Visual Recognition - Final Project

Adithya Nagaraja Somasalle
IMT2021054
Adithya.Somasalle@iiitb.ac.in

Achintya Harsha
IMT2021525
Achintya.Harsha@iiitb.ac.in

Akash Perla
IMT2021530
Akash.Perla@iiitb.ac.in

I. Objective

- To build a model for Visual Question Answering which takes as input an image and a corresponding question and gives an answer to the question.
- Fine-tune pre-trained models with and without the help of LoRA and compare the training time and performance of the fine-tuned models.

II. Dataset

For this project, the dataset used was VQA Dataset released by Georgia Tech. Comprising images sourced from the COCO dataset [3], VQA is annotated with human-generated questions and multiple corresponding answers, fostering the development of models capable of understanding and responding to queries about visual content. The dataset includes over 760,000 questions with around 10 million answers, along with over 200,000 real images and 50,000 abstract scenes. The abstract scenes were specifically designed to test a model's ability to reason without relying on the ability to parse complex images.

III. Tasks

- Create a model that takes as an input an image and a corresponding question and gives an answer to the question.
- First, fine-tune the model you created on the dataset above. Note down the time taken to train this model along with the evaluation metrics.
- Use LoRA to fine-tune the same model and again note down the time taken and the metrics

A. BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a powerful technique for natural language processing (NLP) tasks. Unlike traditional models that read text sequentially, BERT uses a bidirectional approach, meaning it processes text in both directions (left-to-right and right-to-left) simultaneously. This allows BERT to capture a deeper understanding of context and nuances in language.

Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token [4]. Figure 1 illustrates the model architecture

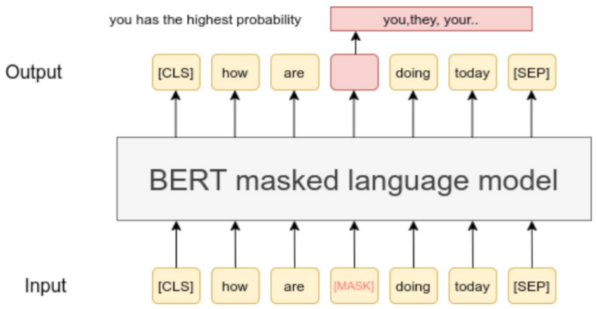

Fig. 1: BERT Architecture. [5]

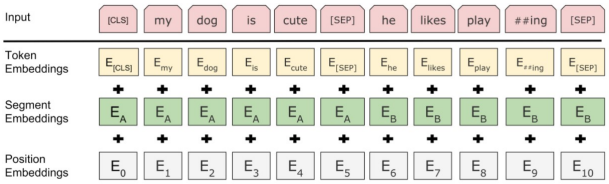

Fig. 2: BERT input representation. [4]

B. VIT

VIT, which stands for Vision Transformer, breaks down an image into patches and feeds them directly into a transformer encoder, similar to the one used in BERT for text. By applying self-attention mechanisms, ViT effectively models the relationships between different parts of an image, capturing global context more efficiently. This allows VIT to model long-range relationships between different parts of the image, potentially capturing more complex visual context compared to CNNs.Figure 3 illustrates the model architecture

C. LORA

Low-Rank Adaptation (LoRA) is a technique designed to enhance the efficiency of fine-tuning large-scale pre-trained language models. [8] By introducing trainable low-rank matrices into each layer of the model, LoRA enables significant parameter reduction while preserving performance. These matrices modify the original weights minimally, allowing the model to adapt to new tasks with fewer resources. This method is particularly useful in scenarios with limited computational capacity or data, as it reduces the need for full model retraining, speeds up the fine-tuning process, and decreases memory usage. Figure 4 illustrates the model architecture

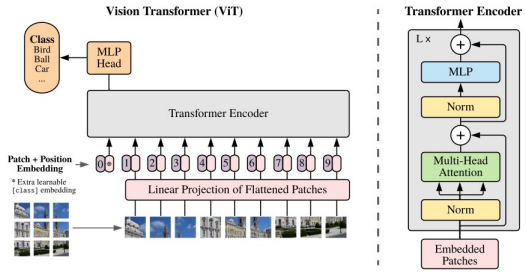

Fig. 3: VIT Archictecture. [6]

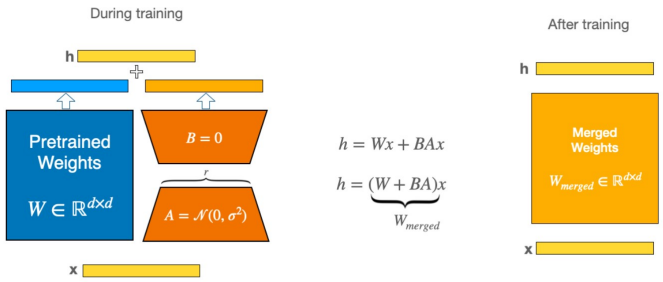

Fig. 4: LoRA Training [7]

Model

Multimodal models can be of various forms to capture information from the text & image modalities, along with some cross-modal interaction as well. Here, we explore "Fusion" Models, that fuse information from the text encoder & image encoder to perform the downstream task (visual question answering).

The text encoder can be a text-based transformer model (like BERT, RoBERTa, etc.) while the image encoder could be an image transformer (like ViT, Deit, BeIT, etc.). After passing the tokenized question through the text-based transformer & the image features through the image transformer, the outputs are concatenated & passed through a fully-connected network with an output having the same dimensions as the answer-space.

Specifically, in this project we have used to BERT and ViT combination due to the huge amount of resources available open source and ease of use.

Since we model the VQA task as a multi-class classification, it is natural to use the Cross-Entropy Loss as the loss function.

Performance Metrics

The Wu & Palmer similarity is a metric designed to measure the semantic similarity between two words or phrases. It does so by considering the positions of the concepts (c1 and c2) in a taxonomy and their relative location to their Least Common Subsumer (LCS(c1, c2)).

In the context of a directed acyclic graph, the Least Common Subsumer is the deepest node that has both considered nodes as descendants. Importantly, each node is considered a descendant of itself.

Wu & Palmer similarity proves effective for single-word answers (the primary focus in our task), but it may not be suitable for phrases or sentences due to its design.

The Natural Language Toolkit (nltk) provides an implementation of the Wu & Palmer similarity score based on the WordNet taxonomy. This implementation to align with the definition of Wu & Palmer similarity as specified in the VQA dataset from Georgia Tech.

Preprocessing of Data

1) The the data given in the VQA dataset by Georgia tech consists the annotation and question file along with the respective folders.
2) It was given that we had to sample 1/4 of the train images. Random images were chosen and new dataset was created. Also the images that were excluded from the sampling, the corresponding questions were removed from the annotations and the questions files as well.
3) Finally a CSV file for train and validation was created from the annotation and the question JSON file consists the important fields.
4) We created a answer_space.txt file which contains all the possible answers from data.csv file. This maps to the possible outputs of the Multi-Layer Perception at the end of the Pipeline of the model.

Model Overview

- **show_example function:** Displays a random (or specified) image from a dataset along with its associated question, answer, and label, using the IPython display module.
- **MultimodalCollator class:** The MultimodalCollator class acts as a data preprocessor for a multimodal VQA model. It takes care of preparing both the text (questions) and images for the model. For text, it uses a tokenizer to convert the questions into numerical representations with consistent length and additional information the model needs. For images, it opens and converts them to a format suitable for the model's input, potentially involving resizing and conversion to numerical tensors. Finally, it combines the processed text and image data along with the labels into a single structure that the model can use for training or prediction.
- **The MultimodalVQAModel class:** Builds a neural network specifically designed for Visual Question Answering (VQA). It utilizes pre-trained models like BERT for text and ViT for images. These encoders process the question and image separately, extracting key features. The model then combines these features to create a richer understanding that considers both textual and visual information. Finally, a classification layer takes this combined representation and predicts the most likely answer from a set of predefined options.
- **'create_multimodal_vqa_collator_and_model' class:** The code defines a function to create tools for processing text and image data together for a Visual Question Answering (VQA) task. It accomplishes this by loading

pre-trained models for both text and image encoding, then building a special collator to combine them, and finally creating a multimodal VQA model itself. This model takes a question and an image as input, and aims to answer the question based on the information in both.

- **'wup_measure' function:** This code defines a function to measure similarity between words. It considers synonyms (using WordNet) for each word. If found, it compares how closely related the synonyms are in WordNet's concept hierarchy. The closer the synonyms, the higher the similarity score. It also factors in how easily synonyms were found for each word.
- **'create_and_train_model' function:** This code trains a model for visual question answering (VQA). It first sets up the necessary components by creating a data processor and the VQA model itself.

```
MultimodalVQAModel(
  (text_encoder): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (image_encoder): ViTModel(
    (embeddings): ViTEmbeddings(
      (patch_embeddings): ViTPatchEmbeddings(
        (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
      )
      (dropout): Dropout(p=0.0, inplace=False)
    )
    (encoder): ViTEncoder(
      (layer): ModuleList(
        (0-11): 12 x ViTLayer(
          (attention): ViTAttention(
            (attention): ViTSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (output): ViTSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
          )
          (intermediate): ViTIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): ViTOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (dropout): Dropout(p=0.0, inplace=False)
          )
          (layernorm_before): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (layernorm_after): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
    (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (pooler): ViTPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (fusion): Sequential(
    (0): Linear(in_features=1536, out_features=512, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.5, inplace=False)
  )
  (classifier): Linear(in_features=512, out_features=29332, bias=True)
  (criterion): CrossEntropyLoss()
)
```

Fig. 6: Model Architecture without LoRA

```
trainable params: 15637140 || all params: 227342888 || trainable%: 6.88
PeftModel(
  (base_model): LoraModel(
    (model): MultimodalVQAModel(
      (text_encoder): BertModel(
        (embeddings): BertEmbeddings(
          (word_embeddings): Embedding(30522, 768, padding_idx=0)
          (position_embeddings): Embedding(512, 768)
          (token_type_embeddings): Embedding(2, 768)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (encoder): BertEncoder(
          (layer): ModuleList(
            (0-11): 12 x BertLayer(
              (attention): BertAttention(
                (self): BertSelfAttention(
                  (query): lora.Linear(
                    (base_layer): Linear(in_features=768, out_features=768, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.1, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=768, out_features=8, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=8, out_features=768, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                  )
                  (key): Linear(in_features=768, out_features=768, bias=True)
                  (value): lora.Linear(
                    (base_layer): Linear(in_features=768, out_features=768, bias=True)
                    (lora_dropout): ModuleDict(
                      (default): Dropout(p=0.1, inplace=False)
                    )
                    (lora_A): ModuleDict(
                      (default): Linear(in_features=768, out_features=8, bias=False)
                    )
                    (lora_B): ModuleDict(
                      (default): Linear(in_features=8, out_features=768, bias=False)
                    )
                    (lora_embedding_A): ParameterDict()
                    (lora_embedding_B): ParameterDict()
                  )
                  (dropout): Dropout(p=0.1, inplace=False)
                )
                (output): BertSelfOutput(
                  (dense): Linear(in_features=768, out_features=768, bias=True)
                  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                  (dropout): Dropout(p=0.1, inplace=False)
                )
              )
              (intermediate): BertIntermediate(
                (dense): Linear(in_features=768, out_features=3072, bias=True)
                (intermediate_act_fn): GELUActivation()
              )
              (output): BertOutput(
                (dense): Linear(in_features=3072, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
          )
        )
        (pooler): BertPooler(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (activation): Tanh()
        )
      )
      (image_encoder): ViTModel(
        (embeddings): ViTEmbeddings(
          (patch_embeddings): ViTPatchEmbeddings(
            (projection): Conv2d(3, 768, kernel_size=(16, 16), stride=(16, 16))
          )
          (dropout): Dropout(p=0.0, inplace=False)
        )
        (encoder): ViTEncoder(
          (layer): ModuleList(
            (0-11): 12 x ViTLayer(
              (attention): ViTAttention(
                (attention): ViTSelfAttention(
                  (query): lora.Linear(
                    (base_layer): Linear(in_features=768, out_features=768, bias=True)
```

Fig. 7: Model Architecture with LoRA

```
                (query): lora.Linear(
                  (base_layer): Linear(in_features=768, out_features=768, bias=True)
                  (lora_dropout): ModuleDict(
                    (default): Dropout(p=0.1, inplace=False)
                  )
                  (lora_A): ModuleDict(
                    (default): Linear(in_features=768, out_features=8, bias=False)
                  )
                  (lora_B): ModuleDict(
                    (default): Linear(in_features=8, out_features=768, bias=False)
                  )
                  (lora_embedding_A): ParameterDict()
                  (lora_embedding_B): ParameterDict()
                )
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): lora.Linear(
                  (base_layer): Linear(in_features=768, out_features=768, bias=True)
                  (lora_dropout): ModuleDict(
                    (default): Dropout(p=0.1, inplace=False)
                  )
                  (lora_A): ModuleDict(
                    (default): Linear(in_features=768, out_features=8, bias=False)
                  )
                  (lora_B): ModuleDict(
                    (default): Linear(in_features=8, out_features=768, bias=False)
                  )
                  (lora_embedding_A): ParameterDict()
                  (lora_embedding_B): ParameterDict()
                )
                (dropout): Dropout(p=0.0, inplace=False)
              )
              (output): ViTSelfOutput(
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (dropout): Dropout(p=0.0, inplace=False)
              )
            )
            (intermediate): ViTIntermediate(
              (dense): Linear(in_features=768, out_features=3072, bias=True)
              (intermediate_act_fn): GELUActivation()
            )
            (output): ViTOutput(
              (dense): Linear(in_features=3072, out_features=768, bias=True)
              (dropout): Dropout(p=0.0, inplace=False)
            )
            (layernorm_before): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (layernorm_after): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          )
        )
      )
      (layernorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (pooler): ViTPooler(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
    (fusion): Sequential(
      (0): Linear(in_features=1536, out_features=512, bias=True)
      (1): ReLU()
    )
    (fusion): Sequential(
      (0): Linear(in_features=1536, out_features=512, bias=True)
      (1): ReLU()
      (2): Dropout(p=0.5, inplace=False)
    )
    (classifier): ModulesToSaveWrapper(
      (original_module): Linear(in_features=512, out_features=29332, bias=True)
      (modules_to_save): ModuleDict(
        (default): Linear(in_features=512, out_features=29332, bias=True)
      )
    )
    (criterion): CrossEntropyLoss()
  )
)
```

Fig. 8: Model Architecture with LoRA

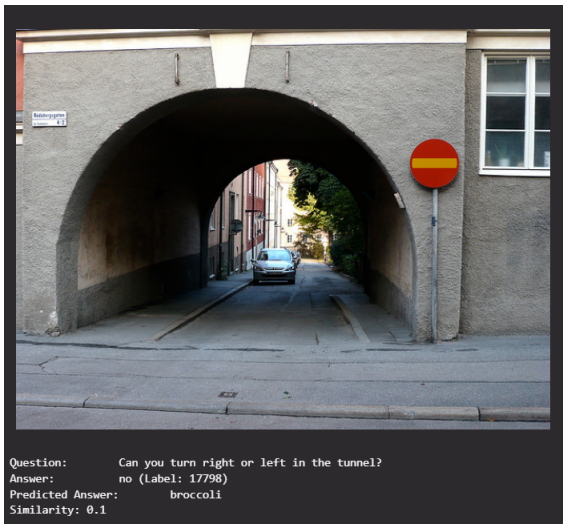

Fig. 9: Inference on figure 1, 2, 3

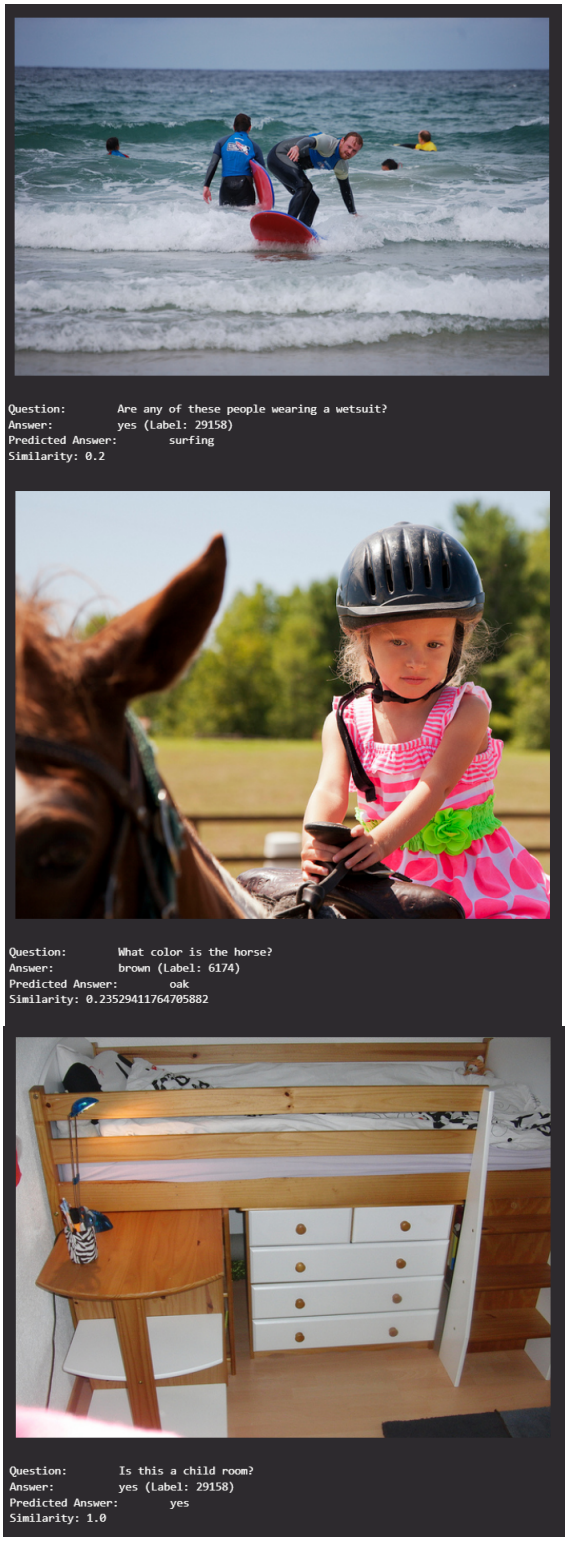

Fig. 10: Inference on figure 4

TABLE I: Comparison of Non-Lora and Lora Models

Model	Trainable Parameters	Accuracy	F1-Score	Precision	Recall	TTT (s)
Non-Lora	15,637,140	0.343	0.0612	0.057	0.066	22,528 sec
Lora	227,342,888	0.245	0.0562	0.0376	0.111	17,195 sec

From the above we can observe the following:

1) Model Specific Observations
   - As shown in Table 1 the trainable parameters have been greatly reduced by LoRa. The trainable parameters of LoRa reduces it to 6.88% of total parameters.
   - The accuracy of Non-Lora model is greater than that of LoRa model
   - The f1-score of both the models are almost similar.
   - The Time Taken for Training of LoRa model is nearly 76.326% that of Non-LoRa model.
   - We can conclude that we have made great optimization using LoRa.
2) Output Specific Observations
   - As shown in Figure 9 and 10, the 1st image/question pair shows a tunnel with tree near-by. The answer given to this is broccoli. The model model mostly misinterpreted the question as it observed the word right and also found a tree near by which is very similar to a broccoli in appearance.
   - The 3rd image says the color of the horse is oak which is a lighter shade of brown making it very similar.
   - The 4th image correctly decided whether the image provided is a child's room or not.
3) General Observations
   - We noticed that the dataset was biased with the questions to which the answers were yes/no. This made the model skewed towards the yes/no answer. Therefore in general, the questions with yes/no classification performed much better than other classification.
   - One possible solution to this to under sample the question with answers that are either yes or no to avoid the skewing of the model.

REFERENCES

[1] GitHUB Link
[2] One Drive Link.
[3] https://faculty.cc.gatech.edu/~parikh/Publications/ICCV2015_VQA.pdf.
[4] https://arxiv.org/pdf/1810.04805.
[5] https://www.sbert.net/examples/unsupervised_learning/MLM/README.html
[6] https://arxiv.org/pdf/2010.11929
[7] https://arxiv.org/pdf/2106.09685
[8] https://www.linkedin.com/pulse/lora-low-rank-adaptation-efficient-fine-tuning-large-language/
[9] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.