AI 825: Visual Recognition

Final Project Report

By
Madhav Sood (IMT2021009)
Yukta Rajapur (IMT2021066)
Brij Desai (IMT2021067)

18h May 2024

Link for the saved models: VR Project Models

Objectives:

- 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

Dataset:

For this project, VQA dataset released by Georgia Tech was used. Given that the dataset size is very large, we will be using only 1/4th of the training images for this project.

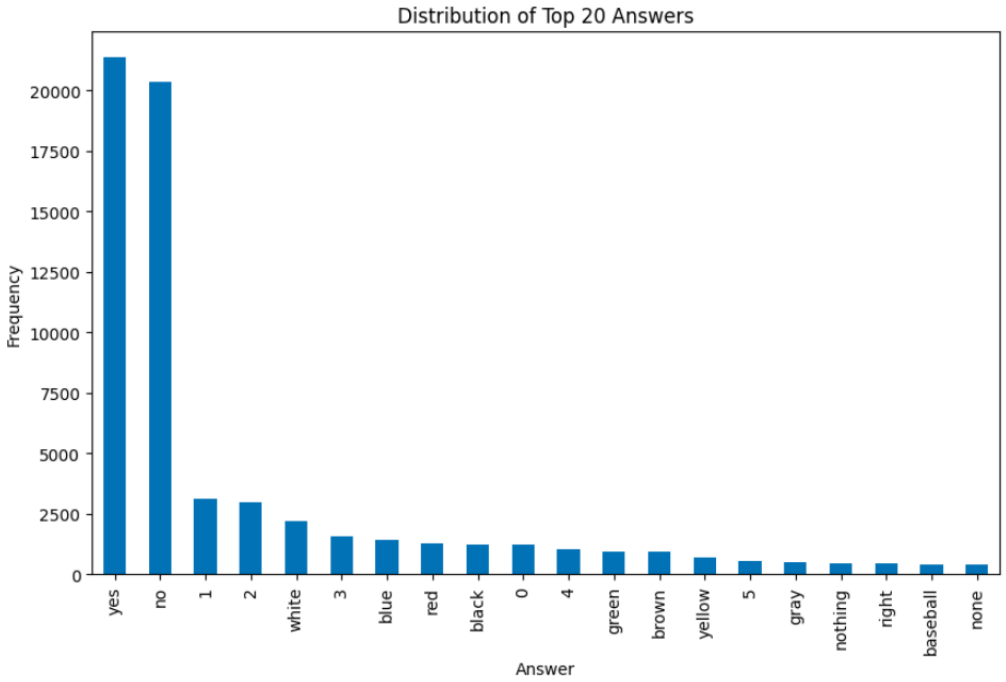
Data Preprocessing:

This dataset has 443757 datapoints and 7 features along with their target label (multiple_choice_answer).

First, we grouped the data by the image_ids and randomly sample 1/4th of the training images (which were 20696 images). Then we sampled the dataset based on these 1/4th training images with left us with 110247 data points.

We observed that most of the questions and answers are simple and clean text, but some questions contain punctuation common word contractions like what's, it's, they're, etc, and noun contractions like guy's, man's, dog's, etc., and some answers also contain punctuation. Hence, we need to perform the data cleaning operation on the question and answer dataset and expand contractions. The question and answer were also converted to lowercase.

We observed the distribution of the top 20 answers as follows:



- A very large number of the answers are “yes” and “no”.
- In number answer_type, 1,2,3,0,4,5 these numbers have more counts than other numbers.

We observed that the dataset is highly imbalanced, and it has 9069 unique answers, which is a very huge number of class labels, so we considered the top 1000(K) answers as the class labels and drop the rest of the datapoints. The top 1000 answers covered 87.88% of the datapoints of the dataset which means we did not lose much data. The top 1000 answer dataset has 96891 datapoints (after filtering the dataset according to the top 1000 answers).

Baseline Model (without LoRA):

First comes the part of forming the dataset and data encoding:

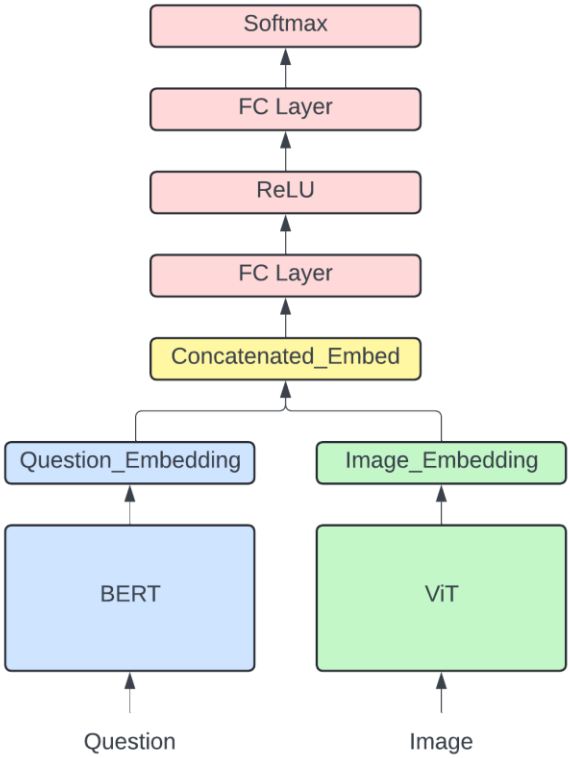
Many machine learning algorithms as well as Deep Learning Algorithms are generally unable to work with text and image data when fed directly into the model. This data must be further converted into numbers and the same is required for both the input and output variables. Here, we have two input variables text question and image, and one output variable which is an answer. So here are the encoding techniques to encode questions, images, and answers.

For Questions:
For questions, we tokenized the question using the BERT_Tokenizer from Hugging Face, to convert the question into a format that can be fed into the BERT Model.

For Images:
For images, we converted the image to tensors using the AutoImageProcessor from Hugging Face, these will be converted into a format such that they can be fed into the Vision Transformer (ViT).

For Answers:
Answers are the target label that we want to predict. These answers are label encoded from 0 to K-1 using a label encoder, where K is the number of class labels which is 1000. We view the visual question answering task as a multi-class classification task.

Model:



The model architecture diagram is as given in the image above. Following is the information about the total parameters and total trainable parameters of the baseline model.

```
==========================================================================================
Model Parameters:
==========================================================================================
Total Parameters: 196197224
Trainable Parameters: 196197224
==========================================================================================
```

The dataset is divided into training and evaluation datasets (80-20 split is done respectively)

We utilize a pre-trained BERT model to generate embeddings for the question and a pre-trained ViT model to produce embeddings for the image. These embeddings are then concatenated. The combined embedding is passed through fully-connected (FC) layers and ultimately through a SoftMax layer to obtain a probability distribution across all possible labels (answers). Loss function used is cross-entropy loss for multi-class classification.

The ViT and BERT models are fine-tuned for the Visual-Question answering task (viewed as a multi-class classification task) using the training dataset.

Results for the Baseline Model:

Number of epochs: 2

Batch Size: 2 (We've used a smaller batch size of 2 to avoid running into CUDA out of memory error)

Training Time: 17279.5814 seconds (or 4 hours 48 mins)

Training Loss: 4.355

Evaluation Loss: 4.291

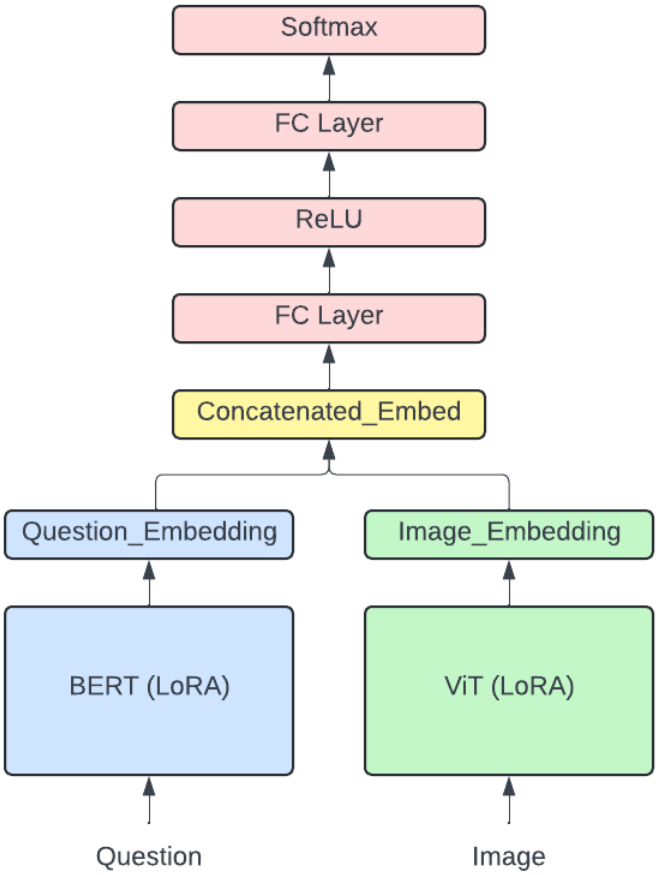
Accuracy (Evaluation): 23.02% (or 0.2302)

F1 Score (Evaluation): 0.136

Recall (Evaluation): 0.230

Precision (Evaluation): 0.100

LoRA Model:



Low-Rank Adaptation (LoRA) is a technique designed to fine-tune large language models (LLMs) efficiently by adding and training a small number of low-rank matrices to the model's existing layers instead of modifying all the parameters. This approach significantly reduces the computational resources and time required for fine-tuning. LoRA leverages the fact that many adaptation tasks can be effectively addressed with low-dimensional changes, allowing the base model to remain mostly unchanged while still adapting to specific tasks or domains. By focusing on a subset of parameters, LoRA maintains the original model's integrity and performance, making it a cost-effective solution for customizing LLMs.

The data pre-processing and encoding are the same as the baseline model.

The model architecture diagram is as given in the image above. Since the BERT and ViT pre-trained models contribute most of the parameters of our model, we applied LoRA only to the BERT and ViT layers.

The LoraConfig was defined as follows:

```
bert_lora_config = LoraConfig(
    r=64,
    target_modules=["query", "value"],
    lora_alpha=32,
    lora_dropout=0.05
)

vit_lora_config = LoraConfig(
    r=64,
    target_modules=["query", "value"],
    lora_alpha=32,
    lora_dropout=0.05
)
```

Following is the information about the total parameters and total trainable parameters of the LoRA models:

BERT:

```
trainable params: 2,359,296 || all params: 111,841,536 || trainable%: 2.1095
```

ViT:

```
trainable params: 2,359,296 || all params: 88,748,544 || trainable%: 2.6584
```

Total:

```
trainable params: 4,798,592 || all params: 196,197,224 || trainable%: 2.4050
```

Results:

Number of epochs: 2

Batch Size: 2 (We've used a smaller batch size of 2 to avoid running into CUDA out of memory error)

Model	Training Time (Seconds)	Training Loss	Evaluation Loss	Accuracy (Eval)	F1 Score (Eval)	Recall (Eval)	Precision (Eval)
baseline	17279.5814	4.355	4.291	0.2302	0.136	0.230	0.100
LoRA (r = 8)	12814.5405	2.846	2.578	0.339	0.269	0.339	0.269
LoRA (r = 64)	12928.4002	2.844	2.567	0.331	0.261	0.331	0.248

It's interesting to note that the accuracy improved after applying LoRA. This might be because of the following reasons:

- **Training Efficiency**: LoRA can achieve faster convergence due to the smaller number of trainable parameters, enabling the model to quickly reach a high-performing state with reduced risk of overfitting. Given that we are training for only 2 epochs, LoRA likely learns more effectively than the baseline model in the short term.

- **Regularization Effect**: The low-rank adaptation can act as a form of regularization, preventing overfitting by reducing the complexity of the modifications made to the model. This can result in better generalization and higher accuracy.

Challenges faced:

- The VQA dataset is very large, making it challenging to download, store, and preprocess efficiently. Working with a subset (1/4th) of the data requires careful sampling to maintain a representative dataset.

- Integrating BERT for text processing and ViT for image processing requires careful design to ensure that the outputs from both models can be effectively combined.

- With model training times extending to 4-5 hours, optimizing our workflow was crucial, as any mistakes resulted in significant time loss.

- We had to carefully manage our Kaggle credits to ensure efficient use across the team.

- We encountered multiple errors with the Trainer while implementing LoRA, and debugging these issues proved to be difficult.