Assigment 3: Image Change Captioning

This package contains the code for **image change captioning**. The image change captioning task require a model to compare two images and describe what is changed.

In this assignment, you'll need to

1. Implement several code snippets, including
   - InfoNCE Loss Function
   - Position Encoding in Transformer
   - Attention in Transformer
2. Run the training and evaluation scripts on your own, on a small subset of CLEVR-CHANGE dataset.

1. Methodology

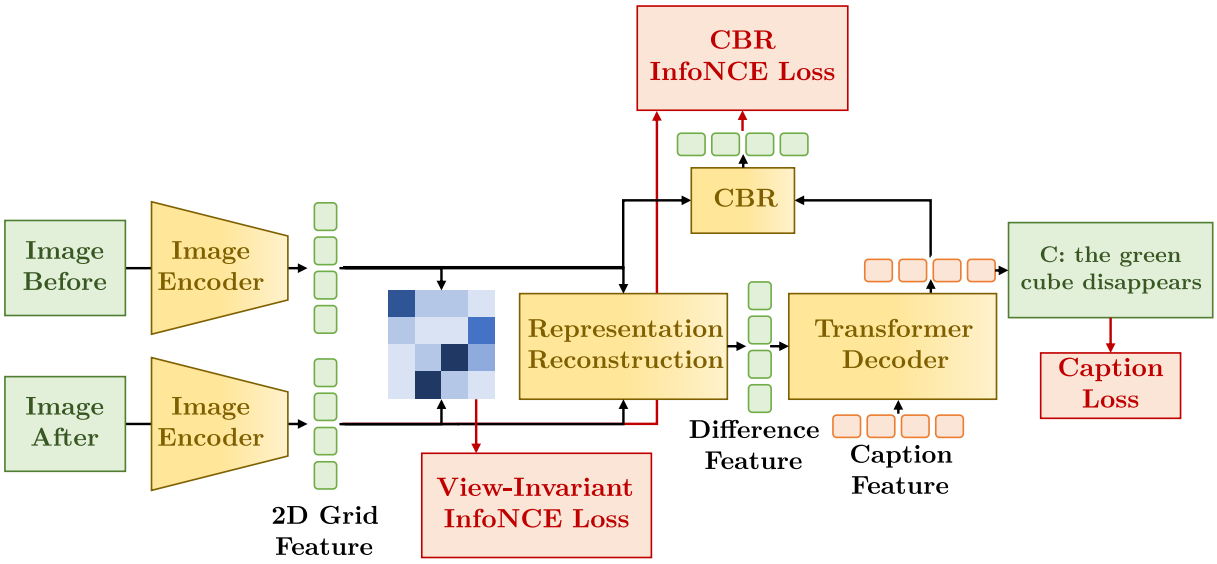

Figure 1: Method Overview

We briefly summarize the used methodology here, as illustrated in Figure 1:

1. **Cross-view Image Pair Encoding:**
   - Utilize a pre-trained CNN to encode "before" and "after" images (I_{bef} and I_{aft}) into grid features X_{bef} and X_{aft}, with $X \in \mathbb{R}^{C\times H\times W}$.
   - Project both representations into a low-dimensional space: $\widetilde{X}_o = \text{conv}_2(X_o) + \text{pos}(X_o)$, where $o \in \{\text{bef}, \text{aft}\}$ and pos is the spatial position encoding.
2. **Self-supervised Cross-view Representation Reconstruction:**
   - Introduce Multi-head Token-wise Matching (MTM) for view-invariant learning.
   - For Token-wise Matching (TM), calculate similarity between query Q and key K:

$$\text{TM}(Q,K) = \frac{1}{2N}\sum_{i=1}^{N}\max_j(e_{ij}) + \frac{1}{2N}\sum_{j=1}^{N}\max_i(e_{ij})$$

     where $e_{ij} = q_i^T k_j$.
   - Extend TM to MTM for fine-grained interaction:

$$\mathrm{MTM}(Q,K) = \mathrm{Concat}_{i'=1}^{h}(\mathrm{head}_{i'})$$
$$\mathrm{head}_{i'} = \mathrm{TM}\left(QW_{i'}^{Q}, KW_{i'}^{K}\right).$$

with each head computed by TM.

3. **View-invariant Representation Learning:**
   - Use MTM for computing similarity in a batch with B image pairs, reshaping $\widetilde{X} \in \mathbb{R}^{D\times H\times W}$ to $\widetilde{X} \in \mathbb{R}^{N\times D}$.
   - Apply the InfoNCE loss for contrastive alignment within the batch:

$$L_{\mathrm{b2a}} = -\frac{1}{B}\sum_{k}^{B}\log\frac{\exp\left(\mathrm{MTM}\left(\widetilde{X}_k^b, \widetilde{X}_k^a\right)/\tau\right)}{\sum_r^B \exp\left(\mathrm{MTM}\left(\widetilde{X}_k^b, \widetilde{X}_r^a\right)/\tau\right)}$$
$$L_{\mathrm{a2b}} = -\frac{1}{B}\sum_{k}^{B}\log\frac{\exp\left(\mathrm{MTM}\left(\widetilde{X}_k^a, \widetilde{X}_k^b\right)/\tau\right)}{\sum_r^B \exp\left(\mathrm{MTM}\left(\widetilde{X}_k^a, \widetilde{X}_r^b\right)/\tau\right)},$$
$$L_{\mathrm{cv}} = \frac{1}{2}(L_{\mathrm{b2a}} + L_{\mathrm{a2b}})$$

   where $\widetilde{X}_k^b$ is the k-th "before" image representation, and $\widetilde{X}_k^a$ is the k-th "after" image representation.

4. **Cross-view Representation Reconstruction:**
   - Use Multi-head Cross-Attention (MHCA) to reconstruct unchanged object representations:

$$\widetilde{X}_{\mathrm{bef}}^{u} = \mathrm{MHCA}\left(\widetilde{X}_{\mathrm{bef}}, \widetilde{X}_{\mathrm{aft}}, \widetilde{X}_{\mathrm{aft}}\right)$$

   and

$$\widetilde{X}_{\mathrm{aft}}^{u} = \mathrm{MHCA}\left(\widetilde{X}_{\mathrm{aft}}, \widetilde{X}_{\mathrm{bef}}, \widetilde{X}_{\mathrm{bef}}\right)$$

   - Integrate unchanged representations into image representations:

$$\widetilde{X}_o^c = \mathrm{LayerNorm}\left(\widetilde{X}_o + \widetilde{X}_o^u\right)$$

   - Integrate into the difference representations:

$$\widetilde{X}_c = \mathrm{ReLU}\left(\left[\widetilde{X}_{\mathrm{bef}}^c; \widetilde{X}_{\mathrm{aft}}^c\right]W_h + b_h\right)$$

   where $[;]$ is the concatenation operation.

5. **Caption Generation:**
   - Translate difference representation into a sentence using a transformer decoder.
   - Calculate probability distributions of target words: $\widetilde{W} = \mathrm{Softmax}\left(\widetilde{H}W_c + b_c\right)$, where $\widetilde{H}$ is the last layer representation of the transformer decoder.

6. **Cross-modal Backward Reasoning (CBR):**
   - Model a "hallucination" representation, push it closer to the "after" representation.
   - Concat the sentence feature $\tilde{T}$ (mean-pooled from $\widetilde{H}$) with hallucinated feature to obtain such hallucination representation

$$\widehat{X}_{\mathrm{hal}} = \mathrm{conv}_2\left(\left[\widetilde{X}_{\mathrm{bef}}; \mathrm{broadcast}_{H\times W}\left(\tilde{T}\right)\right]\right), \widehat{X}_{\mathrm{hal}} \in \mathbb{R}^{D\times H\times W}$$

   - Reconstruct the "after" representation

$$\widetilde{X}_{\text{hal}} = \text{conv}_2\left[\text{MHSA}\left(\widehat{X}_{\text{hal}}, \widehat{X}_{\text{hal}}, \widehat{X}_{\text{hal}}\right)\right]$$

- Use InfoNCE between reconstructed $\widetilde{X}_{\text{hal}}$ and $\widetilde{X}_{\text{aft}}$ for backward reasoning, and $L_{\text{cm}} = \frac{1}{2}(L_{\text{h2a}} + L_{\text{a2h}})$.

7. **Joint Training:**
   - Minimize negative log-likelihood loss for observed word sequence:

$$L_{\text{cap}}(\theta) = -\sum_{t=1}^{m} \log p_\theta(w_t^* \mid w_{<t}^*)$$

   - Optimize final loss function: $L = L_{\text{cap}} + \lambda_v L_{\text{cv}} + \lambda_m L_{\text{cm}}$.

You can also refer to the code for detailed formulations.

2. Environment Configuration

1. Requirement: Linux + NVIDIA GPU.
   - Not tested on Windows/MacOS.
   - Not tested on Google Colab. But you can try to upload the files and run through a notebook. You can run shell commands using `! <command>` in Jupyter Notebook.
2. Make virtual environment with Python 3.8
3. Install PyTorch 1.8. Refer to Installing previous versions of PyTorch.
4. Install requirements (`pip install -r requirements.txt`)
5. Download `en_core_web_sm` english text model for spaCy, by `python3 -m spacy download en_core_web_sm`
6. Setup COCO caption eval tools (github). Or `pip install cocoevalcaps .`

3. Data Preparation

1. Download CLEVR-Change data from here: google drive link, and unzip.[1]

   ```
   tar -xzvf clevr_change.tar.gz
   ```

   Extracting this file will create `data` directory and fill it up with CLEVR-Change dataset.

2. Preprocess data

   - We are providing the preprocessed data here: google drive link. You can skip the procedures explained below and just download them using the following command:

     ```
     cd data
     tar -xzvf clevr_change_features.tar.gz
     ```

   - Extract visual features using ImageNet pretrained ResNet-101:

     ```
     # processing default images
     python scripts/extract_features.py --input_image_dir ./data/images --
     output_dir ./data/features --batch_size 128

     # processing semantically changes images
     python scripts/extract_features.py --input_image_dir ./data/sc_images --
     output_dir ./data/sc_features --batch_size 128

     # processing distractor images
     python scripts/extract_features.py --input_image_dir ./data/nsc_images --
     output_dir ./data/nsc_features --batch_size 128
     ```

- Build vocab and label files using caption annotations:

```
python scripts/preprocess_captions_transformer.py --input_captions_json ./
data/change_captions.json --input_neg_captions_json ./data/
no_change_captions.json --input_image_dir ./data/images --split_json ./data/
splits.json --output_vocab_json ./data/transformer_vocab.json --output_h5 ./
data/transformer_labels.h5
```

4. Code To Implement

In this assignment, you will need to implement some code snippets in the model architecture to finally train the model:

1. InfoNCE Loss. Position:
   - `models/CBR.py, Line 13`
   - `utils/utils.py, Line 292`
   - Note: the two snippets should be identical. You only need to calculate the InfoNCE loss with given unnormalized similarity matrix.
2. Position Encoding. Position:
   - `models/transformer_decoder.py, Line 31`
   - Note: you're going to implement the sinusoidal position encoding as proposed in original Transformer[1] paper. Also refer to this blog for a quick explanation.
3. Attention Mechanisms. Position:
   - Self-attention: `models/transformer_decoder.py, Line 131`
   - Cross-attention: `models/transformer_decoder.py, Line 173`
   - Attention: `models/SCORER.py, Line 48`
   - Note: these snippets are mostly similar. No residual links or LayerNorms to add, as we have already put them in the place if needed.

The places you need to modify in source files start with a comment including `== To Implement ==` . Also refer to the comment provided in the source files.

5. Training

To train the proposed method, run the following commands:

```
# create a directory or a symlink to save the experiments logs/snapshots etc.
mkdir experiments
# OR
ln -s $PATH_TO_DIR$ experiments

# this will start the visdom server for logging
# start the server on a tmux session since the server needs to be up during training
python -m visdom.server

# start training
python train.py --cfg configs/dynamic/transformer_quick.yaml
```

Note that we use a fractional of the whole data (~25%) for training in this assignment.

6. Hints

[1]We also provided mirrored download from Baidu Yun and PKU Netdisk. Downloading from these mirrors are recommended on local machine, and you might want to download from Google Drive when using Colab for faster speed.

1. If you find it difficult to implement them, read the paper and remaining code to better comprehend how each component work together. You can also refer to public Transformer and InfoNCE codes for reference.
2. You can also test your code implementations using less (or more) examples by tweaking `data.train.max_samples` or using less VRAM by reducing batch size at `data.train.batch_size` in the config YAML file. Note that reducing the batch size might lead to a compromised performance, due to the reduced negative sample pool size in InfoNCE.
3. If you want to try full training (takes ~2 4090 hours, BS=128, #Iter=10000), replace the config file to `configs/dynamic/transformer.yaml` .

7. Testing/Inference

To test/run inference on the test dataset, run the following command

```
python test.py --cfg configs/dynamic/transformer.yaml  --snapshot 10000 --gpu 1
```

The command above will take the model snapshot at 10000th iteration and run inference using GPU ID 1.

8. Evaluation

- Caption evaluation

Run the following command to run evaluation:

```
# This will run evaluation on the results generated from the validation set and
print the best results
python evaluate.py --results_dir ./experiments/SCORER+CBR/eval_sents --anno ./data/
total_change_captions_reformat.json --type_file ./data/type_mapping.json
```

Once the best model is found on the validation set, you can run inference on test set for that specific model using the command exlpained in the `Testing/Inference` section and then finally evaluate on test set:

```
python evaluate.py --results_dir ./experiments/SCORER+CBR/test_output/captions --
anno ./data/total_change_captions_reformat.json --type_file ./data/type_mapping.json
```

The results are saved in `./experiments/SCORER+CBR/test_output/captions/eval_results.txt`

9. Hand-In Requirements

You are going to hand in following meterials for scoring:

1. **Python source files**.
   Only submit the four modified files: `models/CBR.py`, `models/transformer_decoder.py`, `utils/utils.py`, `models/SCORER.py`
2. **Brief report** of your own implementation, including:
   1. the evaluation results on both validation and test set
   2. a simple visualization comparing the change caption between (original image, changed image, no-change image) triplet. An example is shown in Figure 2.

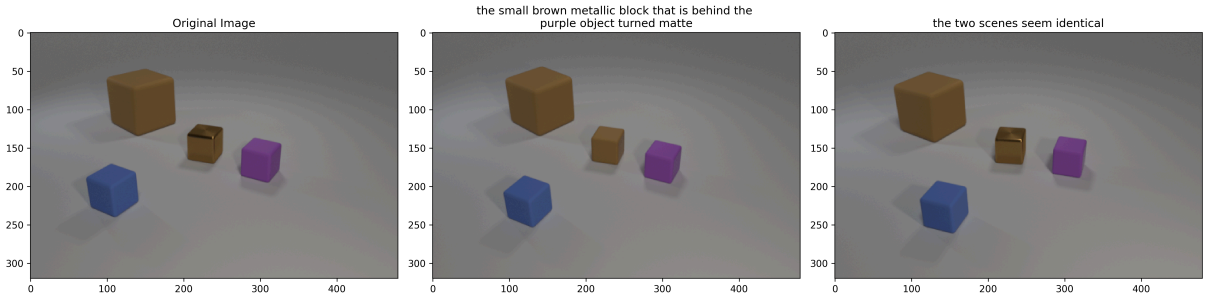

Figure 2: Exemplar Visualization

10. Reference

[1] A. Vaswani *et al.*, "Attention is All you Need", in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, p. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf