Visual Question Answering: Final Project Report

Ram Sai Koushik P
IMT2021072

Barath S Narayan
IMT2021524

Nilay Kamat
IMT2021096

I. Introduction

The objective of this project is to build a model for **Visual Question Answering(VQA)** which takes as input an image and a corresponding question and gives an answer to the question. We will also be fine-tuning pre-trained models with and without the help of Low-Rank Adaptation(LoRA) and **comparing the training times and performances** of these fine-tuned models.

II. Dataset Preparation

The dataset used for this project is the VQAv2 dataset released by Georgia Tech. The dataset size is very large, hence, the following steps were followed for dataset pre-processing:

1) **Creating Metadata Dataframe for Train Images:** A Dataframe was generated to store metadata about the train images, including image name, image ID, image path, and a hashed image ID used for random sampling. This involved parsing file names to extract image numbers and creating a mapping dictionary for hash values(utilized in sampling the dataset).
2) **Random Sampling of Dataset:** Approximately 25% of the images (20,695 out of 82,783) were randomly selected for further processing. Random indices were generated within the range of 1 to 82,783.
3) **Storing Questions in CSV Format:** Questions associated with the images were initially in JSON format , we converted it into CSV file for easier data manipulation.
4) **Filtering Questions Corresponding to Sampled Images:** Only questions corresponding to the images in the sampled dataset were retained. This filtering process ensured that the questions were aligned with the sampled images for training purposes.
5) **Storing Annotations in CSV Format:** Annotations associated with the questions were initially in JSON format, we converted into a CSV format. This allowed for easier data manipulation and storage.
6) **Filtering Annotations Corresponding to Sampled Images:** Similar to the questions, annotations were filtered to include only those associated with the sampled images.
7) **Merging Questions and Annotations with Image Metadata:** The dataframes corresponding to questions and annotations were merged with the image metadata to create a single dataframe for the whole dataset for training. This consolidated dataset contains information about questions, annotations, and corresponding images.
8) **Deleting Unused Images:** Images that were not included in the sampled dataset were removed to optimize storage and ensure that only relevant images were retained for training purposes.
9) **Adding Relative Paths for Kaggle:** A new column was added to store relative paths for Kaggle compatibility, facilitating easier access and usage of the dataset on the Kaggle platform.
10) **Understanding the Dataset:** Finally, an exploration of the dataset was conducted to understand its structure and contents, including checking the number of unique answer types and sample answers associated with questions.

III. Methodology

The idea being followed here is to **generate embeddings** for the **questions** using **BERT** model and **generate image embeddings** using **ViT** model. We then perform **co-attention** on these generated embeddings. This idea was inspired from **ViLBERT Architecture** and we are attempting to create a model that is similar to ViLBERT in order to achieve the objective of the project. We then fine tuned the weights of ViT and BERT models and trained the weights of the co-attention and classification layers to create the final models.

After creating this baseline model, we explored the usage of Low-Rank Adaptation(LoRA) to fine-tune this baseline model.

A. Implementation Details

The implementation of the above mentioned methodology is done through the following classes.

1) Feed Forward Neural Network
2) Attention BERT
3) Self TRM
4) MLP classifier
5) BiAttention for Image and Word
6) Multimodal BERT
7) Multimodal Collator

The descriptions of the above classes are as follows:

1) The **FeedForwardNeuralNetwork** class consists of **two linear layers** with a **GELU** activation function applied after each. The first layer **doubles** the embedding dimension, and the second layer reduces it back to the original size of **768**. This structure refines the input embeddings by introducing non-linearity and enabling more complex feature transformations.

2) The **AttentionBERT** class implements a co-attention mechanism using multi-head attention, specifically designed to process embeddings from BERT. The class initializes with **layer normalization**, a **feed-forward neural network**, and **multi-head attention** layers. In the forward method, query and input features are passed through the multi-head attention mechanism to compute co-attention. The output undergoes residual connections and layer normalization to maintain stability and improve learning. Finally, the output is passed through a feed-forward neural network with additional residual connections and normalization to produce the final refined embeddings.
3) The **SelfTRM** class implements a stack of transformer layers, each utilizing the AttentionBERT module. For fine-tuning using LoRA, this module is enhanced with **PEFT** (Parameter-Efficient Fine-Tuning) models. It initializes with a specified number of layers and heads, creating a list of transformer layers. In the forward method, the input is passed sequentially through each layer, applying self-attention and refining the embeddings at each step.
4) The **MLP classifier** class consists of two linear layers with a GeLU activation function applied after the first layer. The first layer doubles the embedding dimension, and the second layer maps the input features to the output classes. The GeLU activation function introduces non-linearity between the layers, and then argmax is used to obtain the predicted class.
5) The **BiattentionforImageandWord** class implements a **bi-directional attention mechanism** for processing image and word embeddings. It consists of separate sets of layers for images and words, with each set containing alternating self-attention and cross-attention layers. During initialization, multiple AttentionBERT layers are created for both images and words. For fine-tuning using LoRA, this module is enhanced with PEFT models. In the forward method, each **odd-numbered layer** applies **cross-attention** between images and words, while each **even-numbered layer** applies **self-attention** to refine the embeddings. This iterative process allows the model to capture complex interactions and dependencies between visual and textual information, enhancing the overall representation.
6) The **MultimodalBert** class integrates text and image embeddings for multimodal tasks. It utilizes pre-trained models from HuggingFace for text (BERT) and image (ViT) embeddings. The BiattentionforImageandWord module applies alternating self-attention and cross-attention layers to capture interactions between text and image features. Additionally, a SelfTRM module processes the text features with self-attention layers. The final combined class token embeddings from both modalities are passed through an MLP_classifier for classification.
7) The **MultimodalCollator** class is designed to preprocess and batch multimodal data for training. It uses the @dataclass decorator to automatically generate special methods, reducing boilerplate code. The class contains methods for tokenizing text and preprocessing images. The tokenize_text method uses a tokenizer to encode text into input IDs, while the preprocess_images method uses a preprocessor to convert image paths into pixel values. The __call__ method combines these preprocessed texts and images with their corresponding labels into a single batch dictionary, ready for model input.

The function **createMultimodalVQACollatorAndModel** is used to create a multimodal Collator using a pre-trained Tokenizer and feature extractor along with a multimodal model with the help of MultimodalBert class. This is the model we then train and use for our objective.

The model created is trained using the Trainer class from the transformers library. This class simplifies the training process by handling the optimization and evaluation steps. The training arguments are set to specify the training configuration, including the number of epochs, batch size, and evaluation strategy. The trainer is initialized with the multimodal BERT model and the given dataset.

We constructed the Trainer() using Huggingface with the following parameters:

1) model (In the fine-tuning with LoRA, this was replaced by lora_model)
2) multi_args
3) train_dataset=dataset['train']
4) eval_dataset=dataset['test']
5) data_collator=collator
6) compute_metrics=compute_metrics

Here, model indicates the model we have created and want to train. multi_args are the training arguments to be passed to the model in training, train and eval datasets as the name suggests, data_collator is the multimodal collator we have created and compute_metrics is an invokation of the compute_metrics function that calculated the required metrics on our model - Accuracy, F1 Score, Precision, Recall.

IV. Architecture

The architecture of the model created is similar to the ViLBERT model.

We have implemented the above architecture with 3 layers of Self-TRM and 3 layers of Co-TRM. We have used the "**google/vit-base-patch16-224-in21k**" ViT model for feature extraction of images and we have used the pre-trained "**google-bert/bert-base-uncased**" BERT model for feature extraction of questions. The number of heads used per layer is 3 and the self-TRM and cross-TRM architectures are the same as mentioned in the ViLBERT paper, which can be seen as shown in figures 1 and 2 respectively.

The overall architecture of ViLBERT is as shown in 3. The feed forward neural network implemented in the above architecture is as shown in 4. The above model implemented was our base model. The overall architecture of our model is as shown in 5 and 4.

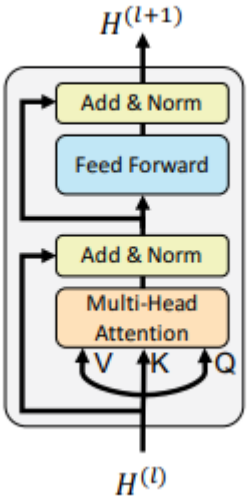

Fig. 1: ViLBERT self-TRM Architecture

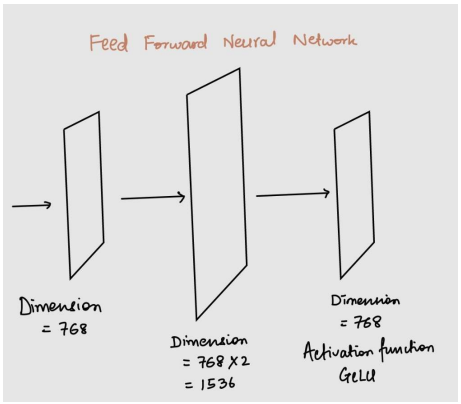

Fig. 4: Feed Forward Neural Network

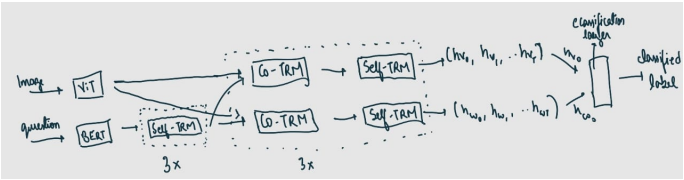

Fig. 5: Model Architecture

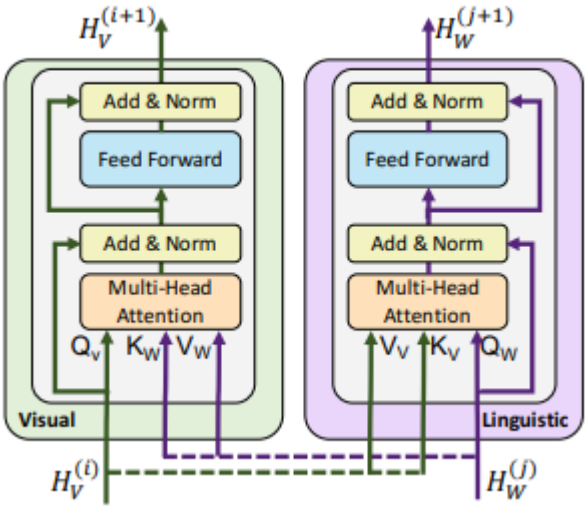

Fig. 2: ViLBERT co-TRM Architecture

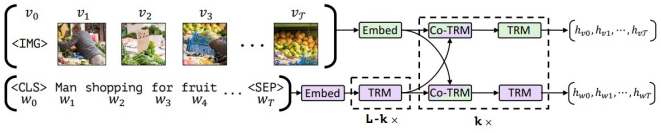

Fig. 3: ViLBERT Architecture

V. Training of the Models

The Training arguments used for training the model are given below

```
multi_args = TrainingArguments(
output_dir="/kaggle/working/vilbert/",
  seed=12345,
  evaluation_strategy="epoch",
  logging_strategy="epoch",
  save_strategy="epoch",
  save_steps=100,
  save_total_limit=1,
  metric_for_best_model="acc",
  per_device_train_batch_size=32,
  per_device_eval_batch_size=32,
  remove_unused_columns=False,
  num_train_epochs=5,
  fp16=False,
  dataloader_num_workers=4,
  load_best_model_at_end=True,
  report_to="tensorboard",
)
```

We have used epoch based evaluation and saving strategy to understand the training of the model at each epoch to get an idea of the number of epochs to train without overfitting to the training data. Batch size has been kept to 32 as higher ones usually resulted in memory error. Load best model at end ensured that the model used after the training was the best model obtained from training. Metric for the best model was choosen as accuracy. This decided which model was loaded at

the end. Save_total_limit ensures that only those many number of models are saved as the models are usually large due to the number of parameters present.

VI. Usage of LoRA

We have leveraged 2 LoRA configurations to train 2 fine-tuned models with LoRA. The ranks used in these 2 models are 6 and 8, with the other values in the 2 configurations as follows:

1) Configuration 1
   a) init_lora_weights="gaussian"
   b) lora_alpha=16
   c) target_modules=["query", "value", "linear"]
   d) lora_dropout=0.1
   e) bias="none"
   f) modules_to_save=["classifier"]
2) Configuration 2
   a) init_lora_weights="gaussian"
   b) lora_alpha=16
   c) target_modules=["out_proj"]
   d) lora_dropout=0.1
   e) bias="none"
   f) modules_to_save=["classifier"]

There are 2 configurations as there are many subclasses which are not identified by the model as torch layers directly. Hence the AttentionBert module is converted to a LORA model using configuration 2 as it contains mainly a Multihead Attention model whose type is out_proj. While the whole model pipeline is converted into a LORA model using configuration 1 as it contains many layers of query, value and linear due to the presence of VIT and BERT architectures. This effectively covers most of the model parameters and hence on usage reduces the number of trainable parameters by a lot which helps in reducing the training time by a signification

VII. Results and Required Metrics

The results of the models we trained and used are as follows:

A. Normal fine tuning(without LoRA)

The metrics for the training part are:

1) Training Time = 17115.4327 seconds
2) Training Loss = 2.0747772261192536

The total parameters used for training were **262011474**.
The metrics for the testing part are:

1) Accuracy = 0.4239708712221258 = 42.39%
2) F1 score = 0.4014791773145704
3) Precision = 0.39607061916033237
4) Recall = 0.4239708712221258

B. Fine tuning using LoRA

1) **LoRA with Rank=6**
   The metrics for the training part are:
   a) Training Time = 13743.687 seconds
   b) Training Loss = 2.4260916360561535

   The **trainable parameters** for this model were **33217449**, which amounted to **12.68%** of the total parameters.
   The metrics for the testing part are:
   a) Accuracy = 0.39952080896502284 = 39.95%
   b) F1 score = 0.36578400452716814
   c) Precision = 0.364303162991559
   d) Recall = 0.39952080896502284
2) **LoRA with Rank=8**
   The metrics for the training part are:
   a) Training Time = 13892.8012 seconds
   b) Training Loss = 2.4209952413532876

   The **trainable parameters** for this model were **33364905**, which amounted to **12.73%** of the total parameters.
   The metrics for the testing part are:
   a) Accuracy = 0.4041429272157869 = 40.41%
   b) F1 score = 0.3784951240048250
   c) Precision = 0.3753541426803139
   d) Recall = 0.4041429272157869

The overall results are as follows:

Model	Accuracy	F1-Score	Precision	Recall
Baseline	42.39%	0.40147	0.39607	0.42397
LoRA(rank=6)	39.95%	0.36578	0.36430	0.39952
LoRA(rank=8)	40.41%	0.37849	0.37535	0.40414

Model	Training Time	Training Parameters
Baseline	17115.4327	262011474
LoRA(rank=6)	13743.687	33217449
LoRA(rank=8)	13892.8012	33364905

VIII. Inference of Trained Models

We have stored the trained models as binary files. We then loaded the models and gained an inference on a small selection of the dataset. The following image gives a clear illustration of the answers given by the models on a few questions pertaining to the corresponding image.

```
What is the elephant's trunk pointing at?
elephant
Is this a zoo?
yes
Does the elephant have tusks?
no
```