COL 774: Assignment 4

Semester II, 2024-25

Marks - 75 + 5 (Bonus)

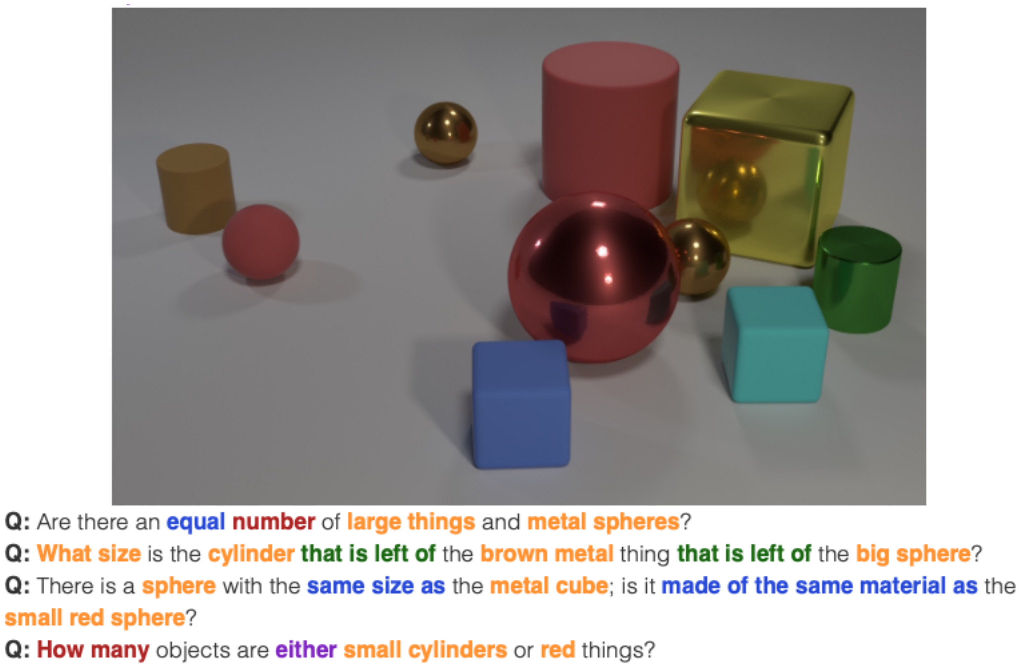
Deadline: Friday May 9, 2025, 11.59 PM

Notes:

- Upload code and a **write-up (pdf)** file, one (consolidated) for each part, which includes a brief description for each question explaining what you did. Include any observations and/or plots required by the question in this single write-up file.
- This assignment has two implementation questions.
- Do not submit the datasets.
- You should use Python and PyTorch as your programming language.
- Your code should have appropriate documentation for readability.
- You will be graded based on what you have submitted as well as your ability to explain your code.
- Refer to the course website for assignment submission instructions.
- This assignment can be done in pairs.
- No buffer days in this assignment.
- Run your code on GPU for faster training.
- We plan to run Moss on the submissions. We will also include submissions from previous years since some of the questions may be repeated. Any cheating will result in a zero on the assignment, a penalty of -10 points and possibly much stricter penalties (including a **fail grade** and/or a **DISCO**).

Visual Question Answering

In this problem, we will work on the task of Visual Question Answering (VQA). In Visual question answering, the objective is to answer (textual) question(s) in reference to an underlying image. For doing so, we will build a multi-modal transformer model which takes image and questions as input and predicts the correct answer for the corresponding question.



Q: Are there an **equal number** of **large things** and **metal spheres**?
Q: **What size** is the **cylinder that is left of** the **brown metal** thing **that is left of** the **big sphere**?
Q: There is a **sphere** with the **same size as** the **metal cube**; is it **made of the same material as** the **small red sphere**?
Q: **How many** objects are **either small cylinders** or **red** things?

Figure 1: A sample image and corresponding question pairs from the CLEVR dataset. The questions in CLEVR are designed to evaluate diverse aspects of visual reasoning, including attribute identification, counting, comparison, spatial relationships, and logical operations (Source: CLEVR).

1 Dataset

For this assignment, we will be using the CLEVR dataset. The CLEVR dataset consists of synthetic images paired with corresponding question-answer pairs (see Figure 1 for examples of the types of questions). The images contain objects that vary in size, shape, color, and material. The questions are designed to require a range of visual reasoning skills, including attribute recognition, spatial localization, counting, and comparison. For more details, please refer to the original paper.

You can download the dataset from here. The dataset contains two main folders: `images` and `questions`. Each of these folders further contains sub-folders containing either **A** or **B** variants of the data. For the core of the problem (Parts 8, 9, 10), we will use the **A** variant of the dataset which has `trainA`, `valA`, and `testA` subsets, along with the corresponding question files: `CLEVR_trainA_questions.json`, `CLEVR_valA_questions.json`, and `CLEVR_testA_questions.json`. Part 11 will make use of **B** variant for evaluation (zero-shot-transfer) while using the model trained on **A**.

2 Data Processing

We begin by processing the dataset through two main steps: **(1) tokenization** and **(2) handling variable-length questions**. In the first step, we apply tokenization, where each question is converted into a sequence of tokens, and each token is then mapped to a unique numerical ID. This can be achieved using a custom tokenizer or a pre-trained one, such as the BERT tokenizer (`bert-base-uncased`) provided by the HuggingFace Transformers library. The second step addresses the variable lengths of questions. To ensure that all inputs have a consistent shape, we determine the maximum question length in the dataset and apply padding to shorter sequences. This standardization is essential for efficient batch processing and compatibility with transformer-based architectures.

3 Network Architecture

Our architecture is inspired by the Transformer model introduced in the *CLEVR* paper, with a key variation: instead of using an LSTM for language encoding, we adopt the Transformer architecture as proposed in *Attention is All You Need*.

The model is composed of four main components: an image encoder, a language encoder, a feature fusion module, and a decoder. Given an image-question pair, the image encoder extracts visual features from the image, while the language encoder encodes the question into a sequence of text features using a Transformer. These modality-specific features are then integrated using a feature fusion module that employs cross-attention to learn a joint representation. This fused representation is subsequently passed to the decoder, which serves as a classifier to predict the correct answer class. Following sections provide deatils of each of the four components

4 Image Encoder

For the vision encoder, we use `ResNet101`, which can be loaded from `torchvision.models`. Initialize the model with `resnet101(pretrained=True)` and remove the final global average pooling and fully connected layers. This results in a feature map of shape $[B, 2048, h, w]$, where B denotes the batch size, and h and w represent the height and width of the extracted feature map, respectively. To obtain the final image feature representation, apply a linear projection using `nn.Linear(2048,` $d^v_{\texttt{embed}}$`)`, where d^v_{embed} is the dimensionality of the image feature embedding. For our setup, we set $d^v_{\text{embed}} = 768$. Since we use ResNet101 purely for feature extraction, we keep its parameters frozen during training by setting `resnet.requires_grad = False`.

5 Text Encoder

For the text encoder, we adopt the architecture proposed in the *Attention is All You Need* paper. We begin by constructing an embedding layer using `nn.Embedding` to learn token representations, where the dimensions are set to the vocabulary size and an embedding dimension (`embed_dim`) of 768. Additionally, append a learnable `[CLS]` token to the word embedding as defined in BERT. While the original Transformer architecture uses fixed sinusoidal positional embeddings, we instead make the positional embeddings learnable. This is implemented using `nn.Parameter(torch.randn(1, max_len, embed_dim))`, allowing the model to optimize positional information during training. We then stack 6 Transformer encoder layers using `nn.TransformerEncoder`, each with 8 attention heads (`num_layers=6`, `nhead=8`). This Transformer-based text encoder captures contextual dependencies between tokens and outputs a embedded feature representing d^t_{embed} the input question.

6 Feature Fusion: Cross Attention

For feature fusion, we employ a `cross-attention mechanism` that enables the text features to attend to relevant spatial regions within the image. To facilitate cross-attention, we ensure that the text and image feature embeddings share the same dimension, i.e. $d^t_{\text{embed}} = d^v_{\text{embed}} = 768$.

In this setup, the text features act as the `query`, while the image features serve as the `keys` and `values`. The cross-attention is implemented using `nn.MultiheadAttention(embed_dim=768, num_heads=8)`. Pass the `[CLS]` (i.e 0th index of output), to the decoder.

7 Classifier

To predict the final answer, we use a decoder composed of a simple two-layer multilayer perceptron (MLP). The MLP first maps the joint feature representation from dimension d_{embed} (i.e., 768) to 500, applies a ReLU activation, and then maps from 500 to the number of answer classes like, `Linear(`d_{embed}`, 500)` $\rightarrow$ `ReLU` $\rightarrow$ `Linear(500, num_classes)`.

8 Training and Evaluation - 40 marks

a) Train the entire model in an end-to-end fashion. Clearly specify the implementation details, including the learning rate, batch size, optimizer used, and number of training epochs. You can use Adam optimizer and cross-entropy as loss function.

b) Plot train and validation loss curves and training and validation accuracy. Comments on under-fitting or overfitting.

c) Evaluate and report accuracy, precision, recall and F1-score on `testA`.

d) Visualize predictions made by your best model on 5 image-question pairs. Plot image, question, predicted answer and ground truth answer.

e) Repeat the above part by visualizing the errors cases, where predicted answers and ground answers do not match. Comment yours observations.

9 Fine-tuning image encoder - 10 marks

Until now, the image encoder (ResNet101) has been kept frozen, meaning its weights were not updated during training. In this section, we fine-tune the image encoder(along with other modules) on the CLEVR dataset to enhance the quality of extracted visual features. Specifically, take the best-performing model from section 8 and unfreeze the image encoder by setting `resnet.requires_grad = True`. **Note that there is no need to retrain the model from scratch—continue training from the existing checkpoint.**

Evaluate the fine-tuned model and report the results in the same manner as in Section 8, including metrics and observations.

10 Further Enhancement - 10 marks + 5 - Bonus

You can further improve the performance of the model using different training techniques or modifying the loss functions. We have proposed two such enhancement below. You can try any one or both of them and report the improvement in the performance of the model. **(Note: +10 if either of the following methods works, with an additional +5 if both methods are successful.)**

a) Use the best model weights trained above (sec 1.8), and further train the model using **Focal Loss**. Report your observations as done in section 8.

b) Using the best model till now, initializing the word embedding layer (nn.Embedding) layer with BERT embedding, instead of random initialization. Report your observations as done in section 8.

11 Zero Shot Evaluation - 15 marks

In this section, we evaluate the model's ability to generalize across variations in object color and shape. To do this, we use the type B variant of the dataset. In type B, cubes are colored red, green, or purple, in contrast to the gray, blue, brown, or yellow cubes in type A. Similarly, cylinders in variant B are gray, blue, brown, or yellow, whereas in variant A they are red, green, purple, or cyan.

a) **Transfer Task:** Evaluate the best model trained on the type A dataset on the type B dataset *without any additional training*. Report the classification accuracy, precision, recall, and F1 score on `testB`. How does it compare with performance on `testA`?

b) **Qualitative analysis:** Visualize example predictions for `testB`. For each case, show the image, the input question, the predicted answer, and the ground truth answer. Analyze the types of errors made by model and comment on your observations.

Submission Instructions

1. Submit two Python files: `part1.py` and `part2.py`. Each script must support the following command-line arguments:

   - **For training:**
     ```
     python partX.py --mode train --dataset <path_to_dataset>
     --save_path <destination_path>
     ```
   - **For inference:**
     ```
     python partX.py --mode inference --dataset <path_to_dataset>
     --model_path <path_to_model>
     ```

   Replace `partX.py` with either `part1.py` or `part2.py` as applicable.

2. Submit a single consolidated report covering both parts in PDF format. Name the report as: `Entry_Number_A4.pdf`.

3. **Upload the best-performing models for each part (8, 9, 10 and 11) to Google Drive**, and include the shared links in your report. Additionally, include a screenshot of the Google Drive folder showing the uploaded files along with their timestamps as proof of submission.

Training Instructions

1. If you are running the code on an HPC cluster without internet access, you can pre-download the BERT model and tokenizer locally using the following code:

```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = BertModel.from_pretrained("bert-base-uncased")

tokenizer.save_pretrained("local_bert")
model.save_pretrained("local_bert", safe_serialization=False)
```

Transfer the entire local_bert folder to the HPC environment. Then, load the model and tokenizer from the local directory as follows:

```
tokenizer = BertTokenizer.from_pretrained("local_bert")
model = BertModel.from_pretrained("local_bert")
```

2. When you are training on HPC, submit batch jobs (refer:HPC).

3. Save intermediate checkpoints during training. Specifically, save the model whenever there is an improvement in validation loss or validation accuracy. This helps ensure that the best-performing model is retained even if later epochs do not yield better results.

4. Tutorials for Pytorch :
   (a) Reference 1
   (b) Reference 2

Reference

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is All You Need.* Advances in Neural Information Processing Systems, 30. https://arxiv.org/pdf/1706.03762

2. Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). *CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning.* Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://arxiv.org/pdf/1612.06890

3. Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 4171–4186. https://aclanthology.org/N19-1423/

4. Karpathy, A. (2019). *A Recipe for Training Neural Networks.* https://karpathy.github.io/2019/04/25/recipe/

5. PyTorch Documentation. *TransformerEncoder.* https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoder.html

6. PyTorch Documentation. *MultiheadAttention.* https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html

7. torchvision.models. *ResNet50.* https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html