Multimodal VQA with Amazon Berkeley Objects Dataset

Nitin Rajesh(MT2024103)
Akash Chaudhari(MT2024012)
Archis Kulkarni(MT2024082)
Github Link

Table of contents

1. Introduction

The assignment requires generating a multiple-choice VQA dataset using the ABO dataset. The first step is to select and prepare the relevant data from the ABO dataset. Once the data is selected, it must be converted into a format suitable for use in a VQA model.

The next step is to evaluate baseline models using standard metrics. This involves comparing the performance of different models on the given task. The goal of this step is to determine which baseline models are most effective and can serve as a starting point for further development.

After evaluating baseline models, the next step is to fine-tune these models using Low-Rank Adaptation (LoRA). LoRA is a technique used to adapt pre-trained models to new tasks or datasets. By applying LoRA to the baseline models, it is possible to improve their performance on specific tasks and adapt them to the ABO dataset.

Finally, the model must be assessed for its performance using standard metrics. This involves evaluating the model's ability to answer questions correctly and accurately. The goal of this step is to determine whether the fine-tuned model has achieved the desired level of performance and can be used in real-world applications.

In addition to these technical steps, VQA is also central to multimodal learning, which combines visual understanding with natural language reasoning. This approach helps develop AI systems that can interpret and respond to complex inputs across different data types, supporting real-world applications like assistive tech and interactive AI.

---

2. Dataset Curation

2.1 Dataset Source

Link to the Dataset
Processed Datasets - Google Drive

Overview
The dataset in question consists of multiple components, including:

- **Image Data**: A collection of 398,212 unique catalog images, each associated with a specific product listing.
- **Multilingual Metadata**: 147,702 product listings containing multilingual metadata, providing detailed information about the products.

Data Formats

- **Images**: The image data is stored in jpg that allows for efficient retrieval and display.
- **JSONL Files**: There are 147,702 JSONL files describing product details, including metadata such as titles, descriptions, and prices.
- **CSV File**: A CSV file containing image paths and image IDs, providing a mapping between the images and their corresponding product listings.

Key Characteristics

- Total number of product listings: 147,702
- Total number of unique catalog images: 398,212

2.2 Preprocessing

ExtractListings.py

Purpose

Extracts `.gz` compressed files from a specified directory, decompresses them, and saves the extracted data as uncompressed files.

Key Functions

- **Workflow**:
  1. Lists all files in the directory.
  2. Filters files ending with `.gz`.
  3. Decompresses each file using `gzip.open` and writes output to a new file.
  4. Prints confirmation for each extracted file.

GenerateCSV.py

Purpose

Processes JSON files from a directory, extracts structured data (`main_image_id` and `product_type` fields), and writes results to a CSV file.

Key Functions

- **Workflow**:
  1. Defines input directory and output CSV path.
  2. Scans directory for `.json` files.
  3. Parses JSON objects, filters rows with required keys, and extracts data.
  4. Writes data to a CSV file with headers.

MergeCSVFiles.py

Purpose

Merges two CSV files (image metadata and product data), performs data cleaning, and produces a final CSV with valid image-product associations.

Overall Architecture

1. **Input Processing**:
   - Reads `images.csv` (image metadata) and `filtered_products.csv` (product data).
2. **Data Transformation**:
   - Merges datasets on `image_id` and `main_image_id`.
   - Filters to retain columns: `image_id`, `path`, `product_type`.
3. **Validation**:
   - Constructs full image paths using a base directory.
4. **Output**:
   - Saves cleaned data to `merged_output.csv`.

DataAnalysis.py

Purpose

Performs exploratory data analysis (EDA) on the merged CSV file, providing insights into dataset structure and product type distribution.

Overall Architecture

1. **Input Processing**:
   - Loads `merged_output.csv` (output from `MergeCSVFiles.py`).
2. **Data Exploration**:
   - Checks for missing values.
   - Analyzes `product_type` distribution.
3. **Statistical Analysis**:
   - Calculates summary statistics for all columns.

2.3 Prompting & Generation

Script: `OllamaMoonDream.py`

Purpose

Generates descriptions for given images using the Moondream model from Ollama. The output consists of single line multi-sentence descriptions of an image.

It consists of objects, colours, state of the objects etc. The model is very small in size leading to faster inference times(average 6 sec per image).

Key Functions & Classes

- `ollama.Client()` : Connects to Ollama API
- `get_image_description()` : Generates VQA responses
- Output tracking via CSV file to avoid reprocessing

Architecture

1. **Initialization**
   - Sets parameters and loads data
2. **Model Integration**
   - Connects to Ollama API (`http://localhost:11434`)
   - Uses Moondream model for generating image descriptions
3. **Image Processing**
   - Validates image file presence
   - Sends prompt and image to model
4. **Output Management**
   - Appends results to `vqa_output.csv`
   - Tracks processed indices to prevent duplication

Dependencies

- `pandas` : CSV handling
- `ollama` : Python client for model interaction
- `tqdm` : Progress display
- `os` : File operations

Script: `OllamaL3.py`

Purpose

Generates question-answer (QA) pairs from image descriptions using the Llama3.2 model from Ollama. As the next step in the dataset creation, we pass the image descriptions generated from the Moondream model to Llama3.2 model along with metadata to generate 4 question answer pairs. Being an efficient text

model its results are more consistent and faster (average ~2 seconds per image) compared to image-text models like Llava.

Key Functions & Classes

- `pd.read_csv()` : Load CSV data
- `ollama.Client()` : Connect to Ollama API
- `generate_qa_from_description()` : Generate QA pairs
- `clean_and_parse_qa()` : Format and validate QA output
- `is_valid_qa()` : Validate QA pairs

Architecture

1. **Initialization**
   - Sets parameters (CSV paths, model name)
   - Loads data and selects subset
2. **Model Integration**
   - Connects to Ollama API (`http://localhost:11434`)
   - Uses Llama3.2 model for QA generation
3. **QA Generation**
   - Sends image descriptions to model
   - Extracts and cleans QA pairs
   - Validates and saves output

Data Flow

1. Load `vqa_output.csv`
2. Process entries from `START_INDEX` to `END_INDEX`
3. For each entry:
   - Send description to Llama3.2
   - Extract and clean QA pairs
   - Save to output CSV

2.4 Dataset Structure

The final schema of the dataset consists of several key components, including image_id, product_type, description, and four pairs of question and answers. The

product types served as classes in this model, with some classes being significantly more prevalent than others.

Upon closer examination, it was discovered that the product types were skewed and imbalanced, resulting in a substantial disparity between the number of observations for each class. Specifically, after merging the dataset, phones cases made up approximately 50% of the total observations, totaling around 63,000 entries. In contrast, the next most prominent class consisted of only 5,000 samples.

The questions posed to the model were diverse in nature, covering a range of topics such as object colors, product types, material, design, objects visible in the images, placement of objects, count of various things, and text visible. Notably, color-related questions were more prominent among these queries.

---

3. Model Choices

3.1 Baseline Models

Several pre-trained models have been evaluated for their performance in various tasks. Specifically, three models, blip-vqa-base and Blip2-Flan-T5-XL and VILT were tested as base models using the transformers framework.

It was observed that the blip-vqa-base model has shown promising results with a small but compatible accuracy level. This suggests that the model is capable of achieving accurate results while maintaining a reasonable level of complexity.

On the other hand, the flan model was found to be too large and resulted in low accuracy levels. Additionally, the vilt model's vocabulary size was limited, which may have contributed to its lower performance.

3.2 Fine-Tuned Models

Blip-VQA-Base was chosen for further LoRA (Low-Rank Adaptation) training. This decision was made to leverage the capabilities of Blip-VQA-Base in adapting to new data distributions, which is essential for improving the performance of the model on various tasks.

To accommodate the increased computational requirements associated with LoRA training, higher amounts of GPU memory was required. But GPU memory posed a constraint with 30 GB maximum capacity, taking advantage of the Kaggle and Colab services. Furthermore, trainable parameters were constrained to 1,179,648, as excessive parameter values have been shown to increase training time without guaranteeing proportional improvement in the model performance.

The LoRA training process involved training on a dataset consisting of 196x196 images, resulting in one-worded answers. Notably, the textual part of the model was trained exclusively during this phase, while the image component remained untouched. This selective approach allowed for focused adaptation of the model to the specific task at hand, enabling more efficient and effective learning.

---

4. Fine-Tuning Approach

4.1 Objective

Fine-tuning a low-resource language model for Visual Question Answering (VQA) on product images is crucial for producing one-worded answers. This process involves fine-tuning a pre-trained base model on a VQA dataset, which typically consists of image-question pairs.

To succeed in VQA tasks, the model must recognize objects and comprehend visual context. Fine-tuning the VQA model enables it to generate accurate answers by capturing product-related vocabulary and syntax nuances along with images. This can be achieved by incorporating product-specific datasets or using domain-adaptive techniques that adapt to product images' characteristics.

4.2 LoRA Configuration

The LoraConfig object is defined with the following:

```
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=["query", "value"],
    lora_dropout=0.1,
    bias="none",
```

```
    task_type=TaskType.QUESTION_ANS,
)
```

This configuration specifies that the Lora (Low-Rank Adaptation) technique will be applied to two specific modules: query and value. The r parameter represents the rank of the adaptation, which is set to 8 in this case. The lora_alpha parameter controls the strength of the adaptation, with a higher value indicating stronger adaptation. The target_modules parameter specifies that only the query and value modules should be updated during fine-tuning.

Parameters in the LoRA were as follows:

```
trainable params: 1,179,648
all params: 385,852,220
trainable%: 0.3057
```

Layers trained
The Lora technique is applied to two types of layers: text encoder and text decoder. In both cases, the LoRA adapters are injected into all 12 layers of the self-attention and cross-attention blocks. Specifically, the LoRA adapters wrap the following projections:

1. Text Encoder (text_encoder)
   Injected in all 12 layers of the self-attention and cross-attention blocks.
   Specifically, the LoRA adapters wrap:
   query projection: lora.Linear
   value projection: lora.Linear
2. Text Decoder (text_decoder)
   Same pattern as text_encoder, in all 12 layers:
   query: wrapped with lora.Linear
   value: wrapped with lora.Linear

These are the only layers updated/trained during LoRA fine-tuning. All other layers remain frozen. These adaptations are applied to the query and value modules of the text encoder and text decoder, respectively.

Layers names trained under lora:

```
text_encoder.encoder.layer.[0-11].attention.self.query
text_encoder.encoder.layer.[0-11].attention.self.value
text_encoder.encoder.layer.[0-11].crossattention.self.query
text_encoder.encoder.layer.[0-11].crossattention.self.value

text_decoder.bert.encoder.layer.[0-11].attention.self.query
text_decoder.bert.encoder.layer.[0-11].attention.self.value
text_decoder.bert.encoder.layer.[0-11].crossattention.self.query
text_decoder.bert.encoder.layer.[0-11].crossattention.self.value
```

About the Base Model:
The BLIP-VQA-Base model is a visual question answering model that leverages the BLIP framework, utilizing a Vision Transformer (ViT-B) as its image encoder for efficient visual data processing.

It employs a multilayered approach to training objectives, including Image-Text Contrastive Loss (ITC), Image-Text Matching Loss (ITM), and Language Modeling Loss (LM).

The model has a parameter count of approximately 385 million and is specifically designed for fine-tuned visual question answering (VQA), demonstrating exceptional performance in various benchmarks.

4.3 Training Strategy

To optimize our model's performance, we employed the following training parameters:

We conducted our training for a total of 10 epochs, with each epoch consisting of 16 batches per device. Additionally, we utilized a constant learning rate of 10^{-4} to regulate the model's parameter updates.

Our dataset consisted of approximately 28,000 images with each having ~4 question answer pairs. This was split into three categories: training ~20,000, validation ~1,000, and testing ~8,000. This division allowed us to monitor the model's performance on both the training and validation sets during the learning process, as well as evaluate its generalization capabilities on unseen test data.

For efficient computation, we leveraged local GPU resources for training. Specifically, to train our LoRA model and for inference tasks, we relied on Kaggle's

2 x T4 GPUs.

---

5. Evaluation Metrics

5.1 Metrics Used

In the context of Visual Question Answering (VQA) tasks in Artificial Intelligence, the metrics mentioned - Accuracy, F1 Score, and BERTScore - are used to evaluate the performance of a model's ability to answer questions based on visual inputs.

Accuracy: This metric measures the proportion of correct predictions made by the model out of all total predictions. In VQA, accuracy is typically calculated as the number of correctly answered questions divided by the total number of questions asked.

F1 Score: The F1 Score (also known as the F-score) is a measure of both precision and recall. The F1 Score is calculated by taking the harmonic mean of precision and recall. In VQA, an F1 Score of 0.8 or higher indicates that the model is able to accurately identify both relevant and irrelevant information in the visual input.

BERT Score: BERT Score is a metric specifically designed for Visual Question Answering tasks. It measures the similarity between the predicted answer and the correct answer based on the BERT model's output embeddings. The BERT Score formula calculates the cosine similarity between the two vectors, which ranges from -1 (completely dissimilar) to 1 (identical). A higher BERT Score indicates that the predicted answer is more similar to the correct answer.

The formula for calculating BERTScore is as follows:

BERTScore = cos(similarity)

where similarity is the cosine similarity between the predicted answer and the correct answer embeddings.

5.2 Metric Interpretation

Accuracy
Strengths:

- Easy to understand and interpret: Accuracy measures how well a model performs on a given task.
  Limitations:
- Ignores nuances of language: Accuracy focuses solely on correct predictions, without considering context, tone, or nuance.

F1 Score
Strengths:

- Balances precision and recall: F1 Score provides a balanced measure of both precision and recall, offering a comprehensive evaluation of a model's performance.
  Limitations:
- Can be sensitive to class imbalance: If the classes are imbalanced, F1 Score may not accurately reflect the model's performance on the majority class.

BERTScore
Strengths:

- Evaluates semantic similarity: BERTScore measures the similarity between predicted and correct answers, providing insight into a model's understanding of semantic relationships.
  Limitations:
- Requires specialized knowledge: BERTScore requires an understanding of the BERT model and its output embeddings.

Why BERTScore is essential for evaluating semantic similarity

BERTScore is essential for evaluating semantic similarity because it:

1. **Captures nuanced language patterns**: By leveraging the transformer architecture and pre-trained BERT model, BERTScore can capture subtle patterns in language that may not be apparent through other metrics.
2. **Provides a more comprehensive evaluation**: BERTScore offers a more detailed understanding of a model's performance on semantic similarity tasks, which is critical for evaluating the effectiveness of models in capturing nuanced relationships between entities or concepts.

3. **Aligns with task requirements**: For many NLP tasks, including Visual Question Answering, semantic similarity is a crucial aspect of evaluation. BERTScore provides a standardized metric that can be used to evaluate these tasks and identify areas for improvement.

5.3 Quantitative Results

Model	Accuracy	F1 Score	BERTScore
Blip2-Flan-T5-XL	0.1217	0.1434	0.8975
Vilt B32	0.2695	0.2703	0.9843
Blip-vqa-base (Pretrained)	0.2183	0.2252	0.9527
Lora Blip-vqa-base (Over Test Data)	0.5118	0.5184	0.9747

Following were the observations:

- Blip-vqa-base was our baseline model. After Lora finetuning, we can see a 2.3x improvement in F1 score, indicating significant improvements for our VQA task for the dataset.
- All models have high Bert similarity score. This signifies the answers are semantically similar but does not necessarily have to be same
- The smallest model, Vilt-B32 has the highest Bert similarity score. However, nearly no improvements were obtained through finetuning.
- Blip2-flan-t5 has the lowest performance despite being the largest model by far. This may be because it is a generic model which has not been adapted for VQA tasks specifically.

Overall, the choice of model has helped us obtain good improvements through finetuning, by balancing model complexity and VQA optimization.

5.4 Issues Faced

The performance of the Vilt model was compromised by its inability to accurately predict meaningful content. This issue may be attributed to the limited size of the vocabulary, which hindered the model's capacity to generalize and produce sensible outputs.

Furthermore, the Llava model proved ineffective in generating question-answer pairs for one-shot image-based queries.

However, the Llama model faced significant challenges in producing questions for certain descriptions. A repeat mechanism had to be implemented to retry generating question answers if it failed. This mechanism helped reducing the failure rate drastically.

In addition, training smaller batches of data for the BLIP model resulted in overfitting. For instance, using 25% of the available data for training led to the model becoming overly specialized and failing to generalize well to unseen data.

Lastly, the dataset used for training the models was found to be heavily skewed towards a single class, eg. phone cases constituting approximately 50% of the total data. This imbalance in the dataset can lead to biased performance and may hinder the model's ability to generalize across different classes or scenarios.

---

6. Contributions and Novelty

The development of an automated VQA dataset generation pipeline was a crucial step in this project.

A dual-stage prompting approach was employed, combining Moondream's image expertise with Llama's text proficiency. This combination enabled the creation of high-quality dataset catering to both image-based and text-based questions and much faster rate and precision.

To ensure fairness across classes, a balanced sampling strategy was implemented. The approach involved selecting an equal amount of images from each class and then applying balanced sampling from the remaining, larger classes. This hybrid method allowed for sufficient data samples while maintaining a fair balance between classes, thereby enhancing the overall quality of the generated dataset.

Robust incremental data generation and in-place retry mechanisms were introduced to mitigate the impact of failed question answer generation attempts. These measures drastically reduced the number of missed QA pairs by minimizing the need for additional rounds of generation. Furthermore, failed samples were negligible, ensuring that the pipeline's performance was not significantly compromised.

Finally, customized evaluation and analysis tools were developed to facilitate parallel training and inference across multiple devices. This enabled efficient processing and analysis of the generated dataset, taking advantage of split work and device availability.

---

7. Conclusion

The project aimed to create an efficient dataset and evaluate various base models to determine the most suitable one. Following this, a LORA (Large Language Observation) model was trained, which demonstrated enhanced results compared to the base models.

However, two other models, Blip2-Flan-T5-XL and Vilt, were found to be less effective due to their size and limitations. The Blip2-Flan-T5-XL model was deemed too large, resulting in inconsistent and promising results that did not meet expectations. In contrast, the Vilt model's vocabulary was limited, leading to wildly different and unrelated results.

On the other hand, the Blip-VQA-Base model, although smaller than the Blip2-Flan-T5-XL model, proved to be more stable and produced better results. The addition of LORA to this model further improved its accuracy and F1-score.

Despite these achievements, there are limitations and potential areas for improvement that need to be addressed. Firstly, using better models for dataset creation could lead to more accurate and reliable results. Additionally, increasing the size of the base model could potentially improve performance. Training the LORA model for a greater number of epochs may also yield better results.

Furthermore, utilizing more efficient hardware could help optimize training times and improve overall performance. Developing smaller and more efficient models could also be beneficial. Exploring different learning approaches, such as DINO, and conducting hyperparameter tuning could provide additional avenues for improvement.

---

8. References

- moondream model
- llama3.2
- Issue
- 🚀 Fine-Tuning BLIP-2 with LoRA on the Flickr8k Dataset for Image Captioning | by Rana Adeel Tahir | May, 2025 | Medium
- Models - Hugging Face
- Salesforce/blip-vqa-base · Hugging Face
- dandelin/vilt-b32-mlm · Hugging Face
- Salesforce/blip2-flan-t5-xl · Hugging Face