released in 2017, Visual Genome (VG) was one of the largest, complete datasets for computer vision research. Containing over 108K images, the VG dataset is more than 25% larger than the VQA training set, which contains only 82K images. However, the true benefit of the VG dataset is the scene graphs that are associated with each image in it. Containing an average of 35 objects, 26 attributes, and 21 pairwise

relationships between the objects, these scene graphs documented not only most of the objects in the images but also the relationships between the objects and different attributes (such as color or size) that they might possess [27]. They additionally leverage WordNet synsets to identify synonymous relationships and object classes [38].

The VG dataset comes from the intersection of MSCOCO images (those that form the basis for the VQA dataset) [34] and YFCC100M [52], the union of which presents a rich dataset of images and tags that simplify the process of grounding image and language. They then gather over 50 annotations [27] that help to create a nearly complete scene graph of all relations of objects in the image. While VQA models are not trained using the VG scene graphs, using such graphs allow models to more easily ground visual concepts with the language used in the questions.

While VG does not present a pixel level annotation like the MSCOCO dataset, it presents the scene graphs that not only help in verifying this experiment but also in allowing it to generalize to future research for broader question classes (such as those that pose relational compositional questions rather than the questions of existence posed in this experiment).

GQA

One issue with the initial VG dataset, however, is that it can be messy to work with. While being nearly complete, the scene graphs are complex with non-standard descriptions and relations. GQA was a dataset published in 2019 that took the first steps to clean and trim the VG dataset and process it better for use in VQA. In particular, it standardized the language used in identifying relationships between objects to match that of the VQA 2.0 dataset [21].

GQA goes on further to generate a diverse dataset of 22M questions that leverage this scene graph structure and is a perfect dataset to use to train or evaluate a model

on the relationships between objects in the image. However, we are unable to leverage the pre-generated dataset for use in our experiment as there are very few questions regarding objects not in the image. That not withstanding, we use the GQA scene graphs as a starting point for generating our dataset.

2.3.3 Captioning datasets

For all captioning models evaluated throught the paper, we use the standard MSCOCO dataset and accompanying captions [34]. More specifically, in MSCOCO, all of the more than 300K images are paired with pixel-level annotations that detail 80 classes of objects in the image as well as 5 ground truth captions per image. Because we use the same images in the training set as those from the VQA dataset, we use 82K images in the training set, giving us 410K captions to train on. The validation set consists of 40K images and nets us 200K captions to evaluate as well. For our evaluation scheme however, we will manipulate both the training and testing sets, and we will address this in Section 3.2.

2.4 Model Survey

In order to test our evaluation framework, we selected three models to use. First, we selected a very rudimentary baseline model called Show and Tell [54]. Second, we selected a revolutionary model that grounds language in image features called the Bottom-Up Top-Down captioning model [2]. Third, we have the Pythia VQA model [23], which builds off the groundwork laid by the Bottom-Up Top-Down model, surpassing the VQA adaptation of the model. We will now detail the architectures of these core models. Note that Bottom-Up Top-Down and Pythia will continue to be used in Chapters 4 and 5.

2.4.1 Show and Tell Captioning Model

Show and Tell is based on the premise that a captioning model can be trained in end-to-end fashion by encoding variable length text input into fixed length feature text and maximizing the likelihood of text output conditioned on the image. The details of the model architecture are summarized from Vinyals et al. [54]. Specifically, this scheme translates to:

$$\theta^* = \text{argmax} \sum_{(I,S)} \log p(S|I;\theta)$$

where θ represents the model parameters and S is the ground truth caption corresponding to the input image I. Via chain rule, we can rewrite the probability as:

$$\log p(S|I;\theta) = \text{argmax} \sum_{t=0}^{N} \log p(S_t|I;\theta;S_0,\ldots,S_{t-1})$$

for a caption of length N, which states that the log-likelihood of some output S is just the sum of the log-likelihoods of each the t^{th} word in the caption conditioned on the previous $t-1$ words. Show and Tell uses a Convolutional Neural Network (CNN) based off the ensemble of batch normalized models from Ioffe et al. [22], which achieved state-of-the-art ILSVRC 2014 performance, to represent images.. Show and Tell also trains using stochastic gradient descent and uses a Recurrent Neural Network (RNN) to model the probabilities $p(S_t|I;\theta;S_0,\ldots,S_{t-1})$ where the previous $t-1$ words are encoded as the hidden state h_t. The memory is updated using the function $h_{t+1} = f(h_t, x_t)$ where f is a non-linear function, in this case a Long-Short Term Memory (LSTM) RNN [20]. A diagram of an LSTM can be seen in Figure 2.1. LSTMs are popular RNNs used to combat the vanishing gradient problem when training neural networks. In the diagram shown, there are three gates which ultimately encode how much information from a state is "remembered" (passed on to the next state)

or "forgotten" (muted signal). Because the gates are sigmoid functions, they help to reduce the effect of the vanishing gradient. The "forget gate" controls the amount of information that memory cell c passes from one state to the next. The input gate filters how much of the input (the output of the hyperbolic tangent function h) reaches the memory. Finally, the output gate filters the signal that is passed on to the sigmoid gate to determine a probability distribution over all words while also passing the signal back through recurrent connections to the next state.

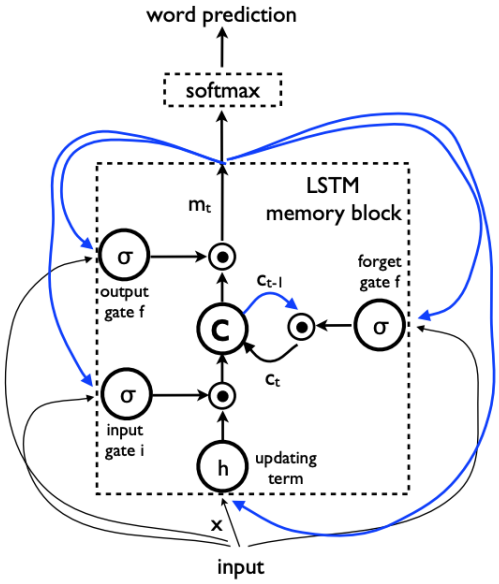


Figure 2.1: In the architecture of an LSTM, input is fed into the model along with the previous state information while various gates determine what proportion of the signal to cancel out [20].

Mathematically, these functions can be represented as follows where all matrices W represent learned parameters, σ is the sigmoid function and h is the hyperbolic

tangent function.

$$
\begin{aligned}
i_t &= \sigma(W_{ix}x_t + W_{im}m_{t-1}) \\
f_t &= \sigma(W_{fx}x_t + W_{fm}m_{t-1}) \\
o_t &= \sigma(W_{ox}x_t + W_{om}m_{t-1}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot h(W_{cx}x_t + W_{cm}m_{t-1}) \\
m_t &= o_t \odot c_t \\
p_{t+1} &= \text{softmax}(m_t)
\end{aligned}
$$

A diagram of the architecture of Show and Tell can be found in Figure 2.2. Each of these LSTMs share parameters as they are the "unrolled" version of the model from Figure 2.1. The image is only passed into the initial state via $x_{-1} = CNN(I)$. Each word S_t is encoded using some weight matrix W_e to generate x_t (the input at each state). Once a specific stop word has been output, the LSTM has finished generating the sentence. During training, the loss for each image is calculated as

$$L(I, S) = -\sum_{t=1}^{T} \log p_t(S_t)$$

which is the negative log-likelihood of choosing the correct word at each step.

2.4.2 Bottom-Up Top-Down Model Architecture

One of the basic premises of the BUTD model is that models that ground the language in the image perform better. That is, top-down task-specific information (such as a question in VQA) passes into the model that triggers visual bottom-up responses from the relevant areas in the image. As compared to conventional vision and language models, which distribute attention uniformly using a grid-based system across the image, the BUTD model targets relevant areas in the image and centers its focus on

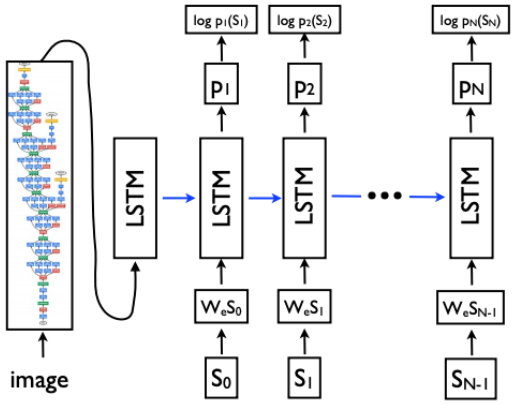


Figure 2.2: In the Show and Tell captioning model architecture, we see the image pass through a CNN before being fed through an unrolled LSTM. At each step, the next word is encoded and fed into the model [20].

the objects and their interactions. In explaining this model, we borrow Anderson's notation [2]. The bottom-up portion of the model outputs k image features, which are represented by $V = \{v_1, \ldots, v_k\}, v_i \in \mathbb{R}^D$, each of which encode a relevant feature in the image. These spatial regions in particular are the bounding boxes output by the bounding boxes of the Faster R-CNN [45] object detection model. The Faster R-CNN model first passes over the image to create a list of target proposals, suppressing multiple labels that reference the same box. In the second pass, it converts the most likely object proposals to feature maps that are combined into a single batch and passed to more CNN layers where the output is converted by a softmax layer to output a probability distribution over class labels for each bounding box. For those final CNN layers, BUTD uses a ResNet-101 [19] pretrained on ImageNet [46]. Finally, to pretrain the Bottom-Up model, they train the model using the Visual Genome [27] dataset, leveraging the scene graphs. For more information on VG, see Section 2.3.2. The Bottom-Up model is task-agnostic, while the Top-Down model is task-dependent.

BUTD Captioning Model

The captioning model utilizes two LSTMs [20] - one for the Top-Down visual component and one for the language component. See Figure 2.3 for a diagram of the model [2]. Consider the simplified equation $h_t = \text{LSTM}(x_t, h_{t-1})$, where x_t is the input vector to the LSTM and h_t is the output vector (therefore, h_{t-1} represents input from the output of the previous time step). Using the output from the Bottom-Up model as input V, Anderson first mean-pools the feature vectors to obtain $\bar{v} = \frac{1}{k}\sum_i v_i$. They concatenate this with the output from the output of the previous step's language model h^2_{t-1} and the embedding of the previously generated word (since there is no text input in a captioning model). That is,

$$x^1_t = [h^2_{t-1}, \bar{v}, W_e \Pi_t]$$

where Π_t is the one-hot encoding representing the word and W_e is the embedding matrix. From there, the output h^1_t gets fed into an attention mechanism which also accepts the feature vectors and computes attentions:

$$a_{i,t} = w_a^T \tanh(W_{va} v_i + W_{ha} h^1_t)$$

These attentions are then turned into probabilities $\alpha_{i,t}$ via a softmax layer. W_{va}, W_{ha}, and w_a are learned weight matrices and vectors. The attentions are then used to compute a convex combination of feature vectors $\hat{v}_t = \sum_{i=1}^{k} \alpha_{i,t} v_i$, which are then passed to the language LSTM along with the output from the Top-Down vision LSTM, h^1_t. Therefore, $x^2_t = [\hat{v}_t, h^1_t]$. The output of the language LSTM is then fed into a softmax layer to calculate the probability of outputting the next word in the caption.

That is, if $y_{1:T}$ denotes the sequence of words up to time T, then we have

$$p(y_t|y_{1:t-1}) = \text{softmax}(W_p h_t^2 + b_p)$$

where W_p and b_p are again learned parameters (weights and biases, respectively). Therefore, the output of an entire sequence is given as the product of these conditional probabilities:

$$p(y_{1:T}) = \prod_{t=1}^{T} p(y_t|y_{1:t-1})$$

For calculating loss, the cross entropy function is used:

$$L_{XE}(\theta) = -\sum_{t=1}^{T} \log(p_\theta(y_t^*|y_{1:t-1}^*))$$

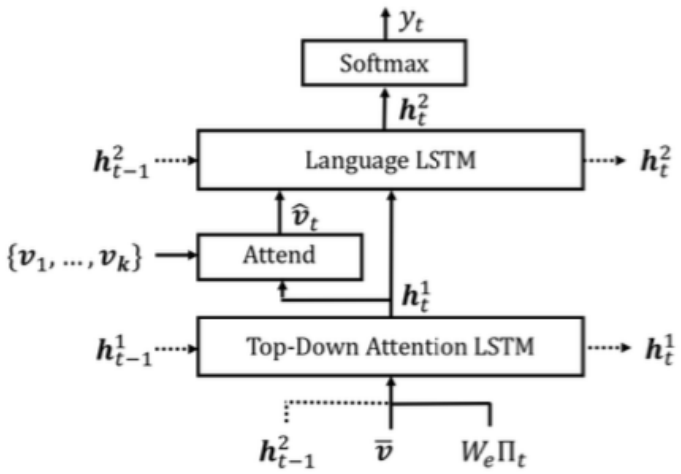


Figure 2.3: In the Top-Down architecture used in the Bottom-Up Top-Down captioning model, we see that text input and image features are fed into the first LSTM before passing through the attention mechanism and language LSTM to generate a prediction [2].

BUTD VQA Model

For the VQA implementation, Anderson again begins with the image features V from the Bottom-Up model. See Figure 2.4 for their implementation of the Top-Down portion of the model. Their implementation uses a multi-modal embedding of the question and image, which as acknowledged by Anderson, has been the basis of several prior works. Next, define the gated hyperbolic tangent function f, which consists of three steps:

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

where W, W', b, and b' are learned, σ is the sigmoid function, and $\circ$ is the Hadamard (element-wise) product. Note that each gated hyperbolic tangent function learns its parameters separately. First, the words in the question are embedded as the hidden state q of a gated recurrent unit (GRU) [9]. The features are concatenated with these hidden states and fed as input to the gated hyperbolic tangent function to generate a_i, which is then converted as before in Section 2.4.2 to a convex combination of feature vectors. That is,

$$\begin{aligned} a_i &= w_a f([v_i, q]) \\ \alpha_t &= \text{softmax}(a_t) \\ \hat{v}_t &= \sum_{i=1}^{k} \alpha_{i,t} v_i \end{aligned}$$

Answers are computed as the output of the sigmoid function combining $h =$

$f(q) \circ f(\hat{v})$ and a learned weight matrix W_o. That is,

$$p(y) = \sigma(W_o f(h))$$

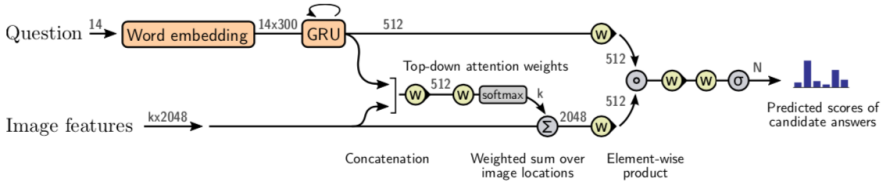


Figure 2.4: In the Top-Down model used in the Bottom-Up Top-Down VQA model, we observe that the question and image features are encoded, concatenated, and passed through a series of gated hyperbolic tangent functions before outputting a distribution over predictions [2].

The BUTD VQA implementation achieved an accuracy of 70.34% in the competition when fully engineered (trained on full training and validation split with an ensemble of 30 models). For more information on both implementations of the BUTD model, see Anderson et al. [2]. More implementation details of the VQA model can also be found in Teney, et al. [51], which elaborates further on the BUTD implementation used in the 2017 VQA challenge. However, those implementation details are less relevant for Pythia's implementation and are therefore omitted.

Pythia Model Architecture

Pythia is a VQA model built off of BUTD and designed by Facebook Research that won the 2018 VQA challenge. We will focus mainly on the incremental changes made to BUTD in this paper [23]. Additionally, the model submitted to the 2018 VQA challenge differed from the version of the model used in this analysis. We will point out specific differences as we proceed. Instead of using the gated hyperbolic tangent function, Pythia utilizes weight normalization [47] and ReLU, which accelerate training without decreasing performance significantly. Additionally, they replace concatena-

tion with Hadamard products. Further, they initialize the word embeddings using GloVe vectors [41] instead of a learned parameter before passing it into the GRU 2.4. In addition, Pythia also tuned many of the hyperparameters of the model including using the Adamax optimizer, which uses the standard Adam optimizer with infinite norm [26]. Another key change made was in the Bottom-Up feature detection model where Pythia replaced the baseline implementation with Detectron, a state-of-the-art object detection model that uses a feature pyramid network to extract and fine-tune features [33].

Now, we will discuss changes that were made solely for the competition and publication. Specifically, Pythia used data augmentation, including additional images and questions from Visual Genome [27] and Visual Dialog [10]. Lastly, Pythia used several different model ensembling strategies that brought together 30 diverse models to increase the performance of the model significantly. From the Pythia ablation study, it is clear that many of their incremental improvements came through the engineering efforts of assembling the ensemble of models. In total, the overall scheme had an accuracy of 72.25% for the VQA competition, eclipsing BUTD by nearly 2 points. However, the Pythia model obtained from their GitHub repository uses a stripped down version of the model and achieves a score of 64.95% when trained on the training set and evaluated on the validation set.

As the results will be particularly relevant in Section 4.2 and Chapter 5, we breakdown the performance of the model by question category of the single-model implementation used in our experiments. For instance, consider questions beginning with the tokens "What", "Is/Are there", and "How many". "Is/are there" questions are binary "yes"/"no" questions, while "What" questions tend to be open ended and "How many" questions require a numeric (or number-like) response. In Table 2.1 we show the results of scoring Pythia on the VQA validation set by question category. In particular, notice that Pythia performs very well on binary questions such as those

that begin with "Is there" or "Are there", while it struggles more with open-ended questions, such as those that begin with "What", or counting questions, such as those that begin with "How many".

	Category Score
Yes/No	83.58
Number	43.95
Other	56.35
Overall	64.95

Table 2.1: We present a summary of VQA scores for the different splits of question categories as achieved by the Pythia model. We observe that the model performs significantly better on binary questions than on other categories.

2.5 Future Work

While we have fleshed out much in this initial set of experiments, there are more avenues to research. First, much like other evaluation metrics and frameworks, it is difficult to conclusively defend a claim that your framework aligns with human belief without a human study. Similar to research into Theory of Mind, a human study would help give more evidence that our framework helps users to understand the beliefs of the model and gain insight into their performance on other dimensions of analysis. Additionally, we have laid the groundwork for analyzing the fourth method by which models ground visual and linguistic information - the relationships between objects. However, a dataset containing a large number of spatial interactions between object classes will be necessary before this can be done. Finally, we have only analyzed this scheme using three models (two captioning models and one VQA model). Further studies could expand this by using other top performing or novel captioning models or pre-trained task-agnostic vision and language models such as LxMERT [50] and ViLBERT [37].

2.6 Concluding Thoughts

In this chapter, we proposed the methodology and motivation behind our unified evaluation framework for vision and language models. We spoke briefly on the inflexibility of current captioning metrics in particular and the lack of insight they give into models. We further emphasize that there is a strong rationale for interpreting models and gaining an understanding of the concepts of which they have a grasp. In particular, we will next analyze the performance of the Bottom-Up Top-Down captioning model, the Show and Tell captioning model, and the Pythia VQA model on their understanding of color, object class, and counting.

Chapter 3

Evaluation Framework: Experiments

In Chapter 2, we introduced our evaluation framework, touching on the motivations behind the scheme and the theoretical justification for using it. Further, we introduced the datasets and models that we would use in evaluating the framework. In this chapter, we introduce the implementation of the framework we will use to analyze vision and language systems along the dimensions of color, object class, and counting. We will then present testing results that correlate strong performance on our framework with an understanding of these detailed concepts.

3.1 Analyzing Along Dimensions

In Section 2.2, we introduced the principle and rational of analyzing captioning and VQA models along different dimensions so as to understand the detailed concepts that different captioning and VQA models do or do not have. Consider Figure 3.1, which contains a simple picture of a reg flag. Unlike many questions in vision and language datasets, the picture clearly contains the flag, the sky, several colors, and nothing else. As such, we would expect any successful vision and language model to

be able to identify that as well, whether it's a captioning model that states "There is a red flag." or if it's a VQA model that answers "yes" to the question "Is there a red flag?" We will proceed with this image and this sample caption and question. We will now introduce the proposed evaluation framework as applied to analyzing the dimension of color.

Figure 3.1: Image of a red flag waving in the sky

3.1.1 Captioning Scheme

For the captioning model, consider the ground truth caption of Figure 3.1 - "There is a red flag." Denote this ground truth caption c_{red}. Now consider caption c_{color} that substitutes a different color for red, for instance c_{blue} is the caption "There is a blue flag." Caption c_{blue} is clearly an incorrect caption as there is no blue flag in the image. Building off of this example, we define a distractor as a caption that changes a word related to the dimension being analyzed that turns a correct caption into an incorrect one. In this case, since the dimension of analysis is color, we changed the ground truth color word "red" into the distractor color word "blue", turning a ground truth caption into an incorrect one. Returning to the captioning model, we want a strong captioning model to tell us that the probability of the ground truth caption

conditioned on the image is greater than the probability of all possible distractor captions. In mathematical notation, this means that we compare $P(c_{red})$ to $P(c_{color})$. For this example, we would therefore desire to have:

$$P(\text{There is a red flag. } | \text{ image}) > \begin{matrix} P(\text{There is a blue flag. } | \text{ image}) \\ P(\text{There is a white flag. } | \text{ image}) \\ P(\text{There is a black flag. } | \text{ image}) \\ P(\text{There is a green flag. } | \text{ image}) \end{matrix}$$

However, even strong performing models may not correctly order the probabilities of the ground truth caption and all distractors. Therefore, for each ground truth caption, we consider the k comparisons between the ground truth caption and the k distractors, and label the comparison correct if $P(c_{red}) > P(c_{color})$ taking the mean accuracy over the k comparisons.

Now, consider the situation where, a priori, "green" is a very unpopular color in the dataset, and as a result, $P(c_{green}|image)$ is often very low. As a result, when calculating the accuracy of a model using this distractor scheme, the model may perform artificially better if there are more distractors that occur with such low probability. Therefore, in our scheme, we use only the top k performing distractors for each caption and image pair. Therefore, for this example with $k = 3$, we would compare $P(c_{red})$ to the three best performing distractors $P(c_{blue})$, $P(c_{white})$, and $P(c_{black})$.

This scheme generalizes nicely to objects and counting dimensions as well. Consider the object class mentioned in the ground truth caption - "flag". Instead of changing the color word to generate incorrect captions, we instead vary the object class in the caption. For instance, given the ground truth caption "There is a red flag.", we would generate distractors using object classes not in the image, such as "dog", "balloon", "chair", and "cup". This analogous extension can be seen in the comparisons of the probabilities as well. We would hope that a strong model would

show the following:

$$P(\text{There is a red flag.}) > \begin{matrix} P(\text{There is a red dog.}) \\ P(\text{There is a red balloon.}) \\ P(\text{There is a red chair.}) \\ P(\text{There is a red cup.}) \end{matrix}$$

Again, we calculate the accuracy of the model on this caption as the mean accuracy over all head-to-head comparisons between the probability of the ground truth caption and each of the distractors. More specifically, in order to show an accurate representation of the model's performance on this dimension, we again use the top k distractors so as to challenge the model the most.

The scheme similarly generalizes to counting with a single caveat: that the grammatical correctness of the sentence might vary slightly between ground truth and caption. Consider the ground truth caption "There is one flag." and the distractor caption "There is many flag." In an effort to reduce affecting the other dimensions, we did not correct for these instances. However, it should be noted that initial experiments did not indicate significant differences between the caption probability of the grammatically correct version and the grammatically incorrect version. Further, even if a model was trained to generate correct captions with incorrect grammar, because high performing NLP models exist to improve the grammatical correctness of such sentences, we felt this was a necessary limitation to accept in order to maintain this basis of comparison.

3.1.2 VQA scheme

We can consider captioning as the basic version of the evaluation framework. Since all of the text comes in one form (a caption of the image), there is a distinct pathway through which to apply this evaluation framework. VQA, however, is more complex as

the text comes in two forms: questions and answers. Further, there are many diverse ways for the text in these two areas to interact as a result of different question types. That is, there are many ways to ask the same question. Consider the question "Is there a red flag?" and associated answer "yes" in response to Figure 3.1. Alternatively, consider the question "What color is the flag?" and the corresponding answer "red". Similarly, it is easier to test in a VQA setting whether or not the model understands if the object tied to the color is in the image at all by asking the questions "Is there a flag?" and comparing the probabilities for the answers of "yes" versus "no". The only related way to test this on a captioning model would be to compare the captions "There is a flag." and "There is not a flag." However, the latter question is clearly drawn from a distribution outside of what the model is trained on and therefore does not accurately measure the understanding of the captioning model. Indeed, it would be inappropriate for a captioning model to output a laundry list of items not in the image rather than describe the contents of it. As a result, since we aim to make a fair comparison between vision and language systems, we do not condition the VQA model on knowing that the related object is in the image. Therefore, we use two schemes to evaluate the performance of the model on the color attribute. The first scheme is quite similar to that of captioning:

$$P(\text{yes} \mid \text{Is there a red flag? , image}) > \begin{matrix} P(\text{yes} \mid \text{Is there a blue flag? , image}) \\ P(\text{yes} \mid \text{Is there a white flag? , image}) \\ P(\text{yes} \mid \text{Is there a black flag? , image}) \\ P(\text{yes} \mid \text{Is there a green flag? , image}) \end{matrix}$$

while the second scheme leverages a second type of question category:

$$P(\text{red} \mid \text{What color is the flag? , image}) > \begin{matrix} P(\text{blue} \mid \text{What color is the flag? , image}) \\ P(\text{white} \mid \text{What color is the flag? , image}) \\ P(\text{black} \mid \text{What color is the flag? , image}) \\ P(\text{green} \mid \text{What color is the flag? , image}) \end{matrix}$$

As before, in each scheme, we compare the probability of the model outputting the ground truth answer as compared to the probability of the answer to the distractor questions. While the color word can be contained in either the question or the answer, leading to more diverse results, the key observation is that the evaluation framework would be testing the model's ability to reason about a single type of question format (either binary or open-ended questions) if we did not open the format to other question types. We considered allowing for even more types of open-ended questions, however we found that supporting the varied amounts of schemes with distractors would net relatively few more training and testing instances overall. For instance, applying a distractor to the question "What is under the red flag?" can be ambiguous as it fails to isolate the desired trait we are testing (that is, the model might answer incorrectly based on the word "under"). Therefore, we stay with only these two schemes.

In a similar fashion to captioning, we can generalize this scheme to support object classes and counting as well. Adapting the binary scheme to object classes is almost the same as before. It reduces to changing the object class instead of the color.

$$P(\text{yes} \mid \text{Is there a red flag? , image}) > \begin{matrix} P(\text{yes} \mid \text{Is there a red dog? , image}) \\ P(\text{yes} \mid \text{Is there a red balloon? , image}) \\ P(\text{yes} \mid \text{Is there a red chair? , image}) \\ P(\text{yes} \mid \text{Is there a red cup? , image}) \end{matrix}$$

However, we found that the dataset has a negligible number of such questions and

therefore do not use them. Instead, for the open-ended case, the scheme is more interesting and more impactful. Since we are only considering questions where the open-ended answer is the object in question, we can observe any such question and answer pair where the distractor's answer is not in the image. As in the case of color, however, we do not test on other open-ended questions where the object is simply contained in the question as it can add confounding variables.

To briefly address the scheme for counting questions, the general scheme is the same as captioning. We again ignore questions of the form "Is there one flag" because there are negligibly few occurrences in the dataset. To allow for open-ended questions, we consider only questions of the form $P(\text{one} \mid \text{How many flags are there?})$ with distractors being of the form $P(\text{many} \mid \text{How many flags are there?})$.

3.2 Differing Levels of Supervision

While in Section 2.2 we described the distractor scheme that would be used to analyze isolated dimensions of each of the models, we will now devise a scheme to show that strong performance on our metric correlates with strong understanding of the dimensions along which these detailed concepts lie. This relies on training a model on multiple datasets with controlled changes between them to measure how the model performance changes as these parameters vary. We introduce different but closely related schemes to target captioning and VQA systems.

3.2.1 Captioning Scheme

Consider a captioning model that has been trained on a dataset that has had all color words removed from the dataset, so that these models can have no understanding of the words "red", "white", or "blue" since they have never seen them before. Therefore, a strong model trained on this modified dataset could output the caption

"There is a flag.", but the probability of outputting "There is a red flag." versus "There is a white flag." should be random chance. We refer to this dataset as having no color supervision since the color words have been removed. Similarly, consider the same original dataset where there is a 50% chance of removing each individual color word. We refer to this data as having 50% color supervision. Likewise, we would refer to the original dataset as having 100% color supervision. Therefore, to verify that our evaluation framework truly is sensitive to the understanding that the models have of these detailed concepts, we can train the same captioning model across the same dataset with different levels of color supervision. We can then plot the performance of the model trained on these different datasets and hope to see a correlation between performance using our color distractor scheme and increasing level of color supervision in the dataset. One point to note is that for datasets at all levels of color supervision, the number of questions remains constant, thereby minimizing the effect that these changes have on the model's performance, helping us to isolate a single dimension to analyze.

While the distractor scheme extended analogously from the color dimension to object classes and counting, testing this scheme at differing levels of supervision does not. In the color supervision case, when you remove a color word (i.e. an adjective), you are still typically left with a valid sentence (or one that, at the very least, is understandable). However, when you remove an object class from a sentence, it often becomes gibberish that doesn't actually test the models' understanding of the dataset without that object. For instance, consider changing "There is a flag." to "There is a." As a result, we do not evaluate our distractor scheme at different levels of supervision along dimensions other than color in this work. That is, the only dimension along which we will vary the level of supervision is color.

Now the task turns to showing that our evaluation framework truly isolates the different dimensions of analysis. That is, to show that our color distractor scheme

measures the performance of that dimension independently of others (and similarly from the perspective of the other dimensions), we can train the models on differing levels of supervision along one dimension and then evaluate using the distractor schemes of other dimensions. Since the dataset varies in nothing other than the presence or absence of the target category words, we would expect the distractor schemes to perform equally well across levels of supervision (except for the one dimension being analyzed).

Further, we can compare between different models trained in the same way. While we do not present a human study that justifies that strictly stronger performance on our task indicates that a model is stronger, we do aim to assert that models with stronger grasps of detailed concepts will perform better on our evaluation metric than weaker models. While using varying levels of supervision makes it simple to compare a model to itself when trained at different levels of color supervision, it could be difficult to assert that different models trained on different datasets can be compared under this scheme. However, for models trained on the same task, we can remove potential confounding variables since we can train them on the same datasets and varying levels of supervision.

Finally, we can actually take these models trained at different levels of color supervision and evaluate them using standard captioning metrics such as METEOR [12], CIDEr [53], and SPICE [1]. Based on our prior claims that these metrics are invariant to detailed concepts, when we evaluate these different models, we would expect to see approximately equal performance across all models in spite of the model having varying levels of an understanding of color.

3.2.2 VQA Scheme

In our scheme for VQA, we apply a scheme similar to that of captioning except that there is a caveat due to the nature of questions and answers. Consider the question

of the form "Is there a red flag?" with the paired answer "yes". We can apply the same scheme as with captioning by removing the color words without changing anything else. However, the issue arises when you have questions with color words in the answer. There were three ways to approach this problem. We could delete these questions entirely from the dataset. The issue with this is that you end up throwing away more than 20K questions to train on. Further, if we wanted to only remove some of the color questions at various levels of supervision, then each of the datasets would have a different size, which could introduce additionally confounding variables into our analysis. Instead of removing the questions with color words, we could replace those answers with a default response such as "don't know" or "unknown", but with this approach, you are implicitly teaching the model that it not only does not understand the color but also that it does not understand the objects mentioned in the question either, thereby decreasing the performance of the model on other dimensions. The third route, and the one we opted for in this analysis, was to convert questions of the form "What color" to questions of the form "Is there a C object" where C is the answer to the original question. While this unfortunately reduces the model's ability to handle questions of the form "What color" outside of the training set, it enables us to compare the models across different levels of supervision more uniformly. Additionally, since we are simply using differing levels of color supervision for comparing models trained on the same types of questions and answers to understand how the model reasons about color, we assert that the model still has a grasp of color even if it does not understand questions that use different types of language to express the same concept. That is, we judge that a VQA model possesses a strong understanding of color if, for any tested question structure, it performs well on the color dimension test.

3.3 Generating Datasets

In this section, we will briefly cover the datasets we used in our experiment with varying levels of supervision. Specifically, we will discuss the metadata of the training data and the validation sets used to test along each dimension. In our scheme, we define color as being one of the following 11 words: 'black', 'blue', 'brown', 'gray', 'green', 'orange', 'pink', 'purple', 'red', 'white', and 'yellow'. We define 80 object classes to be those from the MSCOCO dataset [34]. We define counting words as the 11 numbers 0 through 10.

3.3.1 Captioning Dataset

For generating the captioning datasets, as mentioned in Section 2.3.3, we begin from the MSCOCO captioning dataset. In the MSCOCO validation set, there are 202,654 total captions. We divide these into sets based on if the caption mentions a word for any of the categories. We filter down to 41,927 ground truth captions for the color distractor scheme, 106,465 for the object distractor scheme, and 11,577 for the counting distractor scheme. Each of these sets is augmented by a factor of how many distractors there are for that category. Therefore, the three distractor datasets end up having 461,197 color captions, 8,517,200 object captions, and 127,347 counting captions, respectively.

3.3.2 VQA Dataset

For generating the VQA datasets, as mentioned in Section 2.3.1, we start from the original VQA training dataset. First, we remove all 33,756 of the 443,757 total questions that mention color that do not fit our scheme. That is, we remove questions that mention one of the color words but do not start with "Is there" or "What color is the". In total, there are 690 questions that start with "Is there" and contain a color

word in the original dataset, while there are 27,196 questions that ask "What color is the" in the original dataset. As referenced in Section 3.2.2, we then convert these 27,196 questions into the form "Is there a *color object*?", where *color* is the original ground truth answer to the question and *object* is the referenced object in the question. In the validation set, we have 13,662 ground truth color questions, 21,686 ground truth object questions, and 22,328 ground truth counting questions. Since we needed to convert questions beginning with "What color" to questions beginning with "Is there" in order to maintain a constant numbers of questions over all datasets with varying levels of supervision, we must do the same to the verification questions for color. However, to support our distractor scheme for color, we must augment the dataset with distractor questions (since unlike the other distractor schemes, we have binary questions). Therefore, we end up with 150,272 total color questions.

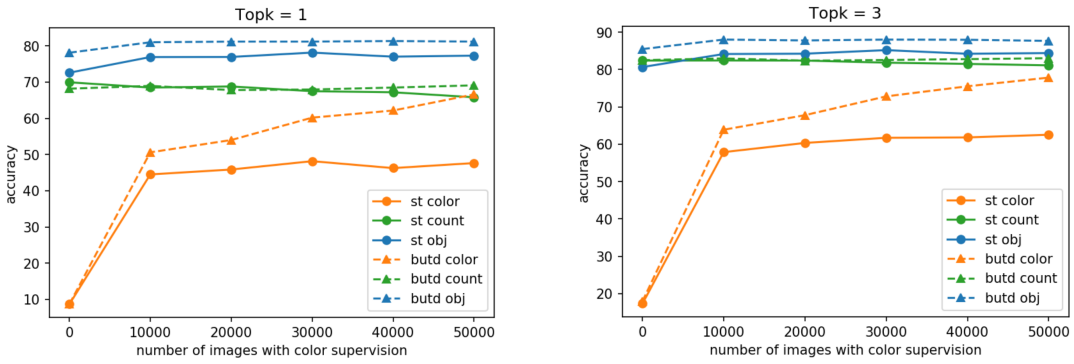
3.4 Experimental Results

We will now discuss the results for the captioning and VQA schemes enumerated in the previous two sections.

3.4.1 Captioning Results

In Figure 3.2, we present a comparison of the Show and Tell captioning model [54] compared with the Bottom-Up Top-Down captioning model [2] trained on increasing levels of color supervision. For each model trained at each level of supervision, we evaluate using all three distractor schemes. As evident from Figure 3.2b, we see that only the performance of the dimension being tested (color in this case) varies with level of supervision. We can compare these results to Figure 3.2a where we see a plot that varies color supervision using the top $k = 1$ distractor scheme instead of $k = 3$. Further, we see the same trends as in the previous figure except that the

accuracies are slightly lower. These trends occur as hypothesized because we are removing head-to-head comparisons between the ground truth probability and the probability of several weaker distractors.



(a) (b)

Figure 3.2: These plots show the accuracy on our evaluation framework versus level of color supervision using the top $k = 1$ distractor scheme for Figure 3.2a and the top $k = 3$ distractor scheme for Figure 3.2b. When evaluating using all distractor schemes, BUTD outperforms Show and Tell in both cases, and only the accuracy on the tested dimension varies with level of supervision. Credit to Zeyu Wang for generating these charts.

Additionally, in Figure 3.3, we evaluate each of the models trained with different levels of color supervision using METEOR, CIDEr, and SPICE. As predicted, however, none of the metrics show any variation in score as the level of color supervision increases, demonstrating that, much like the object classes and counting distractors, performance on these metrics is unaffected by the model's ability to grasp the detailed metric of color.

A final point to note is that the Bottom-Up Top-Down captioning model significantly outperforms the standard Show and Tell model along the color dimension while it shows less differentiation for the other schemes. However, we would expect BUTD to perform better on the analysis based on its performance on other metrics when compared to Show and Tell.

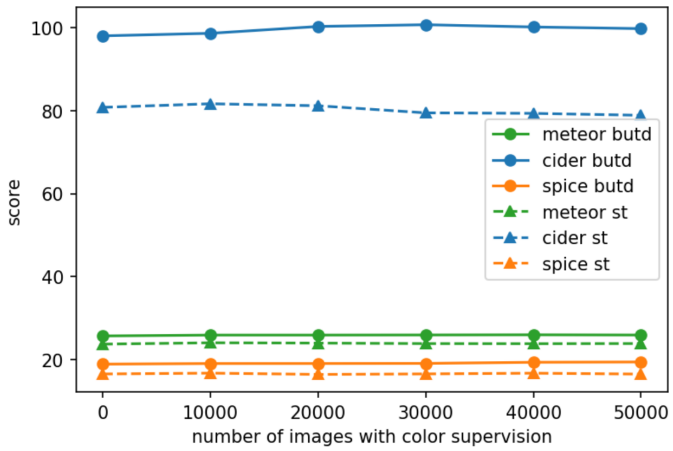


Figure 3.3: This plot shows three different captioning evaluation metrics versus level of color supervision. All three rank BUTD above Show and Tell, which is consistent with our framework, but none of their scores vary with level of color supervision. Credit to Zeyu Wang for generating this chart.

3.4.2 VQA Results

While the results for captioning using our scheme were promising, the scheme did not seem to robustly adapt to VQA in the manner we had hoped. In Figure 3.4, we see that unlike Figures 3.2a and 3.2b, the performance of the color distractors on this split achieve essentially the same score regardless of level of color supervision. However, we see that the performance is essentially 100% for colors, which indicates that something is clearly incorrect with the model performance. Upon analyzing the models on different splits, we found a flaw in the design of the adaptation of the scheme from captioning that we had not anticipated. Namely, in order to increase the number of questions we could consider without varying the number of questions in the dataset at different levels of supervision, we ultimately converted "What color" questions into binary questions beginning with "There is". The initial VQA dataset has approximately 16K questions beginning with "Is there" in the training set, of which about half have answers that are "yes". By converting approximately 27K questions beginning with "What color" to questions beginning with "Is there" (all of which had the answer "yes" under our scheme), we imbalanced the dataset. Now,

instead of having a balanced question prior, we had over 80% of the total questions with the answer "yes". Because this was significantly above the overall performance of the model (64.95%), this caused Pythia at all levels of color supervision to answer "yes" to nearly all questions beginning with "Is there", outputting the answer with probability 1.0 (within numerical error) according to the model. Therefore, since all of the scores were equal, the distractor scheme (depending on how ties were broken) would either give the overall scheme a 0% or 100% since all of the answers were the same (depending on how ties are broken).

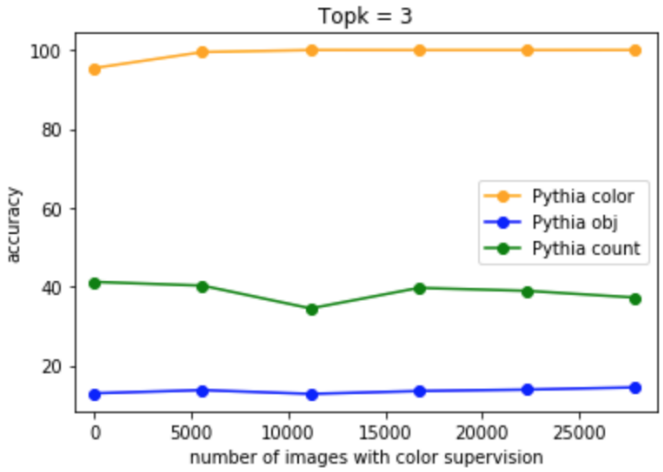


Figure 3.4: This plot shows the accuracy using our evaluation framework versus level of color supervision. The scheme did not generalize to VQA as evidenced by the strange results on all question types and independence of the color distractors when trained on varying levels of color supervision.

3.5 Future Work

There are several insights to draw from this analysis, in particular for the performance of the captioning model. While these results demonstrate that our evaluation framework truly reflects the understanding of detailed concepts that these captioning models have, we do not yet have a conclusive study that allows us to compare models trained on different tasks. These models must be evaluated in slightly dif-

ferent ways that places them on equal footing in order to understand which model truly has a better grasp of these concepts. Further, we have demonstrated a scheme to train captioning models at varying levels of color supervision, and we hope to be able to expand that scheme to the other dimensions of object classes and counting as well. While the initial adaptation of the scheme to VQA did not yield the desired results, there are many adaptations to the scheme that could overcome the issues faced in these experiments. Specifically, we propose converting questions beginning with "What color" to "Is there" questions but adding two copies to the new training datasets - one with the true color and ground truth answer "yes" and the other with a false color and ground truth answer "no" in order to balance the prior. In spite of the initial results on VQA, we believe this framework demonstrates a novel approach to probing vision and language models. This framework allows us to understand these models on a level that cannot be achieved with current evaluation metrics that with more experiments will be able to accurately evaluate models trained on different tasks.

Chapter 4

VQA Compositionality

One of the particular reasons to research vision and language is due to the nature of combining two complex tasks into one. Visual Question Answering [3], more so than captioning and other related tasks, drew our attention due to the high degree of interactivity between user and system. Perhaps had results at the time been stronger for Visual Dialog [10], our interests would have instead been focused there since the model adds in a third complex element - memory. However, the more baseline task of VQA captured our attention since it was both well-developed but also still had room for improvement. In this particular analysis, we use the Pythia VQA model [23], which won the 2018 VQA challenge.

4.1 Pythia Model Failures

In general, state-of-the-art VQA models perform very well. However, because of the high level of interactivity between user and system, it can be very easy to poke holes in the model and emphasize its weaknesses. To get a grasp of the types of answers output by VQA systems, we initially tested dozens of images from the validation set of the VQA dataset [18] with the Pythia model. While performing strikingly well on many questions, we began to notice a tendency of the model to respond to our questions

with an answer that was not in the image at all. While this clearly is just a simple mistake that is to be expected from the system, probing the model further indicated that the model had a glaring internal weakness. Namely, it responded inconsistently between different questions on the same image. For instance, see Figure 4.1a where we have a sandwich on a plate. Understandably, even a strong model might mistake the sandwich for another type of food (for instance, a slice of cake). Indeed, when given this image and the question "Is there a glass plate under the cake?", the answer is "yes", which is incorrect. However, when given the question "Is there cake?", the answer is correct - "no". A second instance can be found in Figure 4.1b, where a young boy plays with a suitcase. Given the question "Is there a suitcase under the elephant?", the model incorrectly outputs "yes", while it correctly answers "no" to the question "Is there an elephant?".

(a) An image of a sandwich on a plate.

(b) An image of a child with a suitcase.

Figure 4.1: These are two images from the validation set of the VQA dataset for which Pythia answers questions incorrectly.

Clearly then, the model loses some information in the language portion of the model as it fails to connect the necessity of an object existing in the image to being an answer to a question about objects in the image. This brings up two related questions. First, is the model only slightly wrong such that a slight perturbation in the question might rectify the mistake? This first question aims to understand how far off the model is from correctly answering the question. Because the model is trained using a fixed budget of questions while trying to understand a wide variety of

syntaxes, the way we ask the question could affect the answer. For instance, consider two equivalent ways of asking about the color of a red car: "Is there a red car?" and "What color is the car?". While seemingly identical questions, a priori, the model likely performs much worse on open-ended questions than it does on binary questions, so it is likely that the model might understand that there is a car and the car is red but is not able to connect that answer to the question "What color is the car?". The second question brought up is, knowing that issues of this type occur, can we devise a scheme to correct for these types of errors? This second question aims at using a post-processing step to treat the model like a black box and attempt to correct for the mistakes that it makes by verifying entailed questions (that is, simple questions whose answers are implied given the answer to the original, more complicated question). Additionally, this scheme cannot truly succeed if these are random errors. In fact, if this scheme were to succeed, it would imply that there are systematic errors within the model that can be corrected for without knowing anything about the architecture simply by applying a consistency check after the fact. We will address the first question in Section 4.2, while we will address the second question in Chapter 5.

As referenced in Section 2.4.2, the Pythia VQA model, developed by Facebook Research, is a state-of-the-art vision and language model that won VQA Challenge 2018, building off the success of the Anderson Bottom-Up Top-Down (BUTD) model [2], which won the VQA challenge the previous year. The BUTD model is commonly used as the baseline for comparison in many VQA and captioning models as a result of its simplicity and strong performance, relying on the idea of grounding the language in the features detected in the image. However, we will now analyze the robustness of the model and observe the cases in which it does not perform as strongly as its VQA Challenge score indicates. In the next section, we will address a scheme that can be used to understand truly how well the model performs when responding to

compositional binary questions, while in Chapter 5, we will leverage the success of the model on binary questions to improve Pythia's performance on both the "other" and "number" question categories.

4.2 Model Robustness

As addressed in Section 4.1, there are many issues with Pythia, especially those that relate to compositionality. (that is, assembling information from multiple parts of the image to answer a complex question about the image). In this section, we devise a scheme using scene graphs to test different types of simple, compositional questions of the form "Is there A and B?" where A and B refer to objects that may or may not be in the image. This scheme appeals to the branch of questions on which Pythia performs best in a vacuum, thereby giving it the best chance at answering the question correctly. Additionally, this scheme is designed to see how well the model learns from the image and entire question, forcing it to rely less on the language prior in order to perform well on the task and learn more from the images about which it answers questions.

4.2.1 Devising a Scheme to Measure Model Robustness

Based on our initial experiments, Pythia seemed to incorrectly answer questions that dealt with compositional information. Most questions of this type ask a question concerning multiple objects in the image, putting strain on the model because it may need to successfully ground multiple objects in the image in relation to one another in order to correctly answer these questions. Some compositional questions are relational in that they use terms such as "next to" or "on top of" to identify particular spatial relationships in the image. Other types of compositional questions might compare the attributes of different objects such as asking if the color of the car matches the

apple on the table.

While these questions certainly would test the model's ability to answer compositional questions, it is important to first identify how accurately a model can actually identify the existence of different objects in the image. This type of question reduces to questions of the form "Is there... ?" and "Are there... ?". For instance, consider the question "Is there A and B?" or alternatively "Are there A and B?". While the rules of English dictate that the latter form is correct, initial results evaluating the model's performance indicated that the model performed better using the question form "Is there A and B?". Further, by using the simplest high performing question structure, we can identify a baseline of the Pythia model's understanding of compositionality. As such, we used this throughout our experiment. Given this question format, we then identified a strategy for testing the robustness of the model - the stress test and the inversion test.

The Stress Test

Consider again, Figure 4.1a where there is a sandwich on the plate. Consider the original question "Is there a plate under the cake?" that the model incorrectly answers "yes". Consider, then, the following questions that stress the model, seeing how far we can tweak the incorrect portion of the question (that is, the word "cake") until the model answers the question correctly. Given that the model correctly answers "yes" to the question "Is there a plate under the sandwich?", the model feasibly might simply mistake a sandwich for cake and reason that, because the objects are related in category (that is, both cake and sandwiches are food items commonly found on plates) that the answer is still "yes". For this particular question, we apply the perturbations found in Table 4.1.

Surprisingly, this state-of-the-art model incorrectly answers questions that clearly could not have been mistaken from the language prior as we had initially assumed.

Question	Answer
Is there a plate under the cake?	yes
Is there a plate under the cup?	yes
Is there a plate under a dog?	yes
Is there a plate under the elephant?	yes
Is there a plate under the airplane?	yes

Table 4.1: We present a series of questions that are created by perturbing the question that is answered correctly (in this case, "Is there a plate under the sandwich?") to see if these changes cause the model to change its answer. The model answered all of these perturbations incorrectly.

Nowhere in the dataset is there a plate under either an elephant or an airplane, and yet the model confidently responded "yes" to all of these questions.

The Inversion Test

Still observing Figure 4.1a, consider the following question perturbations that invert the order of the objects in the question rather than stress them in the aforementioned test. This inversion tests whether the language and vision parts of the Pythia model are properly aligned or whether the language model puts inappropriate emphasis on certain objects in the question. Consider the perturbations of the initial question "Is there a plate under the cake?" paired with their answers as given by the Pythia model seen in Table 4.2.

Question	Answer
Is there a cup on the plate?	yes
Is there a cake on the plate?	no
Is there a cake on top of the plate?	no
Is there an elephant over the plate?	no
Is there an elephant on top of the plate?	yes

Table 4.2: We present a series of questions that invert the order and relationships of objects mentioned in questions to see if it causes the model to change answers. While not all questions were answered correctly, the model's accuracy improved significantly when placing the queried object that is not in the image first in the sentence.

Here we see a less consistent phenomenon. Rather than uniformly answer "yes"

to all questions, only some answers are answered incorrectly. In general, it seems that changing the order of objects in the question by putting the nonexistent object first caused the model to typically understand that the answer to the question could not be correct. However, this theory does not hold perfectly as it clearly depends both on the incorrect object and the preposition used given the contradictory answers in Table 4.2.

4.2.2 Implementation of Scheme

From the above tests, we have identified some interesting phenomena that occur when the Pythia model answers compositional questions. First, the model tends to place inappropriate weight on the first object that appears in the question. Particularly, for relational questions, inverting the order and relationship of the objects in the question likely will be a strong enough perturbation to cause the model to correctly answer the question.

By using our simplified question scheme testing if the model understands the existence of these objects in the image as identified in Section 4.2.1, we can understand the magnitude of these two effects in a controlled setting to establish a baseline of performance for compositional questions. In order to test this scheme, we first create a dataset using images from the VQA validation set (thereby ensuring that the model has not been trained on any of the images) for which we have scene graphs. To obtain these scene graphs, we leverage Visual Genome [27] and GQA [21] scene graphs. We detail the specifics of these datasets in Sections 2.3.2 and 2.3.2. As a result of these improvements, we use GQA instead of VG in our experiments. Additionally, as part of the GQA dataset, we obtain 22M questions on the images leveraging these scene graphs to test entailment. However, since we require many questions involving objects not in the image, we chose not to utilize these pre-generated questions.

4.2.3 VQA Compositionality Dataset

As referenced in Section 2.3.2, the MSCOCO dataset from which the images of VQA are taken contains complete, pixel-level annotations of all images including 80 object classes. For the task of identifying whether or not an object is in the image, the annotations are useful so long as the desired image is among those 80 classes. However, this level of annotation proved to be limiting in testing the compositionality of the VQA Pythia model even though our goal was to see whether it could identify the existence of objects in the image. Another issue presented is that these annotations do not scale beyond simple object detection, preventing future analysis using different question classes under the same process. Therefore, we leveraged the cleaned scene graphs from the GQA dataset without using their pre-generated questions.

Further, since many of the images in the GQA dataset are from MSCOCO - in particular, the training set of the VQA dataset, we narrowed down the focus of the images used to 2,220 images from the VQA validation set that also have accompanying GQA scene graphs. We then set about generating questions of the form "Is there A and B?"

We first iterate over the scene graphs for all objects to create a list of object classes that we have identified in any of the images. We make sure to use a set since many images have duplicate objects in the scene graphs (e.g. multiple pairs of pants or several different cars), and we do not want to have duplicate questions in the dataset for the same image. We then take the intersection of this set with the vocabulary list for the Pythia model so as not to disadvantage the model by using objects of which it has no conception. We then iterate over the GQA scene graph for each image, and for each object that is in the vocabulary set, we add the word to a list of objects contained in that image. Once we obtained a list of objects in each image, we took the symmetric difference (disjunctive set) for each image between the list of objects in the image and the list of total objects. Since the set of objects in the image is a subset

of the total objects in the dataset, this symmetric difference is the list of objects not in the image. We note that, because GQA scene graphs are not technically complete annotations, there is the chance that some objects listed as not in the image could be in the image (or are synonyms for objects in the image). However, as the VG scene graphs are highly-refined using WordNet synsets [38, 16] and over 50 human annotations, we felt this was a safe heuristic to rest upon.

We then define the set of objects in image i as I_i and the set of objects not in image i as N_i. Once we had the sets I_i and N_i for all images i in the dataset, we randomly selected $n = \min\{10, |I_i|\}$ objects from I_i as the set of objects chosen to be used in questions. We additionally select $m = 0.4 \cdot n$ objects from N_i to be the objects used in questions for image i that are not contained in the image. The constant 0.4 is chosen empirically such that the total number of yes and no questions generated are approximately equal. Define these sets as I_i^* and N_i^*, respectively. We held these sets of objects for each image constant because, like in the specific example given in Sections 4.2.1 and 4.2.1, we wanted to see how the model behaved using the same objects as it is easier to measure the effects of the perturbations when more variables are held constant.

When generating questions, we first generated a list of simple questions of the form "Is there X?" for all objects $X \in I_i^* \cup N_i^*$. In total, we generated 22,159 simple questions where 15,677 had ground truth answer "yes" (that is, the object X was contained in the scene graph) and 6,482 has ground truth answer "no" (that is, the object was not in the image). In theory, the accuracy on these questions should display results similar to the object detection rates of the Faster R-CNN object detector that underlies the vision portion of the Pythia model.

Following this, we generated compositional questions of the form "Is there A and B?". First, we generated 65,939 questions where $A \in I_i^*, B \in I_i^*, A \neq B$. These questions help verify that the model understands that these particular objects are in

the corresponding image. We additionally tested 9,174 questions where $A \in N_i^*, B \in N_i^*, A \neq B$. These questions help verify that the model understands that these particular objects are not in the corresponding image. In theory, if the accuracy of the model on the simple questions is p, then we expect the results on the complex questions to perform slightly better than p^2 a priori, assuming that the model answers each object correctly independent of other factors with probability p.

Next, we generated all pairwise questions between objects that are in the image and those that are not. Formally, using the template "Is there A and B?", we generated 60,393 questions where $A \in I_i^*, B \in N_i^*, A \neq B$. Note that the answer to all of these questions is "no". These questions help to evaluate the stress test to see whether or not the model can properly identify that the second object in the question is not in the image and that therefore the answer to the question must be "no".

Finally, we take all of the compositional questions that we have generated and flip the order of the objects. While this seems somewhat pointless for the questions that comprise two objects that are either both true or both false, we use this as a sanity check to make sure that the model performs approximately as well on the respective reversed categories. However, flipping the questions where the true object is the second one listed and the nonexistent object is the first one allows us to evaluate the model's performance on the inversion test. However, given the way these questions are generated, this also allows us to test the inversion and stress tests at the same time.

At a high level of comparison, we can just compare the raw performance values across the different subsets of the dataset against each other. However, consider the case where for image i, Pythia answers the simple questions "Is there X?" where X is in the image incorrectly (therefore, it believes X is not in the image). Knowing that the model already does not detect the object in the image, it is more likely to correctly give the answer "no" to the question "Is there X and Y?" where $Y \in N_i^*$. However, it

would give this answer for the wrong reason. Therefore, we will additionally segment the analysis to give more accurate scores to the model for its performance on the stress test.

4.2.4 Evaluating Performance on Compositionality Dataset

As referenced in Section 4.2.3, there are several evaluation schemes used to assess the performance of the Pythia model on the compositionality dataset. The first is the straight comparison between the VQA accuracy across different splits of data. Because the answers are well-defined here, the VQA accuracy reduces to raw accuracy across all question in the dataset (or subset of the dataset). The results are shown in Table 4.3.

Dataset Name	**Counts**	**Score**
all_questions	293171	67.26
simple	22159	79.11
simple_true	15677	78.57
simple_false	6482	80.42
comp	271012	66.29
comp_true	131878	67.94
comp_true_true	65939	68.00
comp_true_true_rev	65939	67.88
comp_false	139134	64.72
comp_false_false	9174	82.27
comp_false_false_rev	9174	82.19
comp_true_false	60393	58.34
comp_false_true	60393	65.80

Table 4.3: We present a summary of counts of questions and results for the Pythia model's performance on different question splits of the compositionality dataset. Simple questions are of the form "Is there A?", while compositional questions are of the form "Is there A and B?". "True" and "False" refer to whether or not the object (or objects) are in the image. The term "rev" indicates that we reverse the order of the objects mentioned in the base case of questions.

As expected, a priori, the model performs very well on simple questions regardless of whether the object in question is in the image or not. Similarly, the performance

across the raw composition split is approximately p^2 where $p = 0.80$, the raw performance on the simple questions. We see reassuring results when observing that the performance of compositional questions consisting of two objects in the image is the same regardless of the order, while the performance of the model on questions consisting of two objects not in the image is also the same regardless of order. However, we see more interesting trends when observing the steep drop in performance when one object is in the image and the other is not. Perhaps unsurprisingly given our initial research with the stress and inversion tests, the model performed significantly worse when testing in this manner. However, the surprise arises when we pose the questions "Is there A and B?" where $A \in I_i^*, B \in N_i^*, A \neq B$ to the model. In this case, Pythia performs only moderately better than random chance, achieving a score of 58.34%.

Now that we have addressed a high-level analysis of the model's performance, we look to quantify the performance on the stress and inversion tests as referenced before. First, we go through the model's predictions for simple questions (binary questions referencing a single object), comparing them to ground truth answers, and labeling the predictions as true positives (correctly labels an object as in the image), false positives (incorrectly labels an object as in the image), true negatives (correctly labels an object as not in the image), and false negatives (incorrectly labels an object as not in the image). Once we have obtained this list of objects known to be in (or not in) the image (as well as those thought to be or not to be there), we can perform even more fine-grained analyses.

For the simple questions, out of a total of 22,159, there were 12,318 true positives and 5,213 true negatives. We then tested several categories to see how Pythia performed when it understood these objects in a vacuum. When we performed inference on these subsets of questions after filtering out questions that referenced objects of which the model did not have a grasp, all of the datasets were about 67% of the size of

the initial datasets. For instance, over all composition questions where both objects are true, the model achieved a score of 67.94%. However, when we filter out questions involving objects that the model failed to answer correctly in the simple questions, the score jumps to 82.45%. While not as high as anticipated, it demonstrates that the model performs much stronger when it knows information on both objects referenced. Similarly, consider the initial category where both objects mentioned were false. Pythia initially achieved a score of 82.27% on this split, but after filtering out objects it did not have a grasp of, the performance jumped to 95.40% on that split. Progressing to the stress test, we consider Pythia's performance on the split of questions where it knows that the first object mentioned is in the image but the second object might be ambiguous. Interestingly, when the model knows that the first object is in the image and the second is not, Pythia achieves a VQA score of 58.34% - the same as it does when the first object is also ambiguous. However, when it does not know if the second object is in the image or not, the model achieves no better than random chance with a score of 50.65%. Finally, we consider the inversion test - that is, reversing the order of the question such that the object known to be in the image is the second object listed. When the model knows that the first object is not in the image but that the second one is, it achieves a score of 71.10%, significantly higher than it does on the original split. However, this score drops to 60.45% when it is ambiguous whether the first object listed is in the image or not.

4.3 Concluding Thoughts

Overall, evaluating Pythia using this set of binary questions grants some interesting insights into the innerworkings of the model. As expected, the model performs quite well on simple questions of the form "Is there A?" whether the object A is in the image or not. While the performance drops significantly for any compositional scheme of

the form "Is there A and B?", we notice that the model understands some principles of entailment. While the probability of answering questions where both objects are in the image is lower than expected, we can isolate the language model by selecting only questions where Pythia knows both objects are in the image (as shown by the performance on simple questions). In this case, we see that the model achieves a score of 82.45% as opposed to 68.00%, indicating that the language model still does not perfectly understand that these question types only test for the existence of the objects. Observing the stress test by conditioning on objects that the model understood were in the image (while the existence of the other object was ambiguous) showed that the model's performance dropped even further to random chance on this split. This indicates that the model does not have a grasp that both objects in the sentence need to be in the image. Further, we can compare these results to the inversion test to see that the model better understands that the answer should be "no" when it knows that the object not in the image appears first. In fact, it indicates that Pythia's language model puts more attention on the first object in the sentence rather than equal weight on all important parts of the question.

Chapter 5

VQA Self-Consistency Analysis

As referenced in Section 4.1, VQA models, and the Pythia model [23] in particular, are vulnerable to learning from priors. Whether they answer questions with objects known not to be in the image or placing inappropriate emphasis on certain words in the sentence, VQA models can struggle to learn both from the language and visual input to the model. This is precisely why the VQA 1.0 dataset [3] is no longer used today, instead being replaced by VQA 2.0 [18], which was redesigned to help combat this language prior. However, in spite of this, there are still many imbalances in both the object types contained in the images and the language used in the questions to describe them. In Chapter 4, we devised a scheme to evaluate how the language prior affects the model's answers to different questions. In this chapter, we develop a post-processing scheme to verify the results of the Pythia model and attempt to correct incorrect predictions.

An additional motivation for using a verification scheme to determine if a model is consistent with itself is guided by computer vision research into fairness. Several recent fairness works impose constraints on models for answering questions when the attention of the model is away from the object being talked about. For instance, Burns et al. [6] prohibits captioning models from using gender specific language when

generating captions about people with objects. Even if the model may have been correct in its original caption, the idea outlined is to be right for the right reason. Similarly in this chapter, we impose contraints on the model in a post-processing scheme to prevent Pythia from responding with an answer if we determine that it is inconsistent with entailed questions.

5.1 Motivating Internal-Consistency

We formally define internal-inconsistency as an instance where the model answers multiple questions about the same image in contradictory ways. For instance, observe Figure 5.1b, which comes from the VQA test dataset and depicts a cat and a car, two objects that do not tend to appear together. Consider the simple question "What is next to the car?" A VQA model could feasibly incorrectly answer "truck" to this question (since trucks are often seen in images with cars). However, it could correctly respond "no" to the question "Is there a truck?", understanding that the object does not exist in the image. These two questions demonstrate that the model is internally inconsistent, since the answers to questions on the same image returned contradictory results. Similarly, consider Figure 5.1a (seen before in Section 4.1), which contains a sandwich on a plate. Based on human experience, the existence of a sandwich tends to imply the existence of a plate. However, cats and cars tend to only be in the same image by happenstance.

In these examples, we see two different situations that could lead to the same outcome. That is, models might respond that $P(\text{wrong answer}|\text{image, question}) > P(\text{correct answer}|\text{image, question})$ due to some prior information from either the image or question.

Consider again the question paired with Figure 5.1a: "What is on the plate?" Given this question, for the answer output by the model to be correct, the object

(a) An image of a sandwich on a plate.

(b) An image of a cat and car.

Figure 5.1: Two MSCOCO images where one contains two commonly paired objects (sandwich and plate) while the other contains two less related objects (cat and car)

must be in the image. For instance, we know that the answer cannot be "cake" since there is no cake in the image. Similarly, if we ask the question "Are there a cat and car?" for Figure 5.1b, if the model answers "yes", then this implies the answers to two simpler questions. Namely, the answers to "Is there a cat?" and "Is there a car?" must also be "yes". This realization drives at the premise of entailment, the deduction of premise Y given premise X. In their recent work, Ray, et al. attack the question of entailment in VQA by proposing a new dataset for models to train on that enforces consistency in answers through a series of entailed questions [43]. However, in this chapter, we analyze this phenomenon through the lens of the Pythia model, treating it as a black box and attempting to correct for its inconsistencies through a simple post-processing framework that assesses the validity of an answer and changes it if it cannot verify that the answer is consistent with other responses given by the model.

5.2 Internal-Consistency Scheme

The scheme we develop should adhere to several tenants for it to be practical and useful for the Pythia VQA model. First, it should improve the performance of the model by a nontrivial amount when testing out of sample. For our purposes, we use

the standard VQA model trained on the standard training set and evaluate it on the validation set. For reference, the baseline VQA accuracy of this version of the Pythia model (single-model trained on this split) is 64.95. We set a target of improvement over the baseline model of 0.75 in the VQA accuracy metric, since according to the Pythia ablation study, the average incremental improvement for the model for each additional feature is aproximately 0.75 [23]. Second, the validation scheme should be applicable to a large number of questions. If it does not apply to a large number of questions, then it will either be too narrow in its applicability or could target rare question types or words that reflect on the dataset more so than the model. For our purposes, we target affecting more than 23K questions (10% of the validation set), since, in combination with a 0.75 improvement, this indicates that the scheme must dramatically improve performance on the dataset. Third, it may make additional queries to the model to help verify its responses, but this should be a constant factor of calls (or else the scheme becomes intractable in practice). For efficiency, we set this to 5. Finally, it should only take into account the question, image, and list of predictions output by the Pythia model. In summary, the targets set for this evaluation scheme are as follows:

1. Improve performance on validation set by 0.75 VQA accuracy
2. Verify over 10% of the questions in the validation set
3. Make fewer than 5 additional queries to the model per question
4. Use only the question, image, and predictions output by Pythia

In designing the scheme, we targeted different types of questions, since in many cases, the response category is heavily implied by the first few words of the question. For instance, phrases like “Is there...” imply that the answer to the question will be “yes” or “no”, while phrases like “How many...” indicate counts and “What color...”

indicate colors. Further, we can observe more abstract schemes under this umbrella as well. For instance, consider questions of the form "What..." where the response output by a model is an object. These all pose plausible schemes for addressing the verification questions, and according to the distribution of leading n-grams (first n words of the question) in the original VQA dataset [3] (the distribution of question categories was not significantly altered in VQA 2.0 [18]), we can see the effect our scheme could have [3]. Observe the distribution chart in Figure 5.2 to see that the aforementioned leading n-grams cover a significant portion of the dataset.

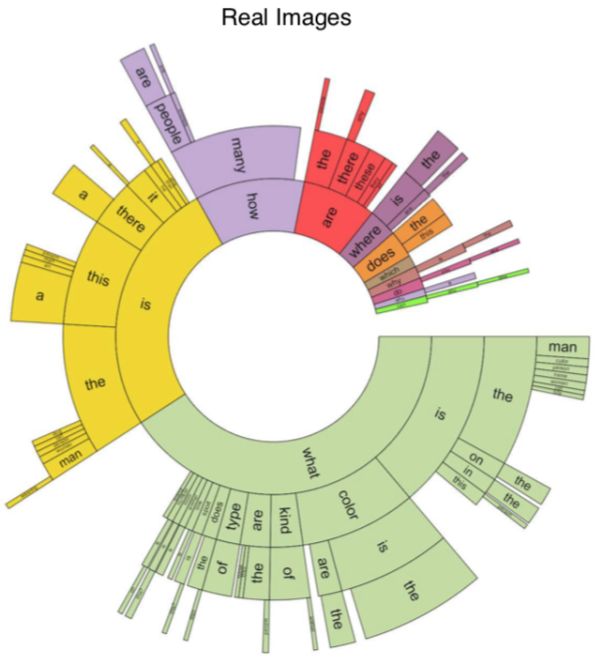


Figure 5.2: "Distribution of questions by their first four words for a random sample of 60K questions" from the VQA 1.0 dataset [3]. Questions that beginning with "What" are the most common, while questions beginning with "Is there", "Are there", and "How many" are also relatively common.

In particular, given that there are 214,354 questions in the validation set, observe Table 5.1, which contains the counts of the questions that begin with each n-gram in our scheme. Specifically, we consider those questions beginning with the phrases "What", "Is there" or "Are there", "Where is" or "Where are", "Is the", and "How

many". Additionally, to measure the effect of segmenting categories down further to allow us to devise more precise schemes, we also remove questions of the form "What color" from the set of questions that begin with "What". We will justify this particular segmentation later in this section.

5.3 Generating Entailed Questions

Now, we will lay out the specific plan used to verify these questions by category. As referenced prior, some of these schemes rely on identifying "objects" in the question or answer. Since, in a real world setting, the VQA model might not have access to the vocabulary list of object classes in the dataset (like we did in Chapter 4), we instead devised a heuristic using the Python NLTK library [4]. The NLTK library is an NLP library that can be used to parse a sentence and identify parts of speech. We used the heuristic that nouns (common and proper, singular and plural) are objects.

- What - For all questions that begin with "What", we observe the proposed prediction P of the model; if it is an object, then we pose the question to the model "Is there P?". If "yes", we do not change the prediction, but if "no", repeat the process with the next highest probability prediction.

- What, no color - For all questions that begin with "What" but not "What color", we repeat the scheme above.

- Is/Are there - For all such questions that contain multiple objects and the prediction output from Pythia being "yes" (that is, implying that all such objects are in the image), we pose the questions "Is there X?" for all objects X in the question. If any return "no", we change the original answer to "no".

- Where is/are - For all questions that begin with "Where is" or "Where are", we use the same scheme as for "What" and use NLTK to test the answer to see

if it is an object or location (using the same heuristic that a noun represents an object or location).

- Is the - For all questions that begin with "Is the", we use the same scheme as "What".

- How many - For questions beginning with "How many", we anticipate that the response will be a number or quantitative measurement X to some object in the sentence. Therefore, we turn the question into "Are there X *objects*?" where *objects* is the first noun found in the sentence.

	Qs. by cat	Fraction total	Num could affect	Num affected
What	88488	0.41	63165	13400
What no color	67959	0.32	43345	10961
Is/Are there	11896	0.06	6093	1038
Where is/are	6675	0.03	4323	1037
How many	23356	0.11	21664	2022
Is the	21557	0.10	10411	5146

Table 5.1: We present a summary of the counts of questions (out of 214,354) based on the n-gram that each question begins with for 6 categories. Further, we include the number of questions that could be affected under the proposed verification scheme and the number whose answers were invalidated and changed under the scheme.

The goal of this verification scheme is to try and leverage the high accuracy rates of the model on questions of the form "Is there X?" to improve the model's performance on more challenging question types. Previously, when we state that we used the heuristic that nouns are objects, we accepted an additional source of error in verifying the answers to these questions by potentially misidentifying an abstract word for an object. For instance, consider the question "What color is the car?" with the answer "blue". While the word "blue" can sometimes be used as a noun (and is indeed identified by NLTK as a noun when used alone), asking the question "Is there blue?" performs worse in Pythia than expected. As a result, in order to help

combat the error incurred from using NLTK, we segmented the "What" category by removing questions that began with "What color".

5.4 Evaluating Performance

In order to evaluate the performance of the model on these different splits, we first run the initial questions through the model in order to get baseline results. We then iterate over each question and prediction list pair and test the question to see if it fits into one of our verification schemes (that is, if the beginning n-gram matches one of our specifications and the additional criteria such as initial prediction is an object). If it does, we then pass the question and prediction pair to a handler function to attempt to verify the initial prediction. For binary questions (such as those beginning with "Is there"), if the output fails the verification test, we flip the answer from a "yes" to a "no", and for other questions, we iterate through the top 5 predictions from the model to see if any of them can be successfully verified. If not, we then return a default answer since the model would be correct for the wrong reasons even if it guessed correctly. After doing this for all questions, we segment our dataset into several different splits for evaluation based on which scheme we tested. Further, we also combine these changed results with the initial predictions to measure the total effect of these changes on the overall performance of the model. As before, we use the VQA Evaluation script for evaluation.

The results can be seen in Table 5.2. Recall that the overall score for Pythia on the validation set is 64.95. First, observe the category scores. These scores are computed as the VQA accuracy on the original Pythia responses across all questions of that category to get an understanding for how each broad category of question performs. As expected based on previous works, binary questions, such as those beginning with phrases like "Is there" and "Are there", have very high accuracy rates. In particular,

the strong performance on "Is/are there" questions achieved the margin for which we had hoped. This high performance was critical since many of our other schemes for verification relied on strong performance of questions like "Is there X?" Further important results were that the performance on other categories hovered between 50 and 60, signifying a large margin for improvement.

	Category Score	Score Pre-Change	Score Post-Change	Score Post-Change (all)
What	57.34	38.80	**42.51**	**65.18**
What no color	51.11	32.55	**43.19**	**65.50**
Is/Are there	85.11	**65.75**	62.45	64.94
Where is/are	36.93	23.11	**47.34**	**65.07**
How many	50.30	52.29	**91.28**	**65.32**
Is the	81.67	**82.62**	16.23	63.36

Table 5.2: We present a summary of scores for the different splits of question categories. For each category, we first show the score on the overall subset of questions for that category before any changes were made. Then, we show the scores on those subsets before and after we perform the verification, bolding the corresponding larger score between "Score Pre-Change" and "Score Post-Change". Finally, we show the score on the overall dataset after making the changes for that subset of questions, bolding the score if it is greater than the performance of the baseline Pythia model - 64.95.

We look to interpret and explain these results in Section 5.5. Now, we call attention to the difference between the scores on the subsets of questions for which we changed answers under our verification scheme and the originals. Important to note is that the scheme proved successful (indeed, quite successful in some cases). In particular, the performance on questions beginning with "How many" dramatically improved from 52.29 (which was already above average for the question category) to 91.28 on the questions changed. From Table 5.1, we see that this improvement was across 2,022 questions out of a total 23,356 questions in the entire category - 8.66% of the whole category. Factoring in the changes, this improves the performance of the entire "How many" category from 50.30 to 53.68.

To our surprise, "Is/are there" performed moderately worse under our scheme, while "Is the" performed abysmally using the post-processing verification, indicating

that our scheme was not appropriately designed to handle incorrect responses for that category. However, the other categories (both "What" categories and "Where is/are") improved by a reasonable amount in each case. Perhaps the most important result being that "What" questions, the most populated category of questions in the entire dataset, representing approximately 1/3 of all questions in the validation set, can be improved using this verification scheme. In particular, when we factored out questions beginning with "What color", the category had the largest overall effect on the overall score of the model. These results not only shows that a great number of questions can be affected by the scheme, but also that further refining the process on broad n-gram categories can lead to significant overall improvements. Finally, we make note that these categories are nearly all mutually exclusive (the obvious exception being "What" and "What no color"), meaning that we can apply these schemes in tandem to give an additive improvement to the model. In particular, when combining the schemes for "What no color", "Where is/are", and "How many", we improve performance of the single-model Pythia model evaluated on the validation set from 64.95 to to 65.99, an incremental gain of 1.04 - well above the margin we were targeting.

5.5 Interpreting Results

While the improvement in accuracy is important and interesting for the scheme, it is even more important to understand from where this incremental gain comes. Ultimately, our verification scheme acts like a regularization parameter by tuning the output of the model to prevent it from outputting values too far from what we expect, and in most cases, regularization tends to negatively affect the performance of a model when the samples come from approximately the same distribution. While the training and validation sets contain different questions and images, the language

and images across the two datasets are relatively similar, meaning that when evaluating out-of-sample, we still expect regularization to decrease the performance of the model. However, it seems that this scheme actually improved the performance of the model, which leaves two likely explanations: the scheme truly helps correct for the language prior or it forces the model to be less explorative in its answers and return "safe" predictions that, under the VQA accuracy metric, match partial ground truth answers.

Table 5.2 indicates that some schemes performed very well while only two impeded the model's performance. Consider the "Is the" scheme, which drastically reduced the performance of the model on that question category. Initially, we believed that questions beginning with "Is the" would have many more multiple choice answers than did other types of questions. However, it appears that the questions we attempted to verify overwhelmingly tended to be binary questions without a uniform scheme to them. Further, the NLTK Python library recognizes the string "yes" as a noun, so using our heuristic, the model attempted to verify every question that received a "yes" prediction by Pythia, which resulted in the verification question "Is there yes?". Clearly this answer is "no", and since the model, a priori, performs very well on questions beginning with "Is the" (81.67 VQA accuracy), this led to getting a large portion of these questions incorrect.

Observing the results for the "Is/are there" category, we see that the initial model performed very well on the questions already. Since our scheme for this category can only turn a "yes" answer into a "no" answer, this means that we are very reliant on the model answering incorrectly almost purely based on language priors - believing some object to be in the image because it commonly appears in the training set with an object recognized in the image. What the scheme cannot overcome as a result of this is the heuristic of treating a noun as an object. Upon analyzing many of the instances where the verification scheme incorrectly flips the result of the model, we noticed that

it tended to be for one of two reasons. The first is that one of the "objects" we try to verify the existence of using "Is there X?" is an abstract or nondescript noun like "options" or place. Less commonly, the question might reference an obscured object such as a "towel rack" covered by a "towel". In this case, Pythia answered "no" that there was not a "towel rack" because it was not visible instead of inferring its existence. As a result of these two phenomena, the verification scheme reduced the accuracy of the model on this category.

For the "Where" scheme, the number of questions changed was fairly small along with the total number of questions in the category. Additionally, many of the questions flipped did not have a complete ground truth answer since VQA requires three human annotations to share an answer in order for it to be ground truth. Of the questions flipped where there was a consensus on the answer, many of them seemed to be due to the fact that Pythia answered almost entirely based on the prior. For instance, Pythia responded "desert" to a question asking "Where is the elephant?" instead of a "zoo" when the picture depicted an elephant behind steel bars with patrons around. However, overall due to the high number of inconsistent human annotations (many of which were similar such as "window" and "windowsill"), the results for this category are less conclusive overall than some others.

The initial results for the "What" category were promising, but upon analyzing them, we found that many of the questions incorrectly changed dealt with color because the model does not perform very well when testing for the existence of a color using "Is there color X?". Once we removed the questions beginning with "What color", the scheme performed even better. Again, however, many of the remaining incorrect changes occurred from testing for the existence of objects that either were not objects at all or were not easily verifiable. One such common example was questions of the form "What is written". It seemed that many of the questions that were correctly flipped were due to Pythia having learned from a prior. For

instance, suitcase and bag were common incorrect answers predicted by the model whenever backpack was the ground truth answer. Another example is that Pythia would commonly output "skis" or "skiing" for an image with a person riding down a snowy mountain instead of "snowboard" or "snowboarding". In both of these cases, the guessed answers were about twice as prevalent in the dataset as the correct answer that was correctly identified by the verification scheme.

Finally, we observe the results from the group that saw the greatest incremental improvement through the scheme - counting questions beginning with "How many". While vision and language models notoriously struggle with counting questions, the verification scheme performs remarkably well. Using MS Excel Pivot Tables, we evaluated the changes that tended to take place in the data and saw several interesting trends. Of the 2,022 instances where the verification check changed the original prediction for a count related question, the change was correct 1,812 times - nearly 90% of the time. It is important to note however that in 945 instances, there were multiple ground truth answers. However, there were 867 instances where the initial prediction was incorrect and the verified answer was correct. In these cases, 495 instances were changes by one (for instance, the initial guess was 2 but corrected to 1). In many cases, this indicates that Pythia was very close to doing well on these questions. However, upon further analysis, we realized that the relative error of these mistakes was high. For instance, 55.6% of the predictions were off by high relative accuracy when the initial prediction was one, two, or three. Additionally, Pythia overwhelmingly tended to predict either one instance too many (38.3%) or far too few (35.9%). There were only 86 instances where Pythia was initially correct but the verification made the answer incorrect, and many of these changes were from the correct answer to either "don't know" or "can't tell".

While Pythia tended to only get answers wrong by a small margin in the "How many" category, the worrying point is the large relative accuracy with which the

model made mistakes. In particular the model tended to incorrectly predict counts when there were a few instances of an object in the image, but by posing the question "Are there X objects?", the model was better able to count the number of detected objects.

5.6 Concluding Thoughts

Referring back to the goals outlined in Section 5.2, we see that we identified a scheme that accomplishes all of these goals. Consider the verification scheme consisting of corrections for "What no color", "Where is/are", and "How many". The implemented Internal-Consistency scheme verifies the answers for 69,332 questions (32.3% of all questions) and improves the performance of the baseline Pythia model by 1.04 from 64.95 to 65.99 - on par with even the most impactful improvements outlined in the Pythia ablation study [23]. In the verification scheme, the only information used came from the original predictions, parsing the question using NLTK, and testing fewer than 5 additional questions.

Ultimately, we are left with a verification scheme that improves the performance of a state-of-the-art VQA model without regard for the inner-workings of the model, meaning that this correctional scheme can generalize to other models as well. Further, while the results and improved performance prove very useful, the analysis in Section 5.5 shows that the new answers output by the model are both reasonable and valid. Our initial skepticism indicated the scheme might have caused the model to return answers based more on a prior. However, it seems that the verification scheme not only enforced the constraint that the model be correct for the right reason but also that it be correct more often as a result by forcing it to pay more attention to the image than the language prior.

Moving forward, our analysis and verification scheme have identified several key

takeaways, laying the groundwork for future self-verification schemes to improve VQA models. We demonstrated that further refining a category, for instance by removing "What color" questions from the "What" category can lead to better incremental improvements in the model. Further, we hypothesize that a different approach to verifying the "Is the" category could yield improved further results. For future analyses, due to the strong performance of this evaluation scheme after the training process, we hypothesize that implementing the verification scheme within Pythia (such that the model self-corrects during training) could improve the model by an even wider margin.

Chapter 6

Practical Extensions

There are many practical applications for vision and language models today. Captioning models can be used to save hundreds of man-hours by automatically generating captions for thousands of images at a time. Visual Question Answering models can be used to answer questions about images or scenes for the visually-impaired. In robotics and self-driving cars, vision and language systems can be used to read road signs and follow detours when navigating. However, there are little explored areas of research for computer vision models, such as in the area of finance. We first observe how computer vision is currently used in finance.

6.1 History of Computer Vision in Finance

Computer vision is currently used in a variety of ways in financial services across the globe. There are no technologies that are truly ubiquitous, however there are several that have grown in their usage. Specifically, these first instances to catch on have been lower risk cases or used as additional verification. Many companies, for instance, have started using facial scans to identify customers and employees as a secondary form of verification before allowing them to interact with sensitive information [24]. We see further applications of this technology, however, at the local bank level where

either a retinal, fingerprint, or facial scan could be required before accessing items and funds stored in the safe-deposit boxes.

Some banks throughout the world have begun deploying new technology that allows users to create a new bank account from the comfort of their own home [24]. Instead of requiring customers to come into a physical location, these banks allow users to use their smartphone camera to digitally verify their identity and speed up the process. Additionally, some businesses have started plans to use computer vision as an initial test for detecting counterfeit money [24]. Currently, many businesses suffer from the inability to check if a currency is counterfeit when paid in per diem, but if this technology pans out as expected, companies accepting per diem may no longer be at risk of such financial losses.

6.2 Failures in Interpretability

As we have addressed throughout this thesis, it is important to gain insight into the models being used so as to anticipate the models' output when fed different input. We explored this motivation heavily with the Theory of Mind in Section 2.2.1. We also designed an evaluation framework for vision and language models to understand how their performance on detailed concepts can be measured in ways that cannot be evaluated with current metrics in Chapters 2 and 3. Finally, we probed the state-of-the-art Pythia model [23] in Chapter 5 to see if we could design a scheme to correct for inconsistent states of the model. We have heavily researched the interpretability of black box machine learning models and have designed schemes to counteract the drawbacks of failures in these models.

However, the stakes for generating an incorrect caption or answer in a VQA model tend to be much lower than mistakes in the realm of finance. If a trading model makes an incorrect prediction, it could lead to a loss of millions of dollars for investors and

employees. For this reason, many finance companies have used black box models sparingly as they tend to lack interpretability. Additionally, because of proprietary information, these trade secrets that disclose what black box models are used tend to be very difficult to come by. From personal experience in finance companies over the past few years, we can say that, while these companies have begun to use such models, their outputs are often aligned with easily interpretable models such as logistic and linear regression models or shallow random forests. For instance, loan companies are required by law to give a reason for adverse action when turning away a customer. If a black box model is used to determine whether or not to give a loan to a customer, there must be some method used to align the scores from the black box model and an interpretable one, or the customer could pursue legal action against the company.

Now, we take a look at a specific instance with Goldman Sachs where lack of interpretability led to the loss of millions of dollars. [31]. In the early 2010s, Goldman Sachs began experimenting with a new machine learning model to do high-frequency trading for them. They unveiled a new technology and set it to work executing trades. After the model had passed their tests, the company deployed the code and, soon thereafter, witnessed it execute a series of poor trades by selling several stocks significantly below the current price. This poor execution caused pandemonium in the stock market and caused many other companies to begin to sell their stock in bulk as they believed Goldman must have had some insight to sell so low. As a result, the stock market tanked several percentage points in thirty minutes causing investors around the world to lose billions of dollars. This phenomenon is known as a flash crash - it comes out of nowhere but also dissipates quite quickly once rational trades begin being executed again. However, in that time, millions of dollars have already changed hands, which could have widespread, unforeseen ramifications [31]. There was a similar instance in Singapore in October 2017, but there is less known about this instance due to proprietary information.

As a result, we observe that evaluation frameworks and verification schemes like those employed in this paper are not reserved to vision and language systems - the analysis here generalizes beyond to broader machine learning research. In particular, finance companies looking to use black box models might consider devising similar evaluation and verification schemes to gain more insight into their models before they deploy them. Additionally, they might consider applying higher levels of regularization to their models to prevent them from outputting responses (or executing trades at prices) that differ too significantly from the training data. Through this process, we believe that companies can and should feel more comfortable integrating black box models into their processes.

6.3 Practical Applications of Vision and Language Models

There are two major uses of imaging data used in financial services that we avoided covering until this point - hedge funds and insurance agencies. First, we consider hedge funds. Hedge funds leverage image data from satellites in order to gain inside information into the brick and mortar stores of companies that they consider investing in. For instance, imagine if a hedge fund considered investing in Walmart before the upcoming holiday season but was not sure how the company was doing recently as the earnings report would not come out until after the holiday. If they had access to satellite imaging of several Walmart parking lots, they could analyze these images and determine if the stores appeared to be empty of full of customers based on how busy the parking lot was. They could then use this information as a proxy in their predictive models. The current issue with this approach is that it can only be done in small scale since VQA models are not currently in use. If we say that Walmart has around 5,000 stores in the United States and that we need 20 images per day of

each parking lot for two weeks before we can accurately include this information in a predictive model, we would need to analyze 1.4M images. Without a machine learning model to process this information, this task becomes impossible. Specifically, by using a VQA model, we allow the hedge fund to leverage not only the characteristics of if a parking lot is full or empty but many other factors as well that can be added as features into the model, thereby improving performance.

Next, we consider an insurance agency whose member got in a car accident. The holder is required to take a picture of the damages and submit the claim to the insurance agency. However, the payout of the claim often depends on waiting for the mechanic that the holder took their car to to inspect the damages. However, with computer vision aiding the company's insight, they can ascertain the validity of the claim, estimate an amount for the damages, and help get the process for repairing the car underway more swiftly. This can be further expedited by allowing insurance agents to query images of the damaged car via a VQA model, allowing them to get the information necessary to expedite the process of assisting their clients. We further propose that faster response rates to claims from insurance holders will improve the company's standing in the eyes of their clientele and lead to an overall growth in business.

6.4 Concluding Thoughts

As demonstrated, there are many varied applications of computer vision and, more specifically, vision and language models in use today. While companies continue to increase their usage of computer vision in financial sectors, they still sit at a rudimentary level of integration, using these models primarily as secondary forms of verification. While these applications are incredibly important in the areas they are used, they do not fully leverage what computer vision has to offer the area. We first

identified areas that highlighted the fears of many corporations when it comes to black box models - failures for unknown reasons. We addressed this scenario by highlighting that an adaptation of our research into evaluation and verification schemes could help companies understand and improve their models, helping to prevent unexpected and drastic failures. Additionally, we addressed two examples in which Visual Question Answering models could prove useful to financial services - namely for hedge funds and insurance companies, demonstrating that vision and language systems could greatly aid companies that are able to leverage image data in their predictive models.

Chapter 7

Conclusion

In summary, motivated by recent trends in vision and language, we sought to analyze models trained on various associated tasks to understand their shortcomings and how we might improve upon them. Specifically, we sought to develop a method of evaluation that gave insight into the detailed concepts that a model understood. We also performed in-depth analyses on the Pythia VQA model in its application to binary questions to understand general trends in the language of questions that cause it to predict incorrect answers. We then applied a verification scheme to improve the performance of the model in cases where we could identify that the model made an incorrect prediction.

First, we developed an evaluation framework to determine if selected captioning and Visual Question Answering models have a comprehensive understanding of vision and language. For our analysis, we used the Show and Tell captioning model, Bottom-Up Top-Down captioning model, and Pythia VQA model. We define this comprehensive awareness to be understanding unary concepts (identifying objects in the images), object attributes (identifying, for instance, the color of objects), and counting concepts (identifying how many of a specific object there are in an image). We proposed our evaluated framework that used a scheme of distractors (replacing a

true portion of the question or caption with a false portion) to compare the probability of the ground truth response to the probability of the distractor. In order to measure the efficacy of our scheme, we trained captioning and VQA models at varying levels of supervision along the color dimension (varying the amount of color words in the dataset) and evaluated it using distractor schemes corresponding to all three dimensions (color, objects, and counting). For captioning, we saw that the distractor schemes for objects and counts did not vary with the level of color supervision while the color distractor scheme did, indicating that our proposed framework succeeded in isolating the target dimension. We then used the same datasets with varying levels of color supervision and evaluated the models' performance using standard captioning metrics to show that performance on these metrics did not improve as the level of color supervision increased. For VQA, we saw that the application of the color scheme did not perform as well as desired. Specifically, we saw that our augmentation of the dataset skewed the prior, resulting in the model answering questions beginning with "Is there" almost universally with the answer "yes".

Then, we took an in-depth look at the Pythia VQA model, motivated by its differing responses to related questions such as "Is there a plate under the cake?" and "Is there cake under the plate?". We identified two schemes - the stress and inversion tests - that helped us to evaluate the model using binary questions that perturbed words in the question (changed words referencing objects in the image to words not in the image) and inverting the order of such words. We found that Pythia had a tendency to place inappropriate attention on the first object referenced in the question, allowing for perturbations to subsequent words to not affect the prediction.

We then devised a verification scheme to correct for mistakes in Pythia's answers without changing the architecture of the model. Specifically, we identified questions beginning with the words "What" (but not "What color"), "How many", and "Where" for which we could correct Pythia's answers. We determined when to change

a prediction by asking verification questions that, if answered incorrectly, would contradict the model's initial answers. Using this scheme, we were able to improve the performance of the model from 64.95% to 65.99%.

Finally, we took a look at the broader applications of computer vision and Visual Question Answering. Specifically, we looked at trends in finance as black box models continue to grow in popularity in the field. We observed instances in which hedge funds and insurance agencies could leverage computer vision, and in particular VQA models, in their valuation of companies and customers, respectively.

Overall, I was very pleased with the research and results. I was particularly happy with the results achieved through the verification scheme proposed as it showed that Pythia, a state-of-the-art VQA system, can actually be corrected through verification schemes that occur after training. While I had hoped that the evaluation framework would extend better to VQA than it did, I believe that future work to adapt the scheme without imbalancing the language prior could show results similar to those from captioning. I believe that the work done in my thesis indicates that there is much more research to be done in evaluation frameworks for vision and language systems as well as verification schemes for VQA models.

Appendix A

Code

I have included some excerpts of my code below.

To score VQA results:

```
import sys
dataDir = '../../VQA'
sys.path.insert(0, '%s/PythonHelperTools/vqaTools' %(dataDir))
from vqa import VQA
from vqaEvaluation.vqaEval import VQAEval

annFile = '''/n/fs/relgeom/VQA/VQA_data/Annotations/
            v2_mscoco_val2014_annotations.json'''
quesFile = '''/n/fs/relgeom/VQA/VQA_data/Questions/
            v2_mscoco_val2014_questions.json'''
resFile = '''/n/fs/visualai-scr/gmccord/composition/
            consistency/original_preds.json'''

# Evaluate above to see if correct
vqa = VQA(annFile, quesFile)
```

```
vqaRes = vqa.loadRes(resFile, quesFile)

# create vqaEval object by taking vqa and vqaRes
vqaEval = VQAEval(vqa, vqaRes, n=2)

# evaluate results
"""
If you have a list of question ids on which you would like
to evaluate your results, pass it as a list to below function
By default it uses all the question ids in annotation file
"""
vqaEval.evaluate()

# print accuracies
print("\n")
print("Overall Accuracy is: %.02f\n"
       %(vqaEval.accuracy['overall']))
```

To generate VQA datasets at different levels of color supervision:

```
imdb_dir = '/n/fs/visualai-scr/gmccord/pythia/data/imdb/vqa/'
np_data = np.load(imdb_dir+'imdb_train2014.npy', allow_pickle=True)

import nltk
def change_what(q_d, color):
    text = nltk.word_tokenize(q_d['question_str'].lower()
            .replace('what color is the ', ''))
    pos_tag = nltk.pos_tag(text)
```

```
    for w,pos in pos_tag:
        if pos in ['NN', 'NNS']:
            break
    new_q = 'Is there a ' + color + ' ' + w + '?'
    new_q_d = q_d.copy()
    new_q_d['question_str'] = new_q
    new_q_d['question_tokens'] = nltk.word_tokenize(new_q.lower())
    new_q_d['all_answers'] = ['yes'] * 10
    new_q_d['answers'] = ['yes'] * 10

    return new_q_d

def replace_color(q_d, r, threshold):
    new_q_d = q_d.copy()
    if r > threshold:
        for c in colors:
            new_q_d['question_str'] =
                new_q_d['question_str'].replace(c, '')
            new_q_d['question_tokens'] =
                nltk.word_tokenize(new_q_d['question_str'].lower())

    return new_q_d

from random import random

count = 0
```

```
count2 = 0
neg_count = 0
q_data = np_data[1:].copy()
c0 = [np_data[0]]
c20 = [np_data[0]]
c40 = [np_data[0]]
c60 = [np_data[0]]
c80 = [np_data[0]]
c100 = [np_data[0]]
for q_d in q_data:
    q = q_d['question_str'].lower()
    ans = q_d['answers']
    if (q.startswith('what color is the ') and
       any(a in colors for a in ans)) or
       ((q.startswith('is there') or
       q.startswith('are there')) and
       any(w in colors for w in q.split())):

        count += 1

        if q.startswith('what color is the '):
            new_q_d = change_what(q_d, ans[0])
            count2 += 1
        else:
            new_q_d = q_d

        r = random()
```

```
            c0.append(replace_color(new_q_d, r, 0))
            c20.append(replace_color(new_q_d, r, 0.2))
            c40.append(replace_color(new_q_d, r, 0.4))
            c60.append(replace_color(new_q_d, r, 0.6))
            c80.append(replace_color(new_q_d, r, 0.8))
            c100.append(replace_color(new_q_d, r, 1))

        elif any(w in colors for w in q.split()) or
             any(a in colors for a in ans):
            neg_count += 1
        else:
            c0.append(q_d)
            c20.append(q_d)
            c40.append(q_d)
            c60.append(q_d)
            c80.append(q_d)
            c100.append(q_d)

print('Desired count: ' + str(count))
print('What color convert count: ' + str(count2))
print('Removed count: ' + str(neg_count))

np.save(imdb_dir+'imdb_color0.npy', np.asarray(c0))
np.save(imdb_dir+'imdb_color20.npy', np.asarray(c20))
np.save(imdb_dir+'imdb_color40.npy', np.asarray(c40))
np.save(imdb_dir+'imdb_color60.npy', np.asarray(c60))
```

```
np.save(imdb_dir+'imdb_color80.npy', np.asarray(c80))
np.save(imdb_dir+'imdb_color100.npy', np.asarray(c100))
```

To generate distractor schemes for VQA:

```
colors = {'black', 'blue', 'brown', 'gray', 'green', 'orange',
          'pink', 'purple', 'red', 'white', 'yellow'}
objects = {'person','bicycle','car','motorcycle','airplane',
'bus','train','truck','boat','traffic light','fire hydrant',
'stop sign','parking meter','bench','bird','cat','dog',
'horse','sheep','cow','elephant','bear','zebra','giraffe',
'backpack','umbrella','handbag','tie','suitcase','frisbee',
'skis','snowboard','sports ball','kite','baseball bat',
'baseball glove','skateboard','surfboard','tennis racket',
'bottle','wine glass','cup','fork','knife','spoon','bowl',
'banana','apple','sandwich','orange','broccoli','carrot',
'hot dog','pizza','donut','cake','chair','couch','potted plant',
'bed','dining table','toilet','tv','laptop','mouse','remote',
'keyboard','cell phone','microwave','oven','toaster','sink',
'refrigerator','book','clock','vase','scissors',
'teddy bear','hair drier','toothbrush'}
counts = {'zero','one','two','three','four','five',
          'six','seven','eight','nine','ten'}

import nltk
def change_what(q_d, color, q_id):
    text = nltk.word_tokenize(q_d['question_str'].lower()
           .replace('what color is the ', ''))
```

```
    pos_tag = nltk.pos_tag(text)

    for w,pos in pos_tag:
        if pos in ['NN', 'NNS']:
            break
    new_q = 'Is there a ' + color + ' ' + w + '?'
    new_q_d = q_d.copy()
    new_q_d['question_str'] = new_q
    new_q_d['question_tokens'] = nltk.word_tokenize(new_q.lower())
    new_q_d['all_answers'] = ['yes'] * 10
    new_q_d['answers'] = ['yes'] * 10
    new_q_d['question_id'] = q_id

    return new_q_d

q_data = np_data[1:].copy()
val_colors = [np_data[0]]
val_objects = [np_data[0]]
val_counts = [np_data[0]]
freq = {}
count = -1
for q_d in q_data:
    q = q_d['question_str'].lower()
    ans = q_d['answers']
    if (q.startswith('what color is the ') and
       any(a in colors for a in ans)):
        count += 1
```

```
    if q_d['image_id'] in freq:
        freq[q_d['image_id']] += 1
    else:
        freq[q_d['image_id']] = 1
    img_id = q_d['image_id'] * 1000

    for a in ans:
        if a in colors:
            word = a
            break

    new_q_d = change_what(q_d, word, img_id + count)
    val_colors.append(new_q_d)
    for c in colors:
        if c != word:
            count += 1
            new_q_d = change_what(q_d, c, img_id + count)
            val_colors.append(new_q_d)

elif (any(a in objects for a in ans)) or
     ((q.startswith('is there') or
     q.startswith('are there')) and
     any(w in objects for w in q.split())):

    val_objects.append(q_d)
elif (q.startswith('how many ') and
    any(a in counts2 for a in ans)):
```

```
        val_counts.append(q_d)

print(len(val_colors))
print(len(val_objects))
print(len(val_counts))

# np.save(imdb_dir+'imdb_val_color.npy', np.asarray(val_colors))
np.save(imdb_dir+'imdb_val_color.npy', np.asarray(val_objects))
# np.save(imdb_dir+'imdb_val_color.npy', np.asarray(val_counts))

# Analyze colors
pid0 = '19624'
pid20 = '19651'
pid40 = '19904'
pid60 = '10097'
pid80 = '161776'
pid100 = '161802'

sup_list = [0, 20, 40, 60, 80, 100]
pid_list = [pid0,pid20,pid40,pid60,pid80,pid100]
target_dir = '''/n/fs/visualai-scr/gmccord/pythia/save
                /vqa_vqa2_pythia/reports/col/'''

indices = []
for pid in pid_list:
    ind = []
    with open(target_dir + pid + '_indices.csv') as csvfile:
```

```
        reader = csv.reader(csvfile)
        for row in reader:
            ind.append([int(x) for x in row])
    indices.append(ind)

actuals = []
for pid in pid_list:
    ind = []
    with open(target_dir + pid + '_actual.csv') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            ind.append([float(x) for x in row])
    actuals.append(ind)

col_scores = []
k=3
val_data = val_colors[1:]
for i in range(len(sup_list)):
    preds_dir = '''/n/fs/visualai-scr/gmccord/pythia/save
                  /vqa_vqaColor{}_pythia/reports/'''
                  .format(sup_list[i])
    pid = pid_list[i]
    count = 0
    score = 0
    j=-1

    for x in range(13661):
```

```
            j += 1
            temp = 1
            if indices[i][j][0] == vocab['yes']:
                a = actuals[i][j][0]
                for _ in range(10):
                    j += 1
                    if indices[i][j][0] == vocab['yes']:
                        if a < actuals[i][j][0]:
                            temp -= 0.33
            else:
                j += 10
                temp = 0

            score += max(temp,0)
            # Ignore other case
        score = score/13661
        col_scores.append(score)
    col_scores

    # Analyze objects
    pid0 = '20843'
    pid20 = '20870'
    pid40 = '11014'
    pid60 = '11040'
    pid80 = '162746'
    pid100 = '162782'
```

```
sup_list = [0, 20, 40, 60, 80, 100]
pid_list = [pid0,pid20,pid40,pid60,pid80,pid100]
target_dir = '''/n/fs/visualai-scr/gmccord/pythia/save
                /vqa_vqa2_pythia/reports/ob/'''

indices = []
for pid in pid_list:
    ind = []
    with open(target_dir + pid + '_indices.csv') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            ind.append([int(x) for x in row])
    indices.append(ind)




ob_scores = []
k=3
val_data = val_objects[1:]
for i in range(len(sup_list)):
    preds_dir = '''/n/fs/visualai-scr/gmccord/pythia/save
                   /vqa_vqaColor{}_pythia/reports/'''
                   .format(sup_list[i])
    pid = pid_list[i]
    count = 0
    score = 0
```

```
for j in range(len(val_data)):
    q_d = val_data[j]
    q = q_d['question_str'].lower()
    ans = q_d['answers']

    if (any(a in objects for a in ans)):
        for a in ans:
            if a in objects:
                word = a

            # do stuff
            word_ind = vocab[word]
            preds = indices[i][j]
            ind = 10
            for k in range(len(preds)):
                if word_ind == preds[k]:
                    ind = k

            if k-ind > 0:
                score += (k-ind)/k

        count += 1

    # Ignore other case

score = score/count
ob_scores.append(score)
```

```
ob_scores


# Analyze counts
pid0 = '17759'
pid20 = '17783'
pid40 = '17811'
pid60 = '8538'
pid80 = '157527'
pid100 = '157564'


sup_list = [0, 20, 40, 60, 80, 100]
pid_list = [pid0,pid20,pid40,pid60,pid80,pid100]
target_dir = '''/n/fs/visualai-scr/gmccord/pythia/save
                /vqa_vqa2_pythia/reports/count/'''


indices = []
for pid in pid_list:
    ind = []
    with open(target_dir + pid + '_indices.csv') as csvfile:
        reader = csv.reader(csvfile)
        for row in reader:
            ind.append([int(x) for x in row])
    indices.append(ind)



count_scores = []
k=1
```

```
val_data = val_objects[1:]
for i in range(len(sup_list)):
    preds_dir = '''/n/fs/visualai-scr/gmccord/pythia/save
                  /vqa_vqaColor{}_pythia/reports/'''
                  .format(sup_list[i])
    pid = pid_list[i]
    count = 0
    score = 0
    for j in range(len(val_data)):
        q_d = val_data[j]
        q = q_d['question_str'].lower()
        ans = q_d['answers']

        if (any(a in counts2 for a in ans)):
            for a in ans:
                if a in counts2:
                    word = a
                    break

            # do stuff
            word_ind = vocab[word]
            preds = indices[i][j]
            ind = 10
            for k in range(len(preds)):
                if word_ind == preds[k]:
                    ind = k
```

```
            if k-ind > 0:
                score += (k-ind)/k

            count += 1

        # Ignore other case

    score = score/count
    count_scores.append(score)
count_scores
```

To measure the internal consistency of Pythia:

```
import json
import nltk

vocab_dir = '/n/fs/visualai-scr/gmccord/pythia/data/vocabs/'
with open(vocab_dir + 'answers_vqa.txt', 'r') as f:
    vocab_l = [x.strip() for x in f.readlines()]

    vocab = set()
for v in vocab_l:
    vocab.add(v)

question_dir = '/n/fs/relgeom/VQA/VQA_data/Questions/'
with open(question_dir +
     'v2_OpenEnded_mscoco_val2014_questions.json', 'r') as f:
    data = json.load(f)
```

```
questions = data['questions']


q_by_img = {}
for q in questions:
    if q['image_id'] in q_by_img:
        q_by_img[q['image_id']].append(q['question_id'])
    else:
        q_by_img[q['image_id']] = [q['question_id']]
q_by_img


count = 0
isThere_dict = {}
for q_dict in questions:
    q = q_dict['question'].lower()
    qid = q_dict['question_id']
    if q.startswith('is there') or q.startswith('are there'):
        count += 1
        isThere_dict[qid] = q
print('Count: ' + str(count))


count = 0
what_dict = {}
for q_dict in questions:
    q = q_dict['question'].lower()
    qid = q_dict['question_id']
    if q.startswith('what'):
        count += 1
```

```
        what_dict[qid] = q
print('Count: ' + str(count))


count = 0
is_the_dict = {}
for q_dict in questions:
    q = q_dict['question'].lower()
    qid = q_dict['question_id']
    if q.startswith('is the '):
        count += 1
        is_the_dict[qid] = q
print('Count: ' + str(count))


count = 0
where_dict = {}
for q_dict in questions:
    q = q_dict['question'].lower()
    qid = q_dict['question_id']
    if q.startswith('where'):
        count += 1
        where_dict[qid] = q
print('Count: ' + str(count))


count = 0
how_many_dict = {}
for q_dict in questions:
    q = q_dict['question'].lower()
```

```
    qid = q_dict['question_id']
    if q.startswith('how many'):
        count += 1
        how_many_dict[qid] = q
print('Count: ' + str(count))

s = set()
for w in sentence.split():
    s.add(w)
v_inter = s.intersection(vocab)
print(v_inter)

tracker = {}
ignore_list = ['photo', 'photos', 'image', 'images', 'picture',
               'pictures', 'pic', 'room', 'rooms']
for k,q in isThere_dict.items():
    text = nltk.word_tokenize(q)
    pos_tag = nltk.pos_tag(text)
    phrases = []
    consec = False
    for w,pos in pos_tag:
        if (pos == 'NN' or pos == 'NNS') and (w not in ignore_list):
            if consec:
                del phrases[-1]
            phrases.append(w)
            consec = True
        else:
```

```
            consec = False
    if phrases != []:
        tracker[k] = phrases
len(tracker)


tracker_g0 = {}
for k,v in tracker.items():
    if len(v) > 1:
        tracker_g0[k] = v
len(tracker_g0)

# pythia_score_images.py
# Adapted from Pythia demo code


import sys
sys.path.append('./content/pythia')
sys.path.append('./content/vqa-maskrcnn-benchmark')


import json


import yaml
import cv2
import torch
import requests
import numpy as np
import gc
import torch.nn.functional as F
import pandas as pd
```

```
import torchvision.models as models
import torchvision.transforms as transforms

from PIL import Image
from IPython.display import display, HTML, clear_output
from ipywidgets import widgets, Layout
from io import BytesIO


from maskrcnn_benchmark.config import cfg
from maskrcnn_benchmark.layers import nms
from maskrcnn_benchmark.modeling.detector import
    build_detection_model
from maskrcnn_benchmark.structures.image_list import
    to_image_list
from maskrcnn_benchmark.utils.model_serialization import
    load_state_dict


from pythia.utils.configuration import ConfigNode
from pythia.tasks.processors import VocabProcessor,
                                    VQAAnswerProcessor
from pythia.models.pythia import Pythia
from pythia.common.registry import registry
from pythia.common.sample import Sample, SampleList
```

```
class PythiaDemo:
    TARGET_IMAGE_SIZE = [448, 448]
    CHANNEL_MEAN = [0.485, 0.456, 0.406]
    CHANNEL_STD = [0.229, 0.224, 0.225]

    def __init__(self):
        self._init_processors()
        self.pythia_model = self._build_pythia_model()
        self.detection_model = self._build_detection_model()
        self.resnet_model = self._build_resnet_model()

    def _init_processors(self):
        with open("./content/model_data/pythia.yaml") as f:
            config = yaml.load(f)

        config = ConfigNode(config)
        # Remove warning
        config.training_parameters.evalai_inference = True
        registry.register("config", config)

        self.config = config

        vqa_config =
            config.task_attributes.vqa.dataset_attributes.vqa2
        text_processor_config =
```

```
            vqa_config.processors.text_processor
        answer_processor_config =
            vqa_config.processors.answer_processor

        text_processor_config.params.vocab.vocab_file =
            "./content/model_data/vocabulary_100k.txt"
        answer_processor_config.params.vocab_file =
            "./content/model_data/answers_vqa.txt"
        # Add preprocessor as that will needed when we are
        # getting questions from user
        self.text_processor =
            VocabProcessor(text_processor_config.params)
        self.answer_processor =
            VQAAnswerProcessor(answer_processor_config.params)

        registry.register("vqa2_text_processor",
            self.text_processor)
        registry.register("vqa2_answer_processor",
            self.answer_processor)
        registry.register("vqa2_num_final_outputs",
            self.answer_processor.get_vocab_size())

    def _build_pythia_model(self):
        state_dict =
            torch.load('./content/model_data/pythia.pth')
        model_config = self.config.model_attributes.pythia
        model_config.model_data_dir = "../../"
```

```
    model = Pythia(model_config)
    model.build()
    model.init_losses_and_metrics()

    if list(state_dict.keys())[0].startswith('module') and \
         not hasattr(model, 'module'):
        state_dict =
            self._multi_gpu_state_to_single(state_dict)

    model.load_state_dict(state_dict)
    model.to("cuda")
    model.eval()

    return model

def _build_resnet_model(self):
    self.data_transforms = transforms.Compose([
            transforms.Resize(self.TARGET_IMAGE_SIZE),
            transforms.ToTensor(),
            transforms.Normalize(self.CHANNEL_MEAN,
                                 self.CHANNEL_STD),
    ])
    resnet152 = models.resnet152(pretrained=True)
    resnet152.eval()
    modules = list(resnet152.children())[:-2]
    self.resnet152_model = torch.nn.Sequential(*modules)
    self.resnet152_model.to("cuda")
```

```
def _multi_gpu_state_to_single(self, state_dict):
    new_sd = {}
    for k, v in state_dict.items():
        if not k.startswith('module.'):
            raise TypeError("Not a multiple GPU state of
                             dict")
        k1 = k[7:]
        new_sd[k1] = v
    return new_sd

def predict(self, url, question):
    with torch.no_grad():
        detectron_features = self.get_detectron_features(url)
        resnet_features = self.get_resnet_features(url)

        sample = Sample()

        processed_text = self.text_processor({"text": question})
        sample.text = processed_text["text"]
        sample.text_len = len(processed_text["tokens"])

        sample.image_feature_0 = detectron_features
        sample.image_info_0 = Sample({
                "max_features": torch.tensor(100,
                                dtype=torch.long)
        })
```

```
        sample.image_feature_1 = resnet_features

        sample_list = SampleList([sample])
        sample_list = sample_list.to("cuda")

        scores = self.pythia_model(sample_list)["scores"]
        scores = torch.nn.functional.softmax(scores, dim=1)
        actual, indices = scores.topk(5, dim=1)

        top_indices = indices[0]
        top_scores = actual[0]

        probs = []
        answers = []

        for idx, score in enumerate(top_scores):
            probs.append(score.item())
            answers.append(
                    self.answer_processor.idx2word(top_in
                        dices[idx].item())
            )

    gc.collect()
    torch.cuda.empty_cache()

    return probs, answers
```

```
def _build_detection_model(self):

    cfg.merge_from_file('''./content/model_data
                /detectron_model.yaml''')
    cfg.freeze()

    model = build_detection_model(cfg)
    checkpoint =
        torch.load('./content/model_data/detectron_model.pth',
                map_location=torch.device("cpu"))

    load_state_dict(model, checkpoint.pop("model"))

    model.to("cuda")
    model.eval()
    return model

def get_actual_image(self, image_path):
    if image_path.startswith('http'):
        path = requests.get(image_path, stream=True).raw
    else:
        path = image_path

    return path
```

```
def _image_transform(self, image_path):
    path = self.get_actual_image(image_path)

    img = Image.open(path)
    im = np.array(img).astype(np.float32)
    im = im[:, :, ::-1]
    im -= np.array([102.9801, 115.9465, 122.7717])
    im_shape = im.shape
    im_size_min = np.min(im_shape[0:2])
    im_size_max = np.max(im_shape[0:2])
    im_scale = float(800) / float(im_size_min)
    # Prevent the biggest axis from being more than max_size
    if np.round(im_scale * im_size_max) > 1333:
            im_scale = float(1333) / float(im_size_max)
    im = cv2.resize(
            im,
            None,
            None,
            fx=im_scale,
            fy=im_scale,
            interpolation=cv2.INTER_LINEAR
     )
    img = torch.from_numpy(im).permute(2, 0, 1)
    return img, im_scale


def _process_feature_extraction(self, output, im_scales,
```

```
        feat_name='fc6', conf_thresh=0.2):
batch_size = len(output[0]["proposals"])
n_boxes_per_image = [len(_) for _ in
            output[0]["proposals"]]
score_list =
        output[0]["scores"].split(n_boxes_per_image)
score_list = [torch.nn.functional.softmax(x, -1) for
                x in score_list]
feats = output[0][feat_name].split(n_boxes_per_image)
cur_device = score_list[0].device

feat_list = []

for i in range(batch_size):
    dets = output[0]["proposals"][i].bbox / im_scales[i]
    scores = score_list[i]

    max_conf = torch.zeros((scores.shape[0])).to(cur_device)

    for cls_ind in range(1, scores.shape[1]):
        cls_scores = scores[:, cls_ind]
        keep = nms(dets, cls_scores, 0.5)
        max_conf[keep] = torch.where(cls_scores[keep]
                > max_conf[keep], cls_scores[keep],
                max_conf[keep])

    keep_boxes = torch.argsort(max_conf,
```

```
                descending=True)[:100]
            feat_list.append(feats[i][keep_boxes])
        return feat_list

    def masked_unk_softmax(self, x, dim, mask_idx):
        x1 = F.softmax(x, dim=dim)
        x1[:, mask_idx] = 0
        x1_sum = torch.sum(x1, dim=1, keepdim=True)
        y = x1 / x1_sum
        return y

    def get_resnet_features(self, image_path):
        path = self.get_actual_image(image_path)
        img = Image.open(path).convert("RGB")
        img_transform = self.data_transforms(img)

        if img_transform.shape[0] == 1:
            img_transform = img_transform.expand(3, -1, -1)
        img_transform = img_transform.unsqueeze(0).to("cuda")

        features =
            self.resnet152_model(img_transform).permute(0, 2,
                3, 1)
        features = features.view(196, 2048)
        return features

    def get_detectron_features(self, image_path):
```

```
        im, im_scale = self._image_transform(image_path)
        img_tensor, im_scales = [im], [im_scale]
        current_img_list = to_image_list(img_tensor,
            size_divisible=32)
        current_img_list = current_img_list.to('cuda')
        with torch.no_grad():
                output =
                    self.detection_model(current_img_list)
        feat_list = self._process_feature_extraction(output,
            im_scales, 'fc6', 0.2)
        return feat_list[0]

# From Zeyu
def predict(demo, url, question, detectron_features,
            resnet_features):
    with torch.no_grad():
        if detectron_features is None or resnet_features is None:
            detectron_features = demo.get_detectron_features(url)
            resnet_features = demo.get_resnet_features(url)
        sample = Sample()
        processed_text = demo.text_processor({"text": question})
        sample.text = processed_text["text"]
        sample.text_len = len(processed_text["tokens"])
        sample.image_feature_0 = detectron_features
        sample.image_info_0 = Sample({
            "max_features": torch.tensor(100, dtype=torch.long)
        })
```

```
        sample.image_feature_1 = resnet_features
        sample_list = SampleList([sample])
        sample_list = sample_list.to("cuda")
        scores = demo.pythia_model(sample_list)["scores"]
        scores = torch.nn.functional.softmax(scores, dim=1)
        scores = scores[0].cpu().numpy()
    gc.collect()
    torch.cuda.empty_cache()
    score_dict = {
        'all_scores': scores.tolist(),
        'yes_score': scores[demo.answer_processor.word2idx('yes')],
        'no_score': scores[demo.answer_processor.word2idx('no')]
    }
    return (detectron_features, resnet_features, score_dict)

def score_images(demo):

    img_dir = '''/n/fs/visualai-scr/gmccord/composition/data
                /images/images/'''
    preds = []

    q_dir = '''/n/fs/visualai-scr/gmccord/composition/code
                /lxmert_all_qs/'''
    with open(q_dir + 'all_questions_sorted.json', 'r') as f:
        qs = json.load(f)

    detectron_features = None
```

```
    resnet_features = None
    prev_id = -1
    for d in qs:
        print(d['image_id'] + ' - ' + d['question_id'], flush=True)
        if d['image_id'] != prev_id:
            detectron_features = None
            resnet_features = None

        path = img_dir + d['image_id'] + '.jpg'
        question = d['question']

        detectron_features, resnet_features, res =
            predict(demo, path, question, detectron_features,
                resnet_features)
        res['image_id'] = d['image_id']
        res['question_id'] = d['question_id']
        preds.append(res)

        prev_id = d['image_id']

    return preds

def main():
    print(torch.cuda.is_available(), flush=True)
    demo = PythiaDemo()

    print('Beginning scoring', flush=True)
```

```
    preds = score_images(demo)

    print('Beginning output', flush=True)
    output_dir =
        '/n/fs/visualai-scr/gmccord/composition/code/'
    with open(output_dir + 'preds.json', 'w') as f:
        json.dump(preds, f)

if __name__ == '__main__':
    main()

import json
import nltk

import pythia_score_images as psi

vocab_dir = '/n/fs/visualai-scr/gmccord/pythia/data/vocabs/'
with open(vocab_dir + 'answers_vqa.txt', 'r') as f:
    vocab_l = [x.strip() for x in f.readlines()]

    vocab = set()
for v in vocab_l:
    vocab.add(v)



question_dir = '/n/fs/relgeom/VQA/VQA_data/Questions/'
with open(question_dir +
    'v2_OpenEnded_mscoco_val2014_questions.json', 'r') as f:
```

```
    data = json.load(f)
questions = data['questions']



q_by_img = {}
for q in questions:
    if q['image_id'] in q_by_img:
        q_by_img[q['image_id']].append(q['question_id'])
    else:
        q_by_img[q['image_id']] = [q['question_id']]


count = 0
isThere_dict = {}
for q_dict in questions:
    q = q_dict['question'].lower()
    qid = q_dict['question_id']
    if q.startswith('is there') or q.startswith('are there'):
        count += 1
        isThere_dict[qid] = q
print('Count: ' + str(count))


tracker = {}
ignore_list = ['photo', 'photos', 'image', 'images',
    'picture', 'pictures', 'pic', 'room', 'rooms']
for k,q in isThere_dict.items():
    text = nltk.word_tokenize(q)
    pos_tag = nltk.pos_tag(text)
```

```
    phrases = []
    consec = False
    for w,pos in pos_tag:
        if (pos == 'NN' or pos == 'NNS') and (w not in
            ignore_list):
            if consec:
                del phrases[-1]
            phrases.append(w)
            consec = True
        else:
            consec = False
    if phrases != []:
        tracker[k] = phrases
print('tracker')

tracker_g0 = {}
for k,v in tracker.items():
    if len(v) > 1:
        tracker_g0[k] = v
print('tracker_g0')



qs = {}
for q in questions:
    qs[int(q['question_id'])] = q['question']
qs
```

```
with open('original_preds.json', 'r') as f:
    pred_list = json.load(f)
preds = {}
for p in pred_list:
    preds[p['question_id']] = p['answer']
print('preds')



demo = psi.PythiaDemo()



def determine_sc_check(q):
    q = q.lower()
    if q.startswith('what'):
        return 'what'
    elif q.startswith('is there') or q.startswith('are there'):
        return 'is/are there'
    elif q.startswith('is the '):
        return 'is the'
    elif q.startswith('where is') or q.startswith('where are'):
        return 'where is/are'
    elif q.startswith('how many'):
        return 'how many'
    else:
        return 'none'
```

```
# If here, original pred == 'yes'
def process_isare_there(img_loc, q, q_id, detectron_features,
            resnet_features):

    if q_id in tracker_g0:
        terms = tracker_g0[q_id]

        for w in terms:
            query = 'Is there ' + w

            detectron_features, resnet_features, top5 =
                demo.predict(img_loc, query,
                    detectron_features, resnet_features)
            if top5[0] != 'yes':
                return (detectron_features, resnet_features, 'no')

    return (detectron_features, resnet_features, None)



# Original pred must be a location
def process_where_isare(img_loc, q, pred, detectron_features,
        resnet_features):
    query = 'Is this ' + pred

    detectron_features, resnet_features, top5 =
        demo.predict(img_loc, query, detectron_features,
            resnet_features)
```

```
    if top5[0] != 'yes':
        detectron_features, resnet_features, top5 =
            demo.predict(img_loc, q, detectron_features,
                resnet_features)

        for w in top5:
            query = 'Is this ' + w
            detectron_features, resnet_features, res =
                demo.predict(img_loc, query,
                    detectron_features, resnet_features)
            if res[0] == 'yes':
                return (detectron_features, resnet_features, w)

    return (detectron_features, resnet_features, None)


# Original pred must be an object
def process_what(img_loc, q, pred, detectron_features,
        resnet_features):
    query = 'Is there ' + pred

    detectron_features, resnet_features, top5 =
        demo.predict(img_loc, query, detectron_features,
            resnet_features)
    if top5[0] != 'yes':
        detectron_features, resnet_features, top5 =
            demo.predict(img_loc, q, detectron_features,
```

```
                resnet_features)

        for w in top5:
            query = 'Is there ' + w
            detectron_features, resnet_features, res =
                demo.predict(img_loc, query,
                    detectron_features, resnet_features)
            if res[0] == 'yes':
                return (detectron_features, resnet_features, w)

    return (detectron_features, resnet_features, None)


# Original pred must be an object
def process_is_the(img_loc, q, pred, detectron_features,
        resnet_features):
    query = 'Is there ' + pred

    detectron_features, resnet_features, top5 =
        demo.predict(img_loc, query, detectron_features,
            resnet_features)
    if top5[0] != 'yes':
        detectron_features, resnet_features, top5 =
            demo.predict(img_loc, q, detectron_features,
                resnet_features)

        for w in top5:
```

```
            query = 'Is there ' + w
            detectron_features, resnet_features, res =
                demo.predict(img_loc, query,
                    detectron_features, resnet_features)
            if res[0] == 'yes':
                return (detectron_features, resnet_features, w)

    return (detectron_features, resnet_features, None)



# obj comes from the question
def process_how_many(img_loc, q, num, obj,
        detectron_features, resnet_features):
    query = 'Are there ' + num + " " + obj

    detectron_features, resnet_features, top5 =
        demo.predict(img_loc, query, detectron_features,
            resnet_features)
    if top5[0] != 'yes':
        detectron_features, resnet_features, top5 =
            demo.predict(img_loc, q, detectron_features,
                resnet_features)

        for n in top5:
            query = 'Are there ' + n + obj
            detectron_features, resnet_features, res =
                demo.predict(img_loc, query,
```

```
                    detectron_features, resnet_features)
            if res[0] == 'yes':
                return (detectron_features, resnet_features, n)

    return (detectron_features, resnet_features, None)


img_dir = '/n/fs/relgeom/VQA/VQA_data/Images/mscoco/val2014/'

# category is the type of consistency check to perform
# ('is/are there', 'what', 'all')
def self_consistency(q_by_img, qs, preds, category):
    preds_changed = {}
    q_id_match = set() # All preds that match category (after
                       # changing the ones that could)
    i=0
    for img_id,q_l in q_by_img.items():

        detectron_features = None
        resnet_features = None
        for q_id in q_l:
            q = qs[q_id]
            cat = determine_sc_check(q)
            i+=1
            if cat != 'none':
                if category == 'all' or cat == category:
                    print(i)
```

```
new_pred = None
if cat == 'is/are there':

    if preds[q_id] == 'yes':
        q_id_match.add(q_id)

        detectron_features,
        resnet_features, new_pred =
        process_isare_there(img_dir +
        'COCO_val2014_' +
        str(img_id).zfill(12) + '.jpg',
        q, q_id, detectron_features,
        resnet_features)
elif cat == 'where is/are':

    text =
        nltk.word_tokenize(preds[q_id])
    pos_tag = nltk.pos_tag(text)
    if pos_tag[0][1] in
        ['NN','NNS','NNP']:

        q_id_match.add(q_id)
        detectron_features,
        resnet_features, new_pred =
        process_where_isare(img_dir +
        'COCO_val2014_' +
        str(img_id).zfill(12) + '.jpg',
```

```
            q, preds[q_id],
            detectron_features,
            resnet_features)
    elif cat == 'what':
        text =
            nltk.word_tokenize(preds[q_id])
        pos_tag = nltk.pos_tag(text)
        if pos_tag[0][1] in ['NN','NNS']:

            q_id_match.add(q_id)
            detectron_features,
            resnet_features, new_pred =
            process_what(img_dir +
            'COCO_val2014_' +
            str(img_id).zfill(12) + '.jpg',
            q, preds[q_id],
            detectron_features,
            resnet_features)
    elif cat == 'is the':
        text =
            nltk.word_tokenize(preds[q_id])
        pos_tag = nltk.pos_tag(text)
        if pos_tag[0][1] in ['NN','NNS']:

            q_id_match.add(q_id)
            detectron_features,
            resnet_features, new_pred =
```

```
                        process_is_the(img_dir +
                        'COCO_val2014_' +
                        str(img_id).zfill(12) + '.jpg',
                        q, preds[q_id],
                        detectron_features,
                        resnet_features)
                elif cat == 'how many':
                    text =
                    nltk.word_tokenize(q.lower()
                        .strip('how many'))
                    pos_tag = nltk.pos_tag(text)

                    if pos_tag[0][1] in ['NN','NNS']:
                        q_id_match.add(q_id)
                        detectron_features,
                        resnet_features, new_pred =
                        process_how_many(img_dir +
                        'COCO_val2014_' +
                        str(img_id).zfill(12) + '.jpg',
                        q, preds[q_id], pos_tag[0][0],
                        detectron_features,
                        resnet_features)

                if new_pred is not None:
                    preds_changed[q_id] = new_pred
print('Done: ' + category)
return (preds_changed, q_id_match)
```

```
preds_changed, q_id_match = self_consistency(q_by_img, qs,
                                             preds, 'how many')
q_id_match_d = {k : 1 for k in q_id_match}

with open('preds_changed_how_many.json', 'w') as f:
    json.dump(preds_changed, f)
with open('q_id_match_how_many.json', 'w') as f:
    json.dump(q_id_match_d, f)
```

To generate compositional questions for Pythia:

```
import json

data_dir = '/n/fs/visualai-scr/gmccord/composition/data/'

print('Load')

with open(data_dir + 'sceneGraphs/val_sceneGraphs.json', 'r') as f:
    val_scene_graphs = json.load(f)
with open(data_dir + 'sceneGraphs/train_sceneGraphs.json', 'r') as f:
    train_scene_graphs = json.load(f)
# Use same total number of unique objects

val_scene_graphs.update(train_scene_graphs)

s = set()
for _,graph in train_scene_graphs.items():
```

```
    for k,obj in graph['objects'].items():
        s.add(obj['name'])

vocab_file = '''/n/fs/visualai-scr/gmccord/pythia/data/vocabs
                        /answers_vqa.txt'''

s_vqa = set()
with open(vocab_file, 'r') as f:
    for word in f:
        if word.strip() in s:
            s_vqa.add(word.strip())

obj_list = [x for x in s_vqa]

import math
import itertools
import random

print('Generate questions')

# Scene graph to use
# Ratio of negative occurrences to positive occurrences
# Empiracally, ratio of 0.4 obtains ~50% positive and negative
# questions
def gen_questions(scene_graph, img_id, object_list, ratio = 0.4):

    scene_graph_objs = set()
```

```
for _,obj in scene_graph['objects'].items():
    scene_graph_objs.add(obj['name'])

n = min(10, len(scene_graph_objs))
m = math.ceil(n * ratio)

# Compute list of objects in scene graph
possible_pos_objects = []
for obj in scene_graph_objs:
    possible_pos_objects.append(obj)
pos_objects = random.sample(possible_pos_objects, k=n)

# Randomly select m objects from the list of objects
possible_neg_objects = []
for obj in object_list:
    if obj not in possible_pos_objects:
        possible_neg_objects.append(obj)
neg_objects = random.sample(possible_neg_objects, k=m)

# Compute all combinations of questions to ask
pos_pos = itertools.combinations(pos_objects, 2)
neg_neg = itertools.combinations(neg_objects, 2)
pos_neg = []
for pos in pos_objects:
    for neg in neg_objects:
        pos_neg.append([pos, neg])
```

```
# Generate questions (simple and composition) and store in dict
questions = {}
answers = {}
ind = 0

simple = []
comp_true = []
comp_true_rev = []
comp_true_false = []
comp_false_true = []
comp_false = []
comp_false_rev = []

simple_answers = []
comp_true_answers = []
comp_true_rev_answers = []
comp_true_false_answers = []
comp_false_true_answers = []
comp_false_answers = []
comp_false_rev_answers = []

# Set constants
yes = 425
no = 1403

# Simple questions
for obj in pos_objects:
```

```
    q = {'image_id': img_id,
         'question' : 'Is there ' + str(obj) + '?',
         'question_id' : img_id + str(ind).zfill(7)}
    simple.append(q)
    a = {'image_id': img_id,
         'labels' : [yes],
         'question_id' : img_id + str(ind).zfill(7),
         'scores' : [1]}
    simple_answers.append(a)
    ind += 1
for obj in neg_objects:
    q = {'image_id': img_id,
         'question' : 'Is there ' + str(obj) + '?',
         'question_id' : img_id + str(ind).zfill(7)}
    simple.append(q)
    a = {'image_id': img_id,
         'labels' : [no],
         'question_id' : img_id + str(ind).zfill(7),
         'scores' : [1]}
    simple_answers.append(a)
    ind += 1

# Composition questions
for objs in pos_pos:
    q = {'image_id': img_id,
         'question' : 'Is there ' + str(objs[0]) + ' and ' +
         str(objs[1]) + '?',
```

```
            'question_id' : img_id + str(ind).zfill(7)}
        comp_true.append(q)
        a = {'image_id': img_id,
            'labels' : [yes],
            'question_id' : img_id + str(ind).zfill(7),
            'scores' : [1]}
        comp_true_answers.append(a)
        ind += 1

        q = {'image_id': img_id,
            'question' : 'Is there ' + str(objs[1]) + ' and ' +
            str(objs[0]) + '?',
            'question_id' : img_id + str(ind).zfill(7)}
        comp_true_rev.append(q)
        a = {'image_id': img_id,
            'labels' : [yes],
            'question_id' : img_id + str(ind).zfill(7),
            'scores' : [1]}
        comp_true_rev_answers.append(a)
        ind += 1
    for objs in pos_neg:
        q = {'image_id': img_id,
            'question' : 'Is there ' + str(objs[0]) + ' and ' +
            str(objs[1]) + '?',
            'question_id' : img_id + str(ind).zfill(7)}
        comp_true_false.append(q)
        a = {'image_id': img_id,
```

```
            'labels' : [no],
            'question_id' : img_id + str(ind).zfill(7),
            'scores' : [1]}
        comp_true_false_answers.append(a)
        ind += 1

        q = {'image_id': img_id,
            'question' : 'Is there ' + str(objs[1]) + ' and ' +
            str(objs[0]) + '?',
            'question_id' : img_id + str(ind).zfill(7)}
        comp_false_true.append(q)
        a = {'image_id': img_id,
            'labels' : [no],
            'question_id' : img_id + str(ind).zfill(7),
            'scores' : [1]}
        comp_false_true_answers.append(a)
        ind += 1
    for objs in neg_neg:
        q = {'image_id': img_id,
            'question' : 'Is there ' + str(objs[0]) + ' and ' +
            str(objs[1]) + '?',
            'question_id' : img_id + str(ind).zfill(7)}
        comp_false.append(q)
        a = {'image_id': img_id,
            'labels' : [no],
            'question_id' : img_id + str(ind).zfill(7),
            'scores' : [1]}
```

```
    comp_false_answers.append(a)
    ind += 1

    q = {'image_id': img_id,
         'question' : 'Is there ' + str(objs[1]) + ' and ' +
         str(objs[0]) + '?',
         'question_id' : img_id + str(ind).zfill(7)}
    comp_false_rev.append(q)
    a = {'image_id': img_id,
         'labels' : [no],
         'question_id' : img_id + str(ind).zfill(7),
         'scores' : [1]}
    comp_false_rev_answers.append(a)
    ind += 1

questions['simple'] = simple
questions['comp_true'] = comp_true
questions['comp_true_rev'] = comp_true_rev
questions['comp_true_false'] = comp_true_false
questions['comp_false_true'] = comp_false_true
questions['comp_false'] = comp_false
questions['comp_false_rev'] = comp_false_rev

answers['simple'] = simple_answers
answers['comp_true'] = comp_true_answers
answers['comp_true_rev'] = comp_true_rev_answers
answers['comp_true_false'] = comp_true_false_answers
```

```
    answers['comp_false_true'] = comp_false_true_answers
    answers['comp_false'] = comp_false_answers
    answers['comp_false_rev'] = comp_false_rev_answers

    return (questions, answers)

qa_set = []

for img_id in val_scene_graphs:
    qa_set.append(gen_questions(val_scene_graphs[img_id], img_id,
            obj_list, 0.4))

print('Make question sets')

simple_qs = []
comp_true_qs = []
comp_true_rev_qs = []
comp_true_false_qs = []
comp_false_true_qs = []
comp_false_qs = []
comp_false_rev_qs = []

simple_as = []
comp_true_as = []
comp_true_rev_as = []
comp_true_false_as = []
comp_false_true_as = []
```

```
comp_false_as = []
comp_false_rev_as = []

for qa in qa_set:
    q = qa[0]
    a = qa[1]

    simple_qs.extend(q['simple'])
    comp_true_qs.extend(q['comp_true'])
    comp_true_rev_qs.extend(q['comp_true_rev'])
    comp_true_false_qs.extend(q['comp_true_false'])
    comp_false_true_qs.extend(q['comp_false_true'])
    comp_false_qs.extend(q['comp_false'])
    comp_false_rev_qs.extend(q['comp_false_rev'])

    simple_as.extend(a['simple'])
    comp_true_as.extend(a['comp_true'])
    comp_true_rev_as.extend(a['comp_true_rev'])
    comp_true_false_as.extend(a['comp_true_false'])
    comp_false_true_as.extend(a['comp_false_true'])
    comp_false_as.extend(a['comp_false'])
    comp_false_rev_as.extend(a['comp_false_rev'])

print('Make "all" sets')

comp_qs = []
comp_qs.extend(comp_true_qs)
```

```
comp_qs.extend(comp_true_rev_qs)
comp_qs.extend(comp_true_false_qs)
comp_qs.extend(comp_false_true_qs)
comp_qs.extend(comp_false_qs)
comp_qs.extend(comp_false_rev_qs)

comp_as = []
comp_as.extend(comp_true_as)
comp_as.extend(comp_true_rev_as)
comp_as.extend(comp_true_false_as)
comp_as.extend(comp_false_true_as)
comp_as.extend(comp_false_as)
comp_as.extend(comp_false_rev_as)

all_questions = []
all_questions.extend(simple_qs)
all_questions.extend(comp_qs)

all_answers = []
all_answers.extend(simple_as)
all_answers.extend(comp_as)

print('Output question sets')

output_dir = './all_temp_qs/'

with open(output_dir + 'all_questions.json', 'w') as f:
```

```
    json.dump(all_questions, f)
with open(output_dir + 'all_answers.json', 'w') as f:
    json.dump(all_answers, f)

with open(output_dir + 'simple_questions.json', 'w') as f:
    json.dump(simple_qs, f)
with open(output_dir + 'simple_answers.json', 'w') as f:
    json.dump(simple_as, f)

with open(output_dir + 'comp_questions.json', 'w') as f:
    json.dump(comp_qs, f)
with open(output_dir + 'comp_answers.json', 'w') as f:
    json.dump(comp_as, f)

with open(output_dir + 'comp_true_questions.json', 'w') as f:
    json.dump(comp_true_qs, f)
with open(output_dir + 'comp_true_answers.json', 'w') as f:
    json.dump(comp_true_as, f)

with open(output_dir + 'comp_true_rev_questions.json', 'w') as f:
    json.dump(comp_true_rev_qs, f)
with open(output_dir + 'comp_true_rev_answers.json', 'w') as f:
    json.dump(comp_true_rev_as, f)

with open(output_dir + 'comp_false_questions.json', 'w') as f:
    json.dump(comp_false_qs, f)
with open(output_dir + 'comp_false_answers.json', 'w') as f:
```

```
    json.dump(comp_false_as, f)

with open(output_dir + 'comp_false_rev_questions.json', 'w') as f:
    json.dump(comp_false_rev_qs, f)
with open(output_dir + 'comp_false_rev_answers.json', 'w') as f:
    json.dump(comp_false_rev_as, f)

with open(output_dir + 'comp_true_false_questions.json', 'w') as f:
    json.dump(comp_true_false_qs, f)
with open(output_dir + 'comp_true_false_answers.json', 'w') as f:
    json.dump(comp_true_false_as, f)

with open(output_dir + 'comp_false_true_questions.json', 'w') as f:
    json.dump(comp_false_true_qs, f)
with open(output_dir + 'comp_false_true_answers.json', 'w') as f:
    json.dump(comp_false_true_as, f)

print('Done')

print(len(simple_qs))
print(len(comp_qs))

# Number of yes and no questions
sum_yes = 0
sum_no = 0
for a in all_answers:
    if a['labels'][0] == 425: # yes
```

```
        sum_yes += 1
    else:
        sum_no += 1

print('Number yes: %d' % sum_yes)
print('Number no: %d' % sum_no)
print('Percentage of yes answers: %f' % (sum_yes /
        len(all_questions) * 100))

import json

# ids_being_used
data_dir = '/n/fs/visualai-scr/gmccord/composition/data/'
with open(data_dir + 'image_data.json', 'r') as f:
    image_data = json.load(f)

# ids_being_used
with open(data_dir + 'lxmert_val_img_id.json', 'r') as f:
    img_ids = json.load(f)
d_img_ids = set()
for k in img_ids:
    d_img_ids.add(k)

vg_ids = set()
for d in image_data:
    if d['coco_id'] in d_img_ids:
        vg_ids.add(d['image_id'])
print(len(vg_ids))
```

```
print('d_img_ids')

input_dir = './all_temp_qs/'
output_dir = './lxmert_all_temp_qs/'

# Input
with open(input_dir + 'all_questions.json', 'r') as f:
    all_questions = json.load(f)
with open(input_dir + 'all_answers.json', 'r') as f:
    all_answers = json.load(f)

with open(input_dir + 'simple_questions.json', 'r') as f:
    simple_qs = json.load(f)
with open(input_dir + 'simple_answers.json', 'r') as f:
    simple_as = json.load(f)

with open(input_dir + 'comp_questions.json', 'r') as f:
    comp_qs = json.load(f)
with open(input_dir + 'comp_answers.json', 'r') as f:
    comp_as = json.load(f)

with open(input_dir + 'comp_true_questions.json', 'r') as f:
    comp_true_qs = json.load(f)
with open(input_dir + 'comp_true_answers.json', 'r') as f:
    comp_true_as = json.load(f)
```

```
with open(input_dir + 'comp_true_rev_questions.json', 'r') as f:
    comp_true_rev_qs = json.load(f)
with open(input_dir + 'comp_true_rev_answers.json', 'r') as f:
    comp_true_rev_as = json.load(f)

with open(input_dir + 'comp_false_questions.json', 'r') as f:
    comp_false_qs = json.load(f)
with open(input_dir + 'comp_false_answers.json', 'r') as f:
    comp_false_as = json.load(f)

with open(input_dir + 'comp_false_rev_questions.json', 'r') as f:
    comp_false_rev_qs = json.load(f)
with open(input_dir + 'comp_false_rev_answers.json', 'r') as f:
    comp_false_rev_as = json.load(f)

with open(input_dir + 'comp_true_false_questions.json', 'r') as f:
    comp_true_false_qs = json.load(f)
with open(input_dir + 'comp_true_false_answers.json', 'r') as f:
    comp_true_false_as = json.load(f)

with open(input_dir + 'comp_false_true_questions.json', 'r') as f:
    comp_false_true_qs = json.load(f)
with open(input_dir + 'comp_false_true_answers.json', 'r') as f:
    comp_false_true_as = json.load(f)

# Define function to convert the files
def convert_val_to_lxmert_val(l, img_ids=vg_ids):
```

```
    new_l = []

    for d in l:
        if int(d['image_id']) in img_ids:
            new_l.append(d)

    return new_l

print('Input received')

all_questions = convert_val_to_lxmert_val(all_questions)
all_answers = convert_val_to_lxmert_val(all_answers)

simple_qs = convert_val_to_lxmert_val(simple_qs)
simple_as = convert_val_to_lxmert_val(simple_as)

comp_qs = convert_val_to_lxmert_val(comp_qs)
comp_as = convert_val_to_lxmert_val(comp_as)

comp_true_qs = convert_val_to_lxmert_val(comp_true_qs)
comp_true_as = convert_val_to_lxmert_val(comp_true_as)

comp_true_rev_qs = convert_val_to_lxmert_val(comp_true_rev_qs)
comp_true_rev_as = convert_val_to_lxmert_val(comp_true_rev_as)

comp_false_qs = convert_val_to_lxmert_val(comp_false_qs)
comp_false_as = convert_val_to_lxmert_val(comp_false_as)
```

```
comp_false_rev_qs = convert_val_to_lxmert_val(comp_false_rev_qs)
comp_false_rev_as = convert_val_to_lxmert_val(comp_false_rev_as)

comp_true_false_qs = convert_val_to_lxmert_val(comp_true_false_qs)
comp_true_false_as = convert_val_to_lxmert_val(comp_true_false_as)

comp_false_true_qs = convert_val_to_lxmert_val(comp_false_true_qs)
comp_false_true_as = convert_val_to_lxmert_val(comp_false_true_as)

print('Beginning Output')

with open(output_dir + 'all_questions.json', 'w') as f:
    json.dump(all_questions, f)
with open(output_dir + 'all_answers.json', 'w') as f:
    json.dump(all_answers, f)

with open(output_dir + 'simple_questions.json', 'w') as f:
    json.dump(simple_qs, f)
with open(output_dir + 'simple_answers.json', 'w') as f:
    json.dump(simple_as, f)

with open(output_dir + 'comp_questions.json', 'w') as f:
    json.dump(comp_qs, f)
with open(output_dir + 'comp_answers.json', 'w') as f:
    json.dump(comp_as, f)
```

```
with open(output_dir + 'comp_true_questions.json', 'w') as f:
    json.dump(comp_true_qs, f)
with open(output_dir + 'comp_true_answers.json', 'w') as f:
    json.dump(comp_true_as, f)

with open(output_dir + 'comp_true_rev_questions.json', 'w') as f:
    json.dump(comp_true_rev_qs, f)
with open(output_dir + 'comp_true_rev_answers.json', 'w') as f:
    json.dump(comp_true_rev_as, f)

with open(output_dir + 'comp_false_questions.json', 'w') as f:
    json.dump(comp_false_qs, f)
with open(output_dir + 'comp_false_answers.json', 'w') as f:
    json.dump(comp_false_as, f)

with open(output_dir + 'comp_false_rev_questions.json', 'w') as f:
    json.dump(comp_false_rev_qs, f)
with open(output_dir + 'comp_false_rev_answers.json', 'w') as f:
    json.dump(comp_false_rev_as, f)

with open(output_dir + 'comp_true_false_questions.json', 'w') as f:
    json.dump(comp_true_false_qs, f)
with open(output_dir + 'comp_true_false_answers.json', 'w') as f:
    json.dump(comp_true_false_as, f)

with open(output_dir + 'comp_false_true_questions.json', 'w') as f:
    json.dump(comp_false_true_qs, f)
```

```
with open(output_dir + 'comp_false_true_answers.json', 'w') as f:
    json.dump(comp_false_true_as, f)


print('END')
```

Bibliography

[1] Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Spice: Semantic propositional image caption evaluation. In *European Conference on Computer Vision*, pages 382–398. Springer, 2016.

[2] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6077–6086, 2018.

[3] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[4] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python*. OReilly Media Inc., 2009.

[5] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In *Conference on fairness, accountability and transparency*, pages 77–91, 2018.

[6] Kaylee Burns, Lisa Anne Hendricks, Kate Saenko, Trevor Darrell, and Anna Rohrbach. Women also snowboard: Overcoming bias in captioning models. *arXiv preprint arXiv:1803.09797*, 2018.

[7] Arjun Chandrasekaran, Deshraj Yadav, Prithvijit Chattopadhyay, Viraj Prabhu, and Devi Parikh. It takes two to tango: Towards theory of ai's mind. *arXiv preprint arXiv:1704.00717*, 2017.

[8] Xinlei Chen, Hao Fang, Tsung-Yi Lin, Ramakrishna Vedantam, Saurabh Gupta, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco captions: Data collection and evaluation server. *arXiv preprint arXiv:1504.00325*, 2015.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[10] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José MF Moura, Devi Parikh, and Dhruv Batra. Visual dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 326–335, 2017.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Michael Denkowski and Alon Lavie. Meteor universal: Language specific translation evaluation for any target language. In *Proceedings of the ninth workshop on statistical machine translation*, pages 376–380, 2014.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[14] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn,

and A. Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015.

[15] Li Fei-Fei, Rob Fergus, and Pietro Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 conference on computer vision and pattern recognition workshop*, pages 178–178. IEEE, 2004.

[16] Christiane Fellbaum. Wordnet. *The encyclopedia of applied linguistics*, 2012.

[17] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural networks*, 1(2):119–130, 1988.

[18] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6904–6913, 2017.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Drew A Hudson and Christopher D Manning. Gqa: a new dataset for compositional question answering over real-world images. *arXiv preprint arXiv:1902.09506*, 2019.

[22] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[23] Yu Jiang, Vivek Natarajan, Xinlei Chen, Marcus Rohrbach, Dhruv Batra, and Devi Parikh. Pythia v0. 1: the winning entry to the vqa challenge 2018. *arXiv preprint arXiv:1807.09956*, 2018.

[24] Naveen Joshi. How machine vision can transform financial services, Sep 2019.

[25] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[29] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE, 2006.

[30] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] Michael Lewis. *An Army of One*, page 215–215. W.W. Norton & Company, Inc., 2015.

[32] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries acl. In *Proceedings of Workshop on Text Summarization Branches Out Post Conference Workshop of ACL*, pages 2017–05, 2004.

[33] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[35] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[36] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[37] Jiasen Lu, Dhruv Batra, Devi Parikh, and Stefan Lee. Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Advances in Neural Information Processing Systems*, pages 13–23, 2019.

[38] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[39] Aida Nematzadeh, Kaylee Burns, Erin Grant, Alison Gopnik, and Thomas L Griffiths. Evaluating theory of mind in question answering. *arXiv preprint arXiv:1808.09352*, 2018.

[40] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[42] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[43] Arijit Ray, Karan Sikka, Ajay Divakaran, Stefan Lee, and Giedrius Burachas. Sunny and dark outside?! improving answer consistency in vqa through entailed question generation. *arXiv preprint arXiv:1909.04696*, 2019.

[44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[45] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[46] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[47] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016.

[48] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[49] Alon Talmor, Yanai Elazar, Yoav Goldberg, and Jonathan Berant. olmpics–on what language model pre-training captures. *arXiv preprint arXiv:1912.13283*, 2019.

[50] Hao Tan and Mohit Bansal. Lxmert: Learning cross-modality encoder representations from transformers. *arXiv preprint arXiv:1908.07490*, 2019.

[51] Damien Teney, Peter Anderson, Xiaodong He, and Anton Van Den Hengel. Tips and tricks for visual question answering: Learnings from the 2017 challenge. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4223–4232, 2018.

[52] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[53] Ramakrishna Vedantam, C Lawrence Zitnick, and Devi Parikh. Cider: Consensus-based image description evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4566–4575, 2015.

[54] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[55] Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R Bowman. Blimp: A benchmark of linguistic minimal pairs for english. *arXiv preprint arXiv:1912.00582*, 2019.