AI 825: Visual Recognition Visual Question Answering Report

Goutham U R
IMT2021045
Goutham.UR@iiitb.ac.in

Pannaga Bhat
IMT2021080
Pannaga.Bhat@iiit.ac.in

Nori Meher Ashish
IMT2021085
meher.ashish@iiitb.ac.in

***Abstract*—This is the report for our team as part of the final project in the AI 825: Visual Recognition course.**

***Index Terms*—BERT, ViT, multimodal**

I. Introduction

A. Objectives

1) **To build a model for Visual Question Answering:**
   - The model should take as input an image and a corresponding question.
   - The model should output an answer to the question.
2) **To fine-tune pre-trained models with and without the help of LoRA:**
   - Fine-tune pre-trained models using traditional methods.
   - Fine-tune pre-trained models using LoRA (Low-Rank Adaptation).
3) **To compare the training time and performance of the fine-tuned models:**
   - Measure and compare the training time for models fine-tuned with and without LoRA.
   - Evaluate and compare the performance of the fine-tuned models using appropriate metrics.

B. Models Used:

1) **BERT:** BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained transformer-based model designed for natural language processing tasks, enabling deep bidirectional understanding of text.
2) **ViT:** ViT (Vision Transformer) is a transformer-based model that applies the transformer architecture to image recognition tasks by treating image patches as tokens, achieving state-of-the-art results on various vision benchmarks.

II. Data Pre-processing

The complete dataset is first downloaded from the Georgia Tech website. Then, we add all the files with an image extension like `.png` or `.jpg` to a list.
The list is then randomly sampled to get a quarter of the elements and these are copied elsewhere and then uploaded to Kaggle.
The `.json` file containing the questions is filtered to extract the questions that only refer to selected subset of the images using the `image_id` property. A new file is created to store these questions.
A similar technique is used to extract the relevant annotations and question ids. These are also stored in a `.json`.
From this, we create 3 `.csv` files that contains a row per question. The properties stored are the question, the answer and the image id. The csv files are:

1) *data_train.csv*: Contains all the questions, answers and image-ids corresponding to the train dataset.
2) *data_val.csv*: Contains all the questions, answers and image-ids corresponding to the validation dataset.
3) *data.csv*: Union of *data_train.csv* and *data_val.csv*.

In addition to this, 3 text files were generated:

1) *train_image_ids.txt*: Contains all the image ids in the train dataset.
2) *val_image_ids.txt*: Contains all the image ids in the validation dataset.
3) *answers.txt*: Contains all the answers in text format.

These files are used for training the model. To validate, this process is repeated for another 25% of the dataset. Note that some images may be present in both datasets. The list of image ids for both subsets is also written to a `.csv`.

III. Model Training

A. Data Collator

We build a data collator that will perform the following tasks:

- `tokenizer`: Takes in the text portions of the training data and tokenizes them.
- `preprocess_images`: Extracts the images from the directory and processesses them into tensors of the RGB pixel values.
- `__call__`: Processes a batch of raw data, including both text and image inputs, and combines the tokenized text data, preprocessed image data, and labels into a single dictionary. This dictionary is then ready to be used as input for a multimodal machine learning model.

B. Multimodal VQA model

The model integrates both text and image information to answer questions based on visual content. The implementation leverages pre-trained models for both text and image encodings, followed by a fusion module that combines these modalities and a classifier that outputs the answer predictions.The model can be broken down into the following components:

- `Text Encoder`: Uses a pre-trained BERT model from the Hugging Face library for text encoding.
- `Image Encoder:`: Uses a pre-trained Vision Transformer (ViT) model from the Hugging Face library for image encoding.
- `Fusion Module`: A sequential model that combines the outputs of the text and image encoders, reduces the dimensionality, and applies non-linear activation and dropout for regularization.
- `Classifier`: A linear layer that maps the fused features to the number of possible answer labels.
- `Loss Function:`: Cross-Entropy Loss, which is suitable for classification tasks.

The text encoder processes the input text, producing a contextual representation and the image encoder processes the input images, producing a visual representation. The encoded text and image features are concatenated and passed through the fusion module to combine the information from both modalities. The fused features are passed through a linear layer to produce the logits for each possible answer. A dictionary containing the logits is returned. If labels are provided, the loss is also computed and included in the output.

This Multimodal VQA model effectively combines textual and visual data to answer questions based on image content. The use of pre-trained models for text and image encoding ensures that the model benefits from rich, pre-learned representations, while the fusion and classification components allow for effective integration and interpretation of the multimodal data

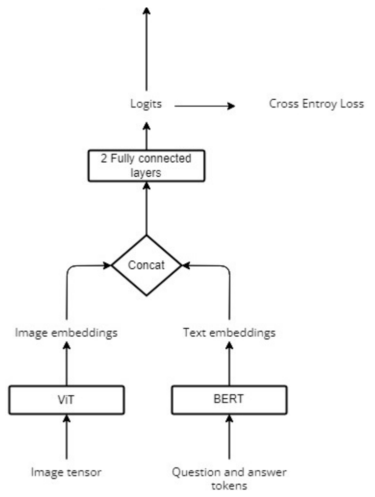

Fig. 1. Image depicting the architecture of the model

C. Performance Metrics

- Wu-Palmer Similarity Function:
  - `wup_measure(a, b, similarity_threshold=0.925)` computes the Wu-Palmer similarity between two words/phrases `a` and `b` using WordNet. If the similarity score exceeds the threshold, it indicates synonyms; otherwise, it is downweighted.
- Semantic Field and Stem Word Processing:
  - `get_semantic_field(word)` retrieves the WordNet synsets for a word and returns them with a weight.
  - `get_stem_word(word)` processes words formatted as `word\d+:wordid`, returning the base word and a weight for adjusting the similarity score.
- Similarity Calculation:
  - `wup_measure` processes input words to get their semantic fields and weights. It calculates the Wu-Palmer similarity between all synset pairs from the two words, finding the maximum score, which is adjusted by weights to produce the final score.
- Batch Processing:
  - `batch_wup_measure(labels, preds)` calculates the average Wu-Palmer similarity for batches of predicted and true labels using `wup_measure` and a predefined `answer_space` dictionary to map labels to words/phrases.

D. Training

1) **Function:** The `create_and_train_model` function aims to streamline the process of building and training a Multimodal VQA model. It takes in datasets, training arguments (`args`), and specifications for text and image encoders to create a model capable of answering questions based on both text and image inputs.
2) **Model Initialization:** It initializes a Multimodal VQA collator and model using the specified text and image encoders (`text_model` and `image_model`).
3) **Trainer Setup:** The function then configures a Trainer object (`multi_trainer`) using the initialized model, training arguments, datasets, collator, and a metrics computation function (`compute_metrics`).
4) **Training and Evaluation:** With the Trainer set up, the function proceeds to train the model and evaluate its performance. It returns both training and evaluation metrics, along with the collator, model, and the Trainer object used for training.
5) **Output Directory Adjustment:** Notably, it adjusts the output directory for the model checkpoints based on the specified multimodal model name (`multimodal_model`). This ensures organized storage of training artifacts specific to each model configuration.

E. Hyperparameters

TABLE I
NAMES AND VALUES OF RELEVANT HYPERPARAMETERS

Name	Value
Output directory	`/kaggle/working/checkpoint/`
Evaluation strategy	Steps
Evaluation steps	100
Logging steps	100
Save steps	100
Metric for best model	Wups
Batch size	64
Epochs	3

F. Model Loading:

1) **Checkpoint Management**: The script identifies the latest checkpoint from a folder containing model checkpoints by parsing the directory names and extracting the checkpoint numbers. It then constructs the path to the latest checkpoint file.
2) **Model Loading**: Using the path to the latest checkpoint, the script loads the state dictionary of a `MultimodalVQAModel` from the checkpoint file. This allows the model to be restored to the state it was in when the checkpoint was saved.
3) **Device Assignment**: Finally, the script moves the loaded model to a specified device, enabling it to utilize the computational resources available on that device, such as a GPU for accelerated computation.

G. LORA Fine Tuning

Low-Rank Adaptation (LoRA) is an efficient fine-tuning technique designed to reduce the computational cost and memory requirements associated with training large pre-trained models. By introducing low-rank updates to the model parameters, LoRA significantly decreases the number of trainable parameters while maintaining high performance. This section details the application of LoRA tuning within the create_and_train_model function.

Traditional fine-tuning of large models involves updating a substantial number of parameters, which can be computationally expensive and require extensive memory. LoRA addresses this by decomposing the parameter update into two low-rank matrices, thereby reducing the number of parameters that need to be updated during training. This method is particularly useful in scenarios where computational resources are limited or when dealing with extremely large models.

Algorithm:

Initialization:

1) Create the Multimodal VQA model using pre-trained text and image encoders.
2) Initialize the tokenizer and feature extractor for the respective encoders.

LoRA Fine-Tuning:

1) Freeze the original parameters of the text and image encoders by setting `requires_grad` to `False`.
2) Introduce low-rank matrices U and V for each parameter in the encoders.
3) Update the parameter values by adding the product of the low-rank matrices U and V^{T}.

Training Setup:

1) Create a `Trainer` instance with the model, training arguments, datasets, and collator.
2) Train the model using the provided training dataset and evaluate its performance on the test dataset.

H. Evaluation Metrics

TABLE II
EVALUATION METRICS

Name	Value
Evaluation Loss	3.40232515335083
Evaluation WUPS	0.37370086595990676
Evaluation Accuracy	0.3421506933229381
Evaluation F1 Score	0.004073877174843525
Evaluation Runtime (s)	471.7977
Evaluation Samples per Second	114.182
Evaluation Steps per Second	1.785
Epoch	3.0

I. Traning Time and Accuracy

TABLE III

Name	Value
Accuracy	34.21%
Training Time	8 hours 52 minutes