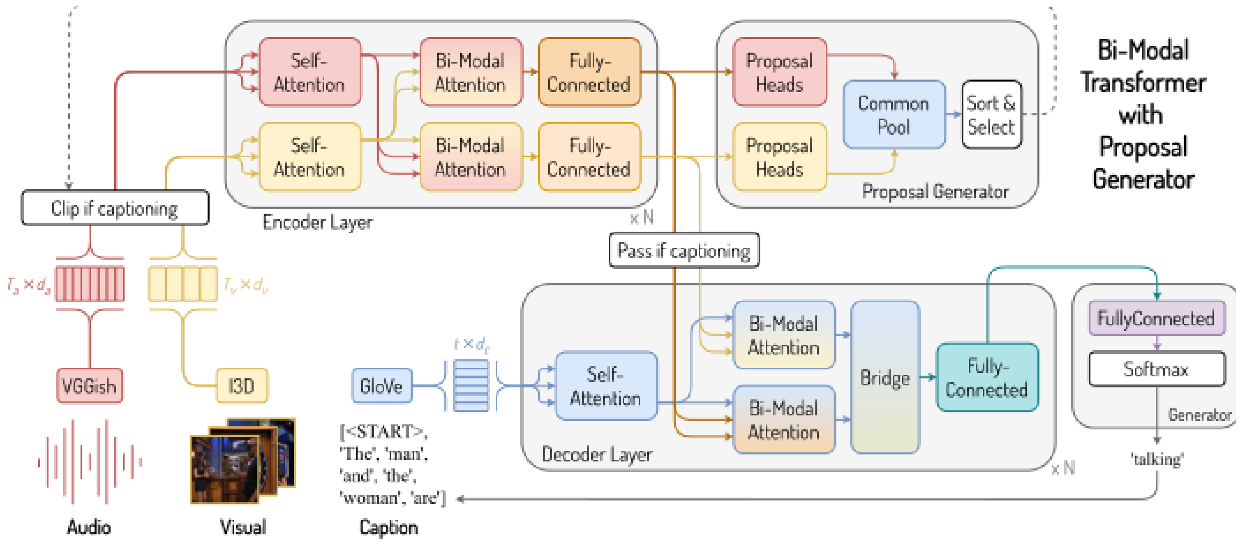
Dense Captioning Events in Videos

Methodology

Audio and visual features are encoded with VGGish and *i3d* while caption tokens with *GloVe*. First, VGGish and I3D features are passed through the stack of *N* bi-modal encoder layers where audio and visual sequences are encoded to, what we call, audio-attended visual and video-attended audio features.



These features are passed to the bi-modal multi-headed proposal generator, which generates a set of proposals using information from both modalities.

Then, the input features are trimmed according to the proposed segments and encoded in the bi-modal encoder again. The stack of *N* bi-modal decoder layers inputs both: a) GloVe embeddings of the previously generated caption sequence, b) the internal representation from the last layer of the encoder for both modalities. The decoder produces its internal representation which is, then, used in the generator model the distribution over the vocabulary for the caption next word.

Functionalities

⟶ download_data.sh :

Downloads the features (I3D and VGGish) and word embeddings (GloVe). The script will download them (~10 GB) and unpack into `./data` and `./.vector_cache` folders. *Make sure to run it while being in BMT folder*
bash ./download_data.sh

⟶main.py (implements train_cap_model.py,train_prop_model.py) :
We train our model in two stages: training of the captioning module on ground truth proposals and training of the proposal generator using the pre-trained encoder from the captioning module.

⟶ Feature Extraction folder for extracting i3d and vgg-ish features
“python main.py --feature_type i3d(or vggish) --device_ids 0 --on_extraction save_numpy
--file_with_video_paths ./sample/sample_video_paths.txt “ in the feature extraction folder creates a output folder which contains flow and rgb numpy files for i3d features and a vggish feature npy file for audio.

⟶ Test/test.py :
Takes i3d and vggish features of the video we want to get captions of , proposal model, captioning model, duration. Output is expected to be :

```
 [
  {'start': 0.0, 'end': 4.9, 'sentence': 'We see the closing title screen'},
  {'start': 2.7, 'end': 29.0, 'sentence': 'A woman is seen running down a track and down a track while others
watch on the sides'},
  {'start': 19.6, 'end': 33.3, 'sentence': 'The man runs down the track and jumps into a sand pit'},
  {'start': 0.0, 'end': 13.0, 'sentence': 'A man is seen running down a track and leads into a large group of
people running around a track'},
  {'start': 0.9, 'end': 2.5, 'sentence': 'We see a title screen'},
  {'start': 0.0, 'end': 1.6, 'sentence': 'A man is seen sitting on a table with a white words on the screen'},
  {'start': 30.0, 'end': 35.2, 'sentence': 'The man runs down the track and lands on the sand'}
]
```

⟶Evaluate :
The performance of the captioning module on ground truth segments might be obtained from the file with the pre-trained captioning module.

```
“python
import torch
cap_model_cpt = torch.load('./path_to_pre_trained_model/best_cap_model.pt', map_location='cpu')
print(cap_model_cpt['val_1_metrics'])
print(cap_model_cpt['val_2_metrics'])”
# To obtain the final results, average values in both dicts
```