Integrating a Memory Bank into BiomedParse

Introduction

BiomedParse is a foundation model for biomedical image parsing that unifies segmentation, detection, and recognition tasks. MedSAM-2 is a model that treats medical images like videos, using a memory mechanism to maintain coherence between successive frames. This document explores the possibility of integrating a similar memory mechanism into BiomedParse.

1. BiomedParse Architecture Breakdown

- Image Encoder to extract features from a single image or slice
- Text Encoder to parse natural language requests
- Mask Decoder to generate the segmentation
- Meta-object Classifier to associate the image with a broad ontology of possible objects and tissues

2. Memory Mechanism in MedSAM-2

MedSAM-2 employs "memory attention," selectively retaining information from multiple slices in a 2D/3D image series. This allows tracking objects coherently across a volume, thereby reducing the need for prompts at every slice.

3. Memory Bank Integration approach:

3.1 Stateful Encoder

Extending the image encoder to handle a "stream" of correlated images, using a key-value recall module that stores partial embeddings from previous slices.

In-Depth Look at the Stateful Encoder

A stateful encoder is an architectural component that goes beyond the traditional “stateless” approach commonly used in many image processing models, including those for segmentation. Rather than treating each image (or slice) independently, a stateful encoder keeps an internal memory of previous computations, allowing it to reference earlier information when processing new data. This ability is particularly beneficial for sequential or volumetric data (like 3D image series), where maintaining spatial consistency between slices is key.

Fundamental Concepts

Imagine you're analyzing a series of slices from a medical scan. A conventional encoder would process each slice from scratch, ignoring valuable context from earlier slices. In contrast, a stateful encoder works by:

- **Maintaining Sequential Context:** It saves "embeddings" or feature representations from earlier slices. These stored representations can help the model interpret the current slice in light of what it has seen before.

- **Using Key-Value Structures:** For each slice, the encoder generates an embedding that highlights its most important features. These embeddings are then stored in a memory bank as key-value pairs. The "key" identifies critical attributes—such as an anatomical feature—and the "value" contains the relevant context.

- **Aggregating Information via Cross-Attention:** When a new slice is processed, a cross-attention module queries the memory bank to retrieve relevant information from previous slices. Combining the local features of the current slice with these historical embeddings typically leads to more consistent and accurate segmentation.

Practical Benefits

The advantages of using a stateful encoder include:

- **Improved Consistency Across Slices:** With memory in place, the model "remembers" where specific structures were located in earlier slices. This results in more uniform segmentation across the entire volume.

- **Reduced Need for Repeated Prompts:** Since the model can recall essential details from its memory, it does not require a full prompt or guidance for every new image. This can significantly cut down on user intervention in long image sequences.

- **Enhanced Efficiency for Sequential Data:** For modalities such as MRI or CT scans, where there are many slices per volume, keeping this contextual information helps prevent errors or inconsistencies that might otherwise occur if each slice were processed in isolation.

How to Implement a Stateful Encoder

There are several practical ideas on how to retrofit an image encoder to make it "stateful":

1. **Extending the Image Encoder:**
   Instead of restarting the processing for every slice, modify the encoder to include a memory component—similar to what you'd find in transformer models. This might involve adding one or more layers specifically responsible for generating and storing embeddings.

2. **Employing Key-Value Mechanisms:**
   As each slice is processed, generate an embedding and store it in a memory bank as a key-value pair. This memory bank should be designed to update dynamically as more slices are processed.
3. **Adding a Cross-Attention Module:**
   When processing a new slice, incorporate a cross-attention module that "looks back" to the stored embeddings. The module should selectively pull in only the most relevant historical features to combine with the current slice's data.
4. **Designing Memory Update Strategies:**
   It is important to establish rules for updating the memory. For instance, you might apply a decay factor to lessen the influence of older embeddings or use a prominence factor to highlight key features. Properly balancing these factors helps keep the memory both useful and manageable.

A Practical Example

Consider segmenting a tumor that spans several slices of a CT scan. A standard encoder might segment each slice independently, resulting in slight variations from one slice to another due to noise or differences in contrast. In contrast, with a stateful encoder:

▪ The first slice is processed, and its embedding—highlighting the tumor—is stored in memory.
▪ When the next slice is processed, the cross-attention module references the stored tumor-related features and combines them with the new slice's local features.
▪ The final output is a segmentation that remains consistent across all slices, even if some individual slices are less clear.

3.2 Cross-Attention within the Mask Decoder

The mask decoder could receive both local visual features and "memory embeddings" from preceding slices, updating the memory at each new step.

3.3 Query Frame and Key Slice

Process an initial reference slice, extracting a key embedding that can be projected onto subsequent slices by cross-attention.

3.4 Update Mechanisms

We'd need to adopt a self-sorting memory bank with a decaying factor for older states and a prominence factor for highly visible objects as in MedSAM-2.

3.5 Alignment of Formats

It is crucial to verify that the embedding dimensions from the image encoder, memory bank, and the decoder are compatible, if not by reducing the feature map size to scale.