Visual Question Answering using BERT and VIT

Krutik Patel
IMT2021024

Suyash Ajit Chavan
IMT2021048

Dhairya Gupta
IMT2021005

1 INTRODUCTION

Visual Question Answering (VQA) combines computer vision and natural language processing to answer questions about images. In this study, we develop a VQA model using BERT for textual embeddings and Vision Transformer (ViT) for visual embeddings, without relying on integrated multimodal models like CLIP or BLIP.

Our primary focus is to compare the traditional fine-tuning approach with Low-Rank Adaptation (LoRA) in terms of accuracy, F1 score, precision, recall, and training time. The findings suggest that LoRA not only enhances the performance of the VQA model but also significantly reduces the training duration, showcasing its potential for more efficient model optimization.

2 DATA PREPROCESSING

2.1 Datasets Used

- **Questions:** v2_OpenEnded_mscoco_train2014_questions
- **Annotations:** v2_mscoco_train2014_annotations
- **Images:** v2_mscoco_train2014_images

All three datasets are in JSON format.

2.2 Data Exploration

Question IDs and annotation IDs are in the same order, allowing for direct index access to retrieve questions and annotations. Two methods are defined:

- **ShowSingleInstance(index):** Displays the question, answer, and image name tuple for the specified index.

Figure 1: Dataset instance 0

- **getSingleInstance(index):** Returns the question, answer, and image name tuple for the specified index.

The following observations are made:

- Each question has an associated **question_type**. There are 65 different question types, which can be utilized for further sampling.
- Each question has 10 answers with corresponding confidence levels such as "yes," "no," and "maybe."

Figure 2: Dataset instance 30000

2.3 Sampling Strategy and Train-Validation Split

- Approximately $\frac{1}{4}$ of the dataset, totaling 150,000 rows, is sampled from the original dataset using **stratified sampling** technique based on the **question type** attribute (**stratify=vqa_dataset["question_type"]**).
- Stratified sampling ensures equal proportions of classes/question types in both the training and validation splits.
- An 80-20 train-validation split is then performed.

2.4 Answer-Based Datasets

Based on the answers, different datasets are created focusing on answer dictionaries (i.e., the set of all possible answers) as follows:

- **Dataset1:** Contains a list of 10 answers as in the original dataset.

	image_name	question	answers	question_type
0	COCO_train2014_000000400999.jpg	What size table is this?	[{'answer': '4x4', 'answer_confidence': 'maybe...	what
1	COCO_train2014_000000284274.jpg	What kind of giraffe are standing in the field?	[{'answer': 'tall one', 'answer_confidence': '...	what kind of
2	COCO_train2014_000000065456.jpg	Will the guy catch the ball?	[{'answer': 'no', 'answer_confidence': 'yes', ...	none of the above
3	COCO_train2014_000000078322.jpg	Which man is more dressed up?	[{'answer': 'none', 'answer_confidence': 'yes'...	which
4	COCO_train2014_000000523894.jpg	Is that zebra eating grass?	[{'answer': 'no', 'answer_confidence': 'yes', ...	is

Figure 3: Dataset1

- **Dataset2:** Contains a single answer with answer confidence equal to "yes" (contains 18,000 different answers in the answer dictionary).
- **Dataset3:** Contains a single answer with answer confidence equal to "yes" and the answer is a single word (contains 8,100 different answers in the answer dictionary).
- **Dataset4:** Contains a single answer with answer confidence equal to "yes" and the answer is either "yes" or "no" (contains 2 different answers in the answer dictionary).

Dataset4 essentially contains Yes-No type questions. Various experiments are conducted on these datasets.

	index	image_name	question	answers	question_type
45	45	COCO_train2014_000000393230.jpg	What color is the sky?	blue and white	what color is the
46	46	COCO_train2014_000000393230.jpg	What is the person wearing?	ski clothes	what is the person
47	47	COCO_train2014_000000393230.jpg	Did someone forget his luggage in the snow?	no	none of the above
48	48	COCO_train2014_000000393230.jpg	What color is his coat?	blue	what color is
49	49	COCO_train2014_000000393230.jpg	Is it daytime?	yes	is it

Figure 4: Dataset2

	index	image_name	question	answers	question_type
147692	150746	COCO_train2014_000000312935.jpg	Is there wine glasses?	no	is there
86695	88502	COCO_train2014_000000418198.jpg	What is in the bowl?	oranges	what is in the
298345	304535	COCO_train2014_000000138234.jpg	Where is the elephant in this photo?	water	where is the
243058	248130	COCO_train2014_000000209967.jpg	Is there a stove?	no	is there a
147921	150976	COCO_train2014_000000162908.jpg	Is the bus moving?	no	is the

Figure 5: Dataset3

	index	image_name	question	answers	question_type
8648	22359	COCO_train2014_000000268518.jpg	Is this dog tired?	yes	is this
172469	436295	COCO_train2014_000000390951.jpg	Are the mirrors gold-plated?	no	are the
135916	343382	COCO_train2014_000000362140.jpg	Does she have long hair?	yes	none of the above
130102	329119	COCO_train2014_000000234378.jpg	Is this a fighter jet?	no	is this a
46863	119706	COCO_train2014_000000529929.jpg	Is one plate healthier than the other one?	yes	is

Figure 6: Dataset4

2.5 DataLoaders

(1) **Dataset Initialization:** The dataset is designed to handle both images and text, using a structured dataset for questions and answers, along with an associated image dataset. The dataset class also includes tools for processing text and images.

(2) **Data Retrieval and Preprocessing:** For each data point, the relevant image and corresponding question-answer pair are retrieved. A check ensures consistency between the question in the text dataset and the image dataset. The image is converted to a suitable format, and the question is processed into a format suitable for model input.

(3) **Training and Validation Setup:** Separate datasets are prepared for training and validation, each using a respective subset of the overall data. These datasets are then used to create data loaders that manage batching and ordering of data points during training and validation. The training data is randomized to improve learning, while the validation data maintains its order for consistent evaluation.

3 MODEL ARCHITECTURES

We implement different types of architectures to process and predict answers from question text and image inputs. The models utilized are as follows:

- Transformer model for Text: BERT ('bert-base-uncased')
- Transformer model for Image: ViT ('google/vit-base-patch16-224-in21k')
- Tokenizers: BERT Tokenizer and ViT Tokenizer
- Embedders: BERT Embedder and ViT Embedder

The details of these architectures are as follows:

3.1 Model1 : Transformer Encoder Model

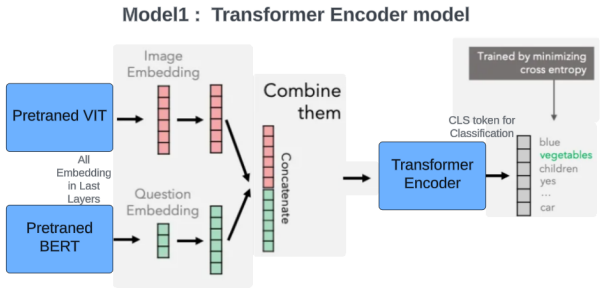

Figure 7: Model1 : Transformer Encoder Model Architecture

- Number of Transformer Layers: 1
- Number of Attention Heads: 12
- Optimizer Used in Training: Adam
- Loss used in Backpropagation: Cross-entropy Loss
- Task to predict answer: Multi-class Classification over answer dictionary

(1) Pretrained BERT and ViT transformers are used to obtain text and image embeddings of dimensions 768. Both questions and answers are embedded using the BERT model.

(2) The embeddings of questions and images from the last layers of BERT and ViT, respectively, are concatenated pairwise (i.e., the first question with the first image, and so on) and fed into the next layer of Transformer Encoder layers.

(3) The task of predicting answers is framed as a multi-class classification problem over the entire answers dictionary, where each answer corresponds to one class.

(4) The Transformer Encoder [CLS] token is used to generate a softmax over the answer dictionary, and CrossEntropyLoss is employed for backpropagation.

```
class VQAModel(nn.Module):
    def __init__(
        self,
        dim_model = 768,      # image and text embeddings concatenated
        nhead = 12,                    # No. of Attention heads
        num_layers = 1,               # No. of encoder layers
        num_classes = 8000
    ):
        super().__init__()
        self.text_embedder = TextEmbedding()
        self.image_embedder = ImageEmbedding()

        encoder_layers = TransformerEncoderLayer(d_model=dim_model, nhead=nhead)
        self.transformerEncoder = TransformerEncoder(encoder_layer=encoder_layer

        self.classifier = nn.Linear(dim_model, num_classes).to(device)
        self.softmax = nn.Softmax(dim=1)
```

Figure 8: Model1 : Transformer Encoder Model

3.2 Model2 : Transformer Encoder - Decoder Model

- Number of Transformer Layers: 1

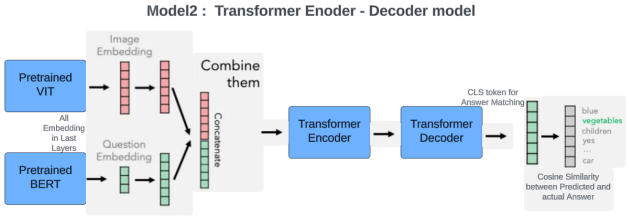

Figure 9: Model2 : Transformer Encoder - Decoder Model Architecture

- Number of Attention Heads: 12
- Optimizer Used in Training: Adam
- Loss used in Backpropagation: Cosine Similarity Loss
- Task to predict answer: Matching predicted embedding with actual Embedding using Cosine Similarity

(1) Pretrained BERT and ViT transformers are utilized to obtain text and image embeddings, each with dimensions of 768. Both questions and answers are embedded using the BERT model.
(2) The embeddings of questions and images from the final layers of BERT and ViT, respectively, are concatenated pairwise and fed into the subsequent layer of Transformer Encoder-Decoder layers.
(3) The task of predicting the answer is formulated as maximizing the cosine similarity between the predicted embedding and the actual answer embedding (same as masking approach in BERT).
(4) The [CLS] token from the Transformer Decoder is used to determine the maximum cosine similarity with the answers from the answer dictionary. The answer with the highest similarity is predicted as the final output.

```
class VQAModel(nn.Module):
    def __init__(
        self,
        dim_model = 768,      # image and text embeddings concatenated
        nhead = 12,                  # No. of Attention heads
        num_layers = 1,              # No. of encoder layers
        num_classes = 8000
    ):
        super().__init__()
        self.text_embedder = TextEmbedding()
        self.image_embedder = ImageEmbedding()

        self.transformer = Transformer(num_encoder_layers=1, num_decoder_layers=1, nhead=nhead,
l).to(device)
```

Figure 10: Model2 : Transformer Encoder - Decoder Model

3.3 Model3 : MLP Model

- Number of MLP Layers: 1
- Optimizer Used in Training: Adam
- Loss used in Backpropagation: Cross-entropy Loss
- Task to predict answer: Multi-class Classification over answer dictionary

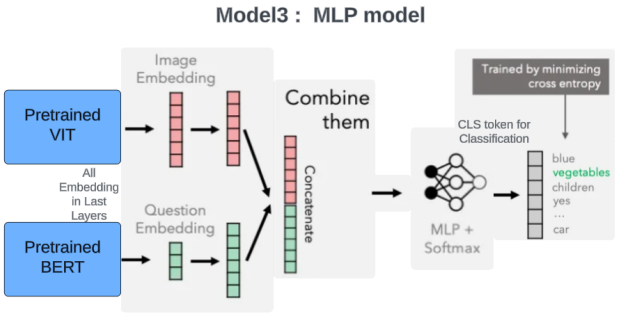

Figure 11: Model3 : MLP Model Architecture

(1) Pretrained BERT and ViT transformers are used to obtain text and image embeddings of dimensions 768. Both questions and answers are embedded using the BERT model.
(2) The embeddings of questions and images from the last layers of BERT and ViT, respectively, are concatenated pairwise and fed into a MultiLayer Perceptron (MLP) layer.
(3) The task of predicting answers is framed as a multi-class classification problem over the entire answers dictionary.
(4) The last layer of the MLP contains nodes equal to the number of classes, with each node corresponding to one class/answer from the answer dictionary. A softmax function is applied at the last layer, and CrossEntropyLoss is used for backpropagation.

```
class VQAModel(nn.Module):
    def __init__(
        self,
        dim_model = 768,      # image and text embeddings concatenated
        nhead = 12,                  # No. of Attention heads
        num_layers = 1,              # No. of encoder layers
        num_classes = 8000
    ):
        super().__init__()
        self.text_embedder = TextEmbedding()
        self.image_embedder = ImageEmbedding()

        self.mlp = nn.Sequential(
            nn.Linear(self.text_embedder.config.hidden_size + self.image_embedde
            nn.ReLU(),
            nn.Dropout(0.4),
        )

        self.classifier = nn.Linear(dim_model, num_classes).to(device)
        self.softmax = nn.Softmax(dim=1)
```

Figure 12: Model3 : MLP Model

3.4 Model4 : Co-Attention Model

- Number of Transformer Layers: 1
- Number of Attention Heads: 12
- Optimizer Used in Training: Adam
- Loss used in Backpropagation: Cross-entropy Loss
- Task to predict answer: Multi-class Classification over answer dictionary

(1) The model design is inspired by the VilBert model and incorporates skip-connections to enhance performance.

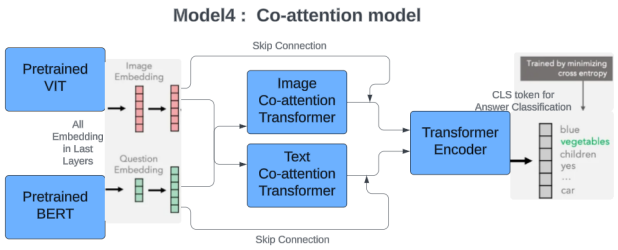

Figure 13: Model4 : Co-Attention Model Architecture

(2) Pretrained BERT and ViT transformers are used to obtain text and image embeddings of dimensions 768. Both questions and answers are embedded using the BERT model.
(3) Two separate transformers are used for co-attention. The image co-attention transformer matches its queries with the keys and values of the question co-attention transformer, and vice versa.
(4) A final transformer layer concatenates these two embeddings and predicts the answer from the answer dictionary, framing it as a multi-class classification problem where each answer corresponds to one class.
(5) Skip connections are used from the pretrained BERT to the text co-attention transformer and from the pretrained ViT to the image co-attention transformer. These skip connections help prevent the vanishing gradient problem and leverage pretrained weights to obtain better embeddings.

```
class ViLBERTModel(torch.nn.Module):
    def __init__(
        self,
        dim_model=768,      # image and text embeddings concatenated
        nhead=12,                   # No. of Attention heads
        num_layers=1,               # No. of encoder layers
        num_classes=8000
    ):
        super().__init__()
        self.dim_model = dim_model
        self.text_embedder = TextEmbedding()
        self.image_embedder = ImageEmbedding()

        self.query = nn.Linear(self.dim_model, self.dim_model).to(device)
        self.key = nn.Linear(self.dim_model, self.dim_model).to(device)
        self.value = nn.Linear(self.dim_model, self.dim_model).to(device)

        self.co_attention_im = nn.MultiheadAttention(dim_model, nhead).to(device)
        self.co_attention_text = nn.MultiheadAttention(dim_model, nhead).to(device)

        self.transformer_encoder_text_1 = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model=dim_model, nhead=nhead),
            num_layers=num_layers
        ).to(device)

        self.transformer_encoder_image_1 = nn.TransformerEncoder(
            nn.TransformerEncoderLayer(d_model=dim_model, nhead=nhead),
            num_layers=num_layers
        ).to(device)
```

Figure 14: Model4 : Co-Attention Model

4 LORA (LOW RANK ADAPTATION)

LORA technique is used on baseline models and different experiments are performed on Rank and Target_modules.

- LORA reduces trainable parameters by decomposing weight matrices into lower-dimensional representations.
- Original weight matrix $W \in \mathbb{R}^{m \times n}$ is approximated as $A \in \mathbb{R}^{m \times r}$ and $B \in \mathbb{R}^{r \times n}$, with $r \ll \min(m, n)$.
- Experiments are conducted varying the rank r and target modules to balance accuracy and computational efficiency.
- LORA maintains high model performance while reducing computational and memory requirements.

```
# Define the LoRA configuration
LORA_R = 8
LORA_ALPHA = 512
LORA_DROPOUT = 0.05

lora_config = LoraConfig(
    r=LORA_R,
    lora_alpha=LORA_ALPHA,
    lora_dropout=LORA_DROPOUT,
    bias="none",
    target_modules=[
        "transformerEncoder.layers.0.linear1",
        "transformerEncoder.layers.0.linear2",
        "image_embedder.image_model.encoder.layer.11.intermediate.dense",
        "image_embedder.image_model.encoder.layer.11.output.dense"
    ]
#     target_modules = linear_layers
)
```

Figure 15: LORA Configuration

The following table provides a comparison of the number of parameters between models without LoRA and with LoRA:

```
Model: baseline ----> trainable params: 207661281 || all params: 207661281 || trainable%: 1.0
Model: lora-4 ----> trainable params: 53,248 || all params: 207,714,529 || trainable%: 0.0256
Model: lora-8 ----> trainable params: 106,496 || all params: 207,767,777 || trainable%: 0.0513
Model: lora-16 ----> trainable params: 212,992 || all params: 207,874,273 || trainable%: 0.1025
Model: lora-32 ----> trainable params: 425,984 || all params: 208,087,265 || trainable%: 0.2047
```

Figure 16: Comparison of Number of Parameters

5 OPTIMIZATIONS

5.1 QLORA - Quantization

QLORA is an optimization technique that extends the Low-Rank Adaptation (LORA) method by incorporating quantization. While LORA reduces the number of trainable parameters by decomposing the weight matrices into lower-dimensional representations, QLORA further enhances this approach by applying quantization to these lower-dimensional matrices.

QLORA results in significant reductions in both memory footprint and computational requirements, allowing for more efficient training and inference without substantial loss in model performance.

6 EXPERIMENTS AND RESULTS

6.1 Setup and Environment

The experiments were conducted on a Kaggle platform utilizing a Tesla P100 GPU environment across seven user accounts. Due to weekly GPU credit limitations, exhaustive exploration of LoRA ranks and layers for all high-performing models was a hard feat to achieve. Consequently, we opted for standard LoRA configurations (rank 8 and 16) for comparison with those models. All experiments were executed for 3 epochs with a batch size of 64.

```python
import torch
from transformers import BitsAndBytesConfig, AutoModel, ViTModel
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training

# Define the BitsAndBytesConfig for 4-bit quantization
config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16,
)

# Load the pre-trained BERT model with quantization config
bert_model = AutoModel.from_pretrained("bert-base-uncased", quantization_config=config)

# Load the pre-trained ViT model with quantization config
vit_model = ViTModel.from_pretrained("google/vit-base-patch16-224-in21k", quantization_config=conf

# Prepare the models for k-bit training
bert_model = prepare_model_for_kbit_training(bert_model)
vit_model = prepare_model_for_kbit_training(vit_model)
```

Figure 17: QLORA Configuration

6.2 Expeiments with Models

6.2.1 *Model1: Transformer Encoder Model* .

(1) **Experiments on Dataset3 (8161 Answers in answer dictionary)**:
- We used an architecture (Model 1) that leverages the semantic information from pre-trained BERT and ViT for question-image tasks. The final layers of both models are concatenated and fed into a transformer encoder for joint processing.
- Classification is performed using the CLS token of the final transformer layer. We evaluate Model 1 with various fine-tuning methods, including no adaptation (baseline), standard LoRA with different ranks and layers, and QLoRA (refer to Tables 3 and 5). While the baseline achieved marginally superior accuracy, LoRA for linear layers only exhibited faster training times with comparable performance (Table 3).
- Notably, this approach outperformed LoRA applied to all layers (rank 8), highlighting its efficiency and effectiveness. These findings suggest that LoRA for linear layers offers a compelling trade-off between accuracy and training speed.

(2) **Experiments on Dataset4 (Yes-No type answers)**:
- We initially explored fine-tuning Model 1 with and without LoRA (refer to Table 2) for the yes-no dataset. However, both approaches yielded underwhelming performance.
- The baseline model achieved comparable performance to Model 1 with better precision and faster training time. Consequently, we opted to discontinue experiments with different LoRA ranks and layers, focusing on alternative fine-tuning strategies.

Model	Train Time	F1	Precision	Recall	Accuracy (out of 1)
Model1 (baseline)	3.626	**9.915e-5**	**5.931e-5**	**3.019e-4**	**0.19644**
Model1 (lora-8)	3.067	9.640e-5	5.736e-5	3.017e-4	0.19009
Model1 (lora-16)	3.176	9.640e-5	5.736e-5	3.017e-4	0.19010
Model1 (lora-32)	3.356	9.908e-5	5.927e-5	3.017e-4	0.19643
Model1 (lora-All-linear-layers-rank-8)	1.575	9.453e-5	5.601e-5	3.014e-4	0.19310
Model1 (lora-layers-2)	1.61	9.233e-5	5.557e-5	3.017e-4	0.19421

Table 1: Baseline vs LoRA for Model1 (8161) dataset

Model	Train Time (in hrs)	F1	Precision	Recall	Accuracy (out of 1)
Model1 (baseline)	3.626	**0.5538**	0.5632	**0.5448**	**0.5511**
Model1 (lora-8)	**2.29**	0.5483	**0.5830**	0.5211	0.5422

Table 2: Baseline vs LoRA for Model1 (yes-no dataset)

6.2.2 *Model2: Transformer Encoder - Decoder Model*.

(1) We pivoted our approach after observing limited success with the previous transformer/encoder models (refer to Table 3). To focus on semantic understanding of the data, we employed a sequence-to-sequence architecture with a transformer encoder-decoder pair.
(2) This model generates a sequence of tokens, and the loss is calculated using cosine similarity between the generated and target sequences.
(3) Notably, this approach can handle variable-length outputs, unlike the previous models. This architecture achieved significant performance gains, with the cosine similarity loss dropping below 1e-4 within the first epoch for no adaption (without LoRA).
(4) As this is a generative task, we transitioned to different evaluation metrics beyond classification accuracy. Given the promising results, we further explored the impact of LoRA with various rank values (refer to Table 6).

Model	Train Time (in hrs)	Avg. Cosine Similarity	Avg. Correlation	Avg. Euclidean Distance	Avg. Manhattan Distance
Model2 (baseline)	2.39	**0.9993**	**0.9992**	18874	434319
Model2 (LoRA 8)	**1.39**	0.5385	0.5380	24556	567850
Model2 (LoRA 16)	1.44	0.8403	0.8405	20350	411413
Model2 (LoRA 8-all linear layers)	1,55	0.9992	0.9989	**9338**	**206278**

Table 3: Baseline vs Model3 (MLP Model) on dataset3 (8161 classes)

6.2.3 *Model3: MLP Model* .

(1) In Model1 and Model2, a Transformer architecture was employed to predict the answers.
(2) However, the simple encoder layer in Model1 did not yield the expected results. Consequently, we utilized a Multilayer Perceptron (MLP) to perform the multi-class classification task.
(3) The final layer in the MLP contains nodes equal to the number of classes/answers in the answer dictionary. The model incorporates the ReLU activation function and the concept of dropout.
(4) This MLP model (Model3) achieved superior accuracy compared to Model1.

Model	Train Time (in hrs)	F1	Precision	Recall	Accuracy (out of 1)
Model1 (baseline)	3.626	**9.915e-5**	**5.931e-5**	**3.019e-4**	**0.19644**
Model3 (MLP Model)	2.553	0.03751	0.052	0.028	**0.33009**

Table 4: Baseline vs Model3 (MLP Model) on dataset3 (8161 classes)

6.2.4 ***Model4: Coattention Model*** .

(1) Due to the limitations of other encoder based and linear models (refer to previous sections), we explored a co-attention mechanism with skip connections to enhance interaction between image features and text embeddings.
(2) We hypothesized that large input patches might be hindering attention to text information. The model employed linear layers to generate queries, keys, and values for both image and embedding pairs, enabling the text transformer to attend to image features and vice versa.
(3) While this co-attention layer utilized two transformers, leading to increased training time, the model unfortunately did not exhibit significant improvements in any evaluation metric.
(4) Consequently, further experimentation with this approach was discontinued.

Model	Train Time	F1	Precision	Recall	Accuracy (out of 1)
Model1 (baseline)	**3.626**	**9.915e-5**	**5.931e-5**	**3.019e-4**	**0.19644**
Model4	3.667	9.320e-5	5.678e-5	3.011e-4	0.18780

Table 5: Model1(baseline) vs Model4 (8161) dataset

6.3 Experiments with LoRA and other Optimizations

6.3.1 ***Ranks***. Experiments on Rank in LORA Configuration are performed as follows :

- Rank=4: Low-rank approximation with rank 4.
- Rank=8: Low-rank approximation with rank 8.
- Rank=16: Low-rank approximation with rank 16.
- Rank=32: Low-rank approximation with rank 32.

for all above rank experiments : Target Modules =

- `transformerEncoder.layers.0.linear1`
- `transformerEncoder.layers.0.linear2`
- `image_embedder.image_model.encoder.layer.11.intermediate.`
- `image_embedder.image_model.encoder.layer.11.output.dense`

6.3.2 ***Layers*** . LORA is performed over different set of Layers of models as follows :

1. Some linear Layers + Image Embedder Layers :
Target Modules =

- `transformerEncoder.layers.0.linear1`
- `transformerEncoder.layers.0.linear2`
- `image_embedder.image_model.encoder.layer.11.intermediate.dense`
- `image_embedder.image_model.encoder.layer.11.output.dense`

2. All Linear Layers + Image Embedder Layers :
Target Modules =

- `transformerEncoder.layers.all_linear`
- `image_embedder.image_model.encoder.layer.11.intermediate.`
- `image_embedder.image_model.encoder.layer.11.output.dense`

3. Image Embedder Layers + Text Embedder Layers + Classifier Layers:
Target Modules =

- `image_embedder.image_model.encoder.layer.11.intermediate.`
- `image_embedder.image_model.encoder.layer.11.output.dense`
- `text_embedder.text_model.encoder.layer.11.intermediate.dense`
- `text_embedder.text_model.encoder.layer.11.output.dense`
- `classifier`

Model	Train Time	F1	Precision	Recall	Accuracy (out of 1)
Model1 (baseline)	**3.626**	**9.915e-5**	**5.931e-5**	**3.019e-4**	**0.19644**
Model1 (LoRA)	3.667	9.320e-5	5.678e-5	3.011e-4	0.18780
Model1 (QLoRA)	**3.500**	9.200e-5	5.500e-5	3.000e-4	0.18500

Table 6: QLoRA vs LoRA vs Baseline (8161 dataset)

6.4 Training Loss for Models

The training loss for the following models was generated using TensorBoard. Various comparisons were conducted to analyze the performance of the models. Each model was trained for three epochs.

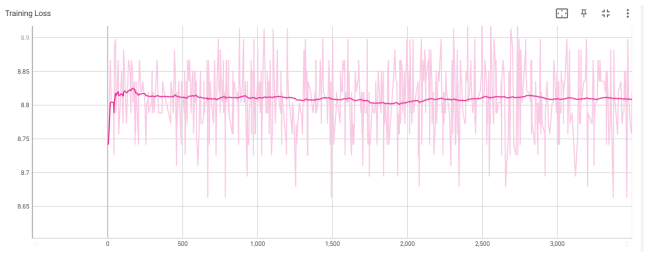

Figure 18: Baseline Model

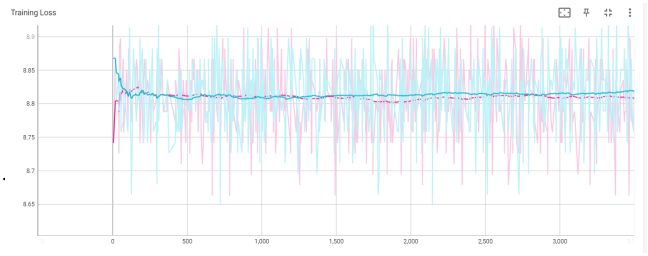

Figure 19: Baseline model(pink) vs Lora with rank = 4(blue)

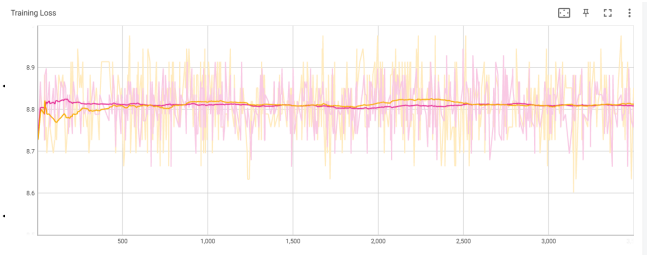

Figure 20: Baseline model(pink) vs Lora with rank = 8(orange)

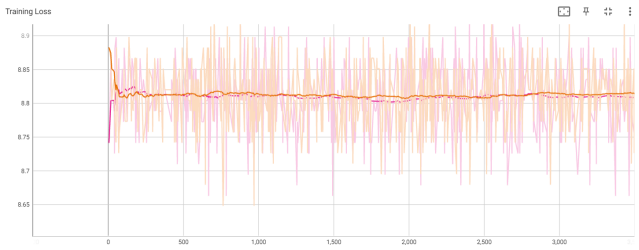

Figure 21: Baseline model(pink) vs Lora with rank = 16(orange)

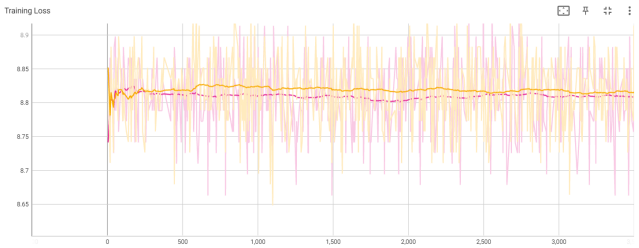

Figure 22: Baseline model vs Lora with rank = 32

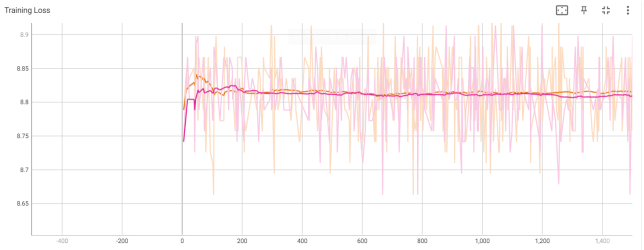

Figure 23: Baseline model vs Co-attention model (Model4)

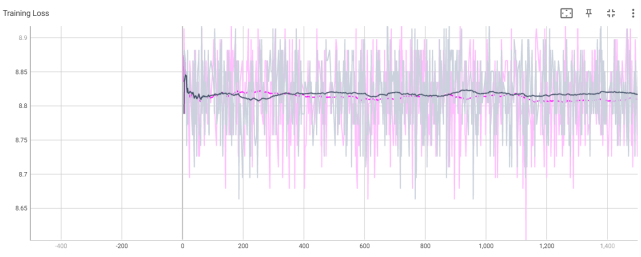

Figure 24: Lora with all linear layers vs Lora with Classification layers

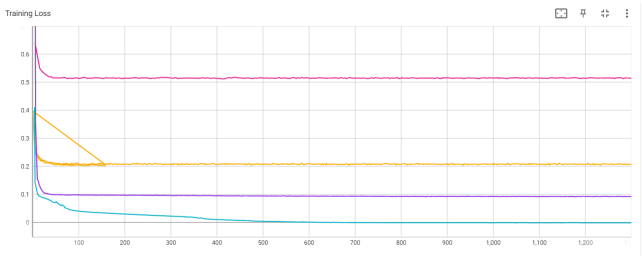

Figure 25: Encoder - Decoder Model (Model2) with baseline(light blue), lora-all-layer(violet), lora-16(orange), lora-8(pink)

7 EXAMPLE INFERENCES

Below are example inferences from various models :

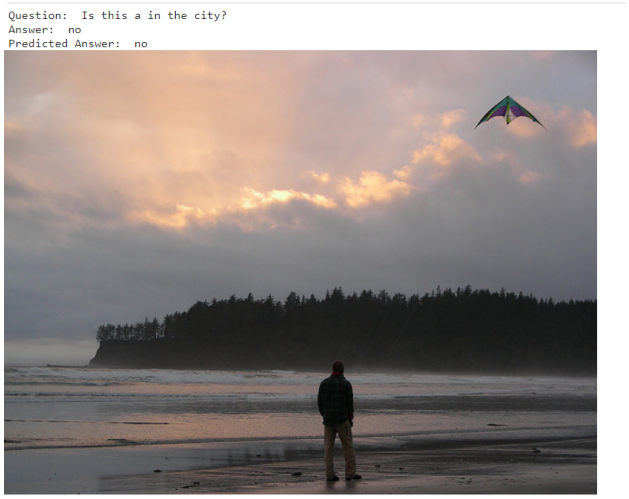

Figure 26: Example Inference of Model1 (baseline) on 8161 dataset

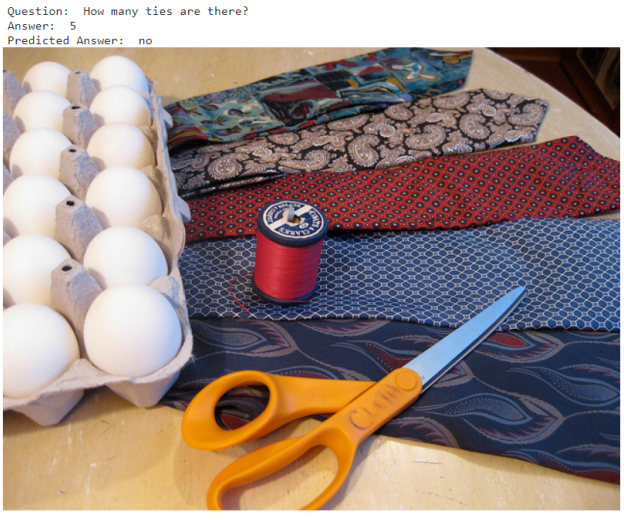

Figure 27: Example Inference of Model1 (baseline) on 8161 dataset

Figure 28: Example Inference of Model1 (baseline) on Yes-no dataset

Figure 29: Example Inference of Model1 (baseline) on Yes-no dataset

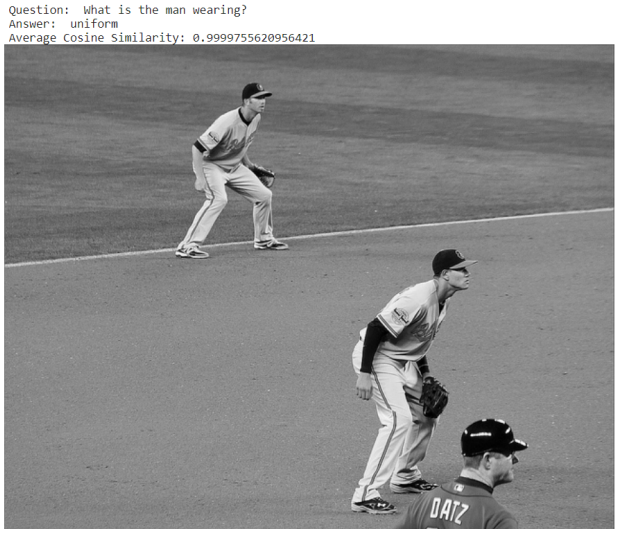

Figure 30: Example Inference of Model2 (baseline) on 8161 dataset

Figure 31: Example Inference of Model2 (baseline) on 8161 dataset

8 CONCLUSION

- The MLP model (Model3) produced better results compared to the baseline model (Model1).
- The encoder-decoder model (Model2) significantly reduced the training loss.
- Applying LoRA for all Linear layers shows comparable performance with the baseline, with significantly lesser time.
- LoRA optimization reduced the number of model parameters to 1 to 2 percent of their original count.
- LoRA optimization also decreased the training time by approximately 15 minutes per epoch due to the reduction in the number of model parameters.
- QLoRA optimization stores weights in low-precision bytes, which reduces memory storage requirements at the cost of a slight decrease in accuracy.