VR Mini-Project 2: Multimodal Visual Question Answering using ABO Dataset

Vasu Aggarwal	IMT2022073
Ananthakrishna K	IMT2022086
Anurag Ramaswamy	IMT2022103

May 18, 2025

1 Introduction

In this mini-project, we create a Visual Question Answering (VQA) dataset using the Amazon Berkeley Objects (ABO) dataset, evaluating baseline models such as BLIP (Bootstrapping Language-Image Pre-training) and ViLT (Vision-and-Language Transformer), perform finetuning using Low-Rank Adaptation (LoRA), and assess the performance of all models using different evaluation metrics.

2 Dataset

2.1 Data Overview

We are using the small variant of the Amazon Berkeley Objects Dataset, which contains around 150,000 Amazon product listings with multilingual metadata and around 400,000 unique catalog images of items on a white background.

- **Metadata Files:** `listings_*.json` from `../dataset/listings/metadata/`, containing product information.
- **Image Metadata CSV:** `images.csv` from `../dataset/`, providing data such as `image_id` and `path` for each image.
- **Image Files:** Actual product images stored in `../dataset/small/`.

2.2 Metadata Structure and Preprocessing

JSON file in the metadata directory contains many products. Each line is a separate product and has information like the product's name, category, color and keywords.

- **item_name:** A list that contains the product's name.
- **product_type:** This field provides the product's category or type.
- **color:** The key describing the primary colors associated with the item.
- **language_tag:** This is to indicate the language of the corresponding value. Only English-language entries have been included.
- **main_image_id:** A key field used to link the listing metadata to the corresponding image. Only records with a valid `main_image_id` are stored for use.

Preprocessing

- We get product details like name, category, color, and keywords from the metadata.
- Only English entries are kept for simplicity.
- Duplicate values are removed.
- Values are joined into clean, easy-to-use text strings.
- We have drop the entries having incomplete features.

2.3 Image Metadata and Merging:

The `images.csv` file contains information about each image, such as its ID, file path, height, and width. We removed the height and width columns.

We combined the image data with product details from the metadata. We added the product's name, category, color, and keywords using function `progress_apply`.

2.4 Data Cleaning

Data cleaning ensures that the dataset is accurate and useful. In this process:

- Only text fields containing English letters (ASCII characters) are kept.
- Rows with missing or empty important information are removed.

3 Data Curation

We used the Google Gemini API (gemini-1.5-flash) to generate visual question-answer pairs from product images and their metadata. The API was configured with custom parameters like temperature and token limits to balance creativity and control in the outputs.

3.1 Prompt Design and Question Types

The prompt we used is given below:

```
    prompt_parts = [
        "Context about the image, make sure to also look at the image and analyse
             it before generating :",
        f"- Name: {metadata_without_brand.get('name', 'N/A')}",
        f"- Category: {metadata_without_brand.get('category', 'N/A')}",
        f"- Main Color Provided: {metadata_without_brand.get('color', 'N/A')}",
        f"- Keywords: {metadata_without_brand.get('keywords', 'N/A')}\n",
        "Instructions:",
        "1. Analyze the provided image and the context.",
        "2. Generate exactly 5 (five) distinct questions about prominent visual
             features, objects, colors, materials, or attributes clearly visible
             in the image.",
        "3. Each question MUST have a single-word answer directly verifiable from
             the image.",
        "4. The 5 questions generated MUST be different from each other, and make
             sure that the questions are answerable just by looking at the image
             for exmaple do not ask questions like brand, if it there in the
             metadata but not in the image",
        "5. Provide the output strictly in the following numbered format, with
             each question and answer pair clearly marked. Do not include any
             other text before or after this numbered list:\n",
        "6. Remember to make the questions easy and answerable just by loking at
             the image, like color shape etc",
        """1.
Question 1: [Your first question here]
Answer 1: [Your single-word answer here]
2.
Question 2: [Your second question here]
Answer 2: [Your single-word answer here]
3.
Question 3: [Your third question here]
Answer 3: [Your single-word answer here]
4.
Question 4: [Your fourth question here]
Answer 4: [Your single-word answer here]
5.
Question 5: [Your fifth question here]
Answer 5: [Your single-word answer here]""",
        "\nImage:",
        img,
    ]
```

Listing 1: Prompt

These questions are intended to be answerable solely through visual inspection (e.g., color, shape, texture), with each answer being a single word. This design ensures that the dataset is suitable for training and evaluating image-understanding models.

3.2 QA Dataset Structure

The generated question-answer dataset is stored in CSV format. Each row in the CSV represents a single question derived from an image. The structure of the CSV includes the following columns:

- **Image_id**: A unique identifier or filename of the image associated with the question.
- **Question**: The question text, usually focused on visual attributes (e.g., color, shape, texture).
- **Answer**: Consist of correct answer of each given image question pair.

This format allows easy parsing and is compatible with training pipelines for classification-based Visual Question Answering (VQA) models.

4 Evaluation Metrics

4.1 Exact Match Accuracy

Exact Match Accuracy is the fraction of predictions by the model that exactly match the ground-truths. It is often used in question answering tasks where exact string matching with ground truths is required. In our mini-project, it signifies how often the model's prediction exactly matches the correct label. It is given by the formula:

$$\text{Accuracy} = \frac{\text{Number of exactly correct predictions}}{\text{Total number of examples}}$$

4.2 Average BERTScore F1

BERTScore evaluates semantic similarity by comparing contextual embeddings of tokens from the ground truths and predictions, generated using a pre-trained BERT model (in our case, `bert-base-uncased`). The F1 score is then derived by computing the harmonic mean of precision and recall based on cosine similarity between token embeddings. This metric allows us to capture semantic similarity between ground truths and predictions even when the exact overlap between them is minimal.

4.3 Average BLEU-1 Score

It measures how many words in the prediction exactly match the words in the ground truth. It is calculated using the `sentence_bleu` function from NLTK with smoothing (Chen & Cherry method 1) to avoid zero scores for short sequences. This method captures exact similarity at the word level. This metric is not that useful in our case since it compares a sequence of words and we are working on single word predictions.

4.4 Average ROUGE-L (F1)

ROUGE-L computes the longest common subsequence (LCS) between the prediction and the ground truth. The F1 variant combines both precision and recall of the LCS, making it useful for evaluating any partial matches. We used the `rouge-score` Python library with stemming enabled, which reduces words to their base form (running becomes run), allowing different word forms with the same meaning to be treated the same during evaluation. This metric is not that useful in our case since it compares a sequence of words and we are working on single word predictions.

4.5 Average METEOR Score

METEOR checks how similar a predicted answer is to the correct answer by matching words, even if they are in different forms (using stemming), have similar meanings or appear in a different order. Unlike BLEU, it also gives penalties if the word order is very different. This metric works well in cases where exact matches are not required but the meaning and structure of the answer are important.

4.6 Average CIDEr-D Score

CIDEr-D measures how well a predicted answer matches the ground truth by comparing groups of words (n-grams) and giving more importance to rare ones. We used the `pycocoevalcap` library to compute this score by organizing predictions and answers into dictionaries and this metric is especially useful when there are multiple correct answers to a single question.

4.7 Normalized Levenshtein Similarity

Normalized Levenshtein Similarity evaluates string similarity based on the number of edits (insertions, deletions, substitutions) required to convert the predicted text into the ground truth. This edit distance is then normalized by dividing by the length of the longer string to get the value between 0 (completely different) and 1 (identical). This metric is useful for capturing similarity between almost same answers and minor typographical errors in short text predictions.

5 Baseline Evaluation

To establish a performance baseline, we evaluated two pre-trained Vision-Language models on our curated Visual Question Answering (VQA) dataset without any finetuning:

- **BLIP (Salesforce/blip-vqa-base):** A Transformer-based VQA model combining visual and textual encoders.

- **ViLT (dandelin/vilt-b32-finetuned-vqa):** A vision and language transformer that directly processes image patches and text tokens without a separate image encoder.

We chose BLIP and ViLT because they are popular pre-trained VQA models. BLIP is good at understanding both images and text, while ViLT is lighter and faster, making it useful for quick testing. So, they gave us a solid baseline to compare our finetuned models against.

5.1 Baseline Evaluation Results

To assess model performance, we used the following evaluation metrics which are explained in detail in the Evaluation Metrics section :
Baseline BLIP (Salesforce/blip-vqa-base)

- **Exact Match (EM) Score:** 8.06%
- **Average BERTScore F1:** 0.6344
- **Average BLEU-1 Score:** 0.0817
- **Average ROUGE-L (F1):** 0.0871
- **Average METEOR Score:** 0.0476
- **Average CIDEr-D Score:** 0.2048
- **Normalized Levenshtein Similarity:** 0.2326

Baseline ViLT (dandelin/vilt-b32-finetuned-vqa)

- **Exact Match (EM) Score:** 3.76%
- **Average BERTScore F1:** 0.6271
- **Average BLEU-1 Score:** 0.0378
- **Average ROUGE-L (F1):** 0.0414
- **Average METEOR Score:** 0.0228
- **Average CIDEr-D Score:** 0.0946
- **Normalized Levenshtein Similarity:** 0.1675

These results provide a starting point to compare with our LoRA finetuned models in the next section.

6 Finetuning with LoRA

To finetune the performance of our multimodal Visual Question Answering (VQA) model, we used Low-Rank Adaptation (LoRA), a parameter-efficient finetuning approach. LoRA introduces a small number of trainable parameters without modifying the large base model, which helps in increasing performance. We finetuned only the BLIP model(Salesforce/blip-vqa-base) throughout all LoRA experiments.

6.1 Model Configurations

All the models were ran for 3 epochs each to ensure uniformity in results. Three configurations of LoRA were finetuned and evaluated:

- **Model 1 (Best Model):** LoRA rank $r = 16$, $alpha = 32$
- **Model 2:** LoRA rank $r = 16$, $alpha = 16$, with no dense layer added.
- **Model 3:** LoRA rank $r = 8$, $alpha = 8$, a lighter configuration for reduced compute usage.

Among these, **Model 1** had the highest performance across all evaluation metrics and was used as the final model in the inference file.

6.2 Training Strategy

Due to frequent crashes of Kaggle kernels, we avoided using `transformers.TrainingArguments` and opted for a manual training loop. This approach provided the following benefits:

- **Save model checkpoints** after every epoch to prevent loss of progress.
- **Implement early stopping** based on validation loss values, reducing unnecessary training time. Early stopping was triggered when validation loss failed to improve over consecutive epochs.

6.3 Evaluation Metrics

To assess model performance, we used the following evaluation metrics to compare with the baseline models :

Performance of Best Model (Model 1):

- **Exact Match (EM) Score:** 31.80%
- **Average BERTScore F1:** 0.7410
- **Average BLEU-1 Score:** 0.3180
- **Average ROUGE-L (F1):** 0.3270

- **Average METEOR Score:** 0.1673
- **Average CIDEr-D Score:** 0.7951
- **Normalized Levenshtein Similarity:** 0.4352

Performance of Model 2:

- **Exact Match (EM) Score:** 29.88%
- **Average BERTScore F1:** 0.7368
- **Average BLEU-1 Score:** 0.2988
- **Average ROUGE-L (F1):** 0.3074
- **Average METEOR Score:** 0.1584
- **Average CIDEr-D Score:** 0.7470
- **Normalized Levenshtein Similarity:** 0.4197

Performance of Model 3:

- **Exact Match (EM) Score:** 26.75%
- **Average BERTScore F1:** 0.7265
- **Average BLEU-1 Score:** 0.2675
- **Average ROUGE-L (F1):** 0.2764
- **Average METEOR Score:** 0.1423
- **Average CIDEr-D Score:** 0.6688
- **Normalized Levenshtein Similarity:** 0.3907

6.4 Observations

- In the best model (Model 1), LoRA finetuning led to a substantial improvement in performance compared to the baseline model. The **Exact Match Accuracy** increased from **8.06%** to **31.80%**, a relative gain of **294.5%**. The **BERTScore F1** improved from **0.6344** to **0.7410** (**+16.8%**), and other metrics showed similar upward trends:
  - **Average BLEU-1 Score:** Increased from 0.0817 to 0.3180 (**+289.2%**)
  - **Average ROUGE-L (F1):** Increased from 0.0871 to 0.3270 (**+275.4%**)
  - **Average METEOR Score:** Increased from 0.0476 to 0.1673 (**+251.5%**)
  - **Average CIDEr-D Score:** Increased from 0.2048 to 0.7951 (**+288.2%**)
  - **Normalized Levenshtein Similarity:** Increased from 0.2326 to 0.4352 (**+87.1%**)

These results clearly demonstrate the effectiveness of our LoRA finetuning.

- Increasing LoRA rank and alpha (Model 1) provided better convergence and higher semantic accuracy.
- Model 2 showed lower performance due to the absence of a dense layer.
- Model 3 underperformed due to reduced capacity and potential underfitting.

6.5 Challenges Faced

We faced many issues during finetuning:

- **Long training time:** Each epoch required approximately 1 to 1.5 hours to complete, so we could not run many models because of time and resource constraints on Kaggle which had a weekly limit of 30 hours.
- **Kernel instability:** The Kaggle kernel frequently crashed during training, particularly during longer epochs or GPU-intensive steps.

Due to the above challenges, we decided to implement manual training and checkpointing strategies to maintain progress across epochs.

7 Experiments

We found that the quality of generated questions was very sensitive to how the prompt was written. Our initial simple prompts led to around 50,000 easy Q&A pairs (qna.csv)—mostly straightforward questions like color or object type that didn't require much reasoning. To fix this, we experimented with more detailed prompts and generated a second set of around 25,000 moderate to hard Q&A pairs (qna_2.csv) that focused on finer details.

Due to limited time and compute resources, we couldn't do a lot of hyperparameter tuning. Our best results came from LoRA with rank r=16 , alpha=32, Learning Rate=5e-5 and AdamW optimizer which gave significant improvements across all evaluation metrics as seen above.

8 Repository

- GitHub Link: `https://github.com/vr-assignments/VR_MiniProject2_IMT2022_073_086_103`
- Contains all the datasets(including the curated dataset qna_2.csv) and notebooks used for data curation, finetuning and evaluation.
- Contains the inference script in the required format along with a ReadMe with instructions on how to run the code.

9 Individual Contributions

IMT2022073 Vasu Aggarwal: Wrote code for evaluating the predictions from models(including baseline) and helped in data curation.

IMT2022086 Ananthakrishna K: Wrote code for data curation, generating the VQA dataset and the code for inference.

IMT2022103 Anurag Ramaswamy: Wrote code for finetuning the BLIP models and helped in generating the VQA dataset.

All the remaining tasks including documentation were done by all 3 members equally.