Draft Methods

Created	@November 24, 2024 3:16 PM

Methods

Our proposed framework aligns pre-trained unimodal encoders (e.g., **ResNet-18** for images and **GPT-2** for text) into a shared multimodal latent space using a set of trainable linear adapters. This section describes the data pipeline, model architecture, training procedure, and evaluation metrics in detail.

1. Data Pipeline

The data pipeline processes paired image and text data from the Flickr30k dataset. This dataset provides a natural starting point for aligning visual and textual modalities due to its paired image-caption structure.

1. **Image Processing**:
   - Images are resized to 224×224 pixels.
   - They are normalized to match the preprocessing used in ResNet-18 training.
   - The processed images are converted into tensors and batched for input to the image encoder.

   Code Reference: The `ImageTransforms` module in the code handles resizing, normalization, and tensor conversion for image data.

2. **Text Processing**:
   - Captions are tokenized using a BERT tokenizer, and special tokens (e.g., [CLS], [SEP]) are added.
   - Tokenized text is padded to ensure uniform input length across the batch.
   - Text data is converted into tensors for input to the text encoder.

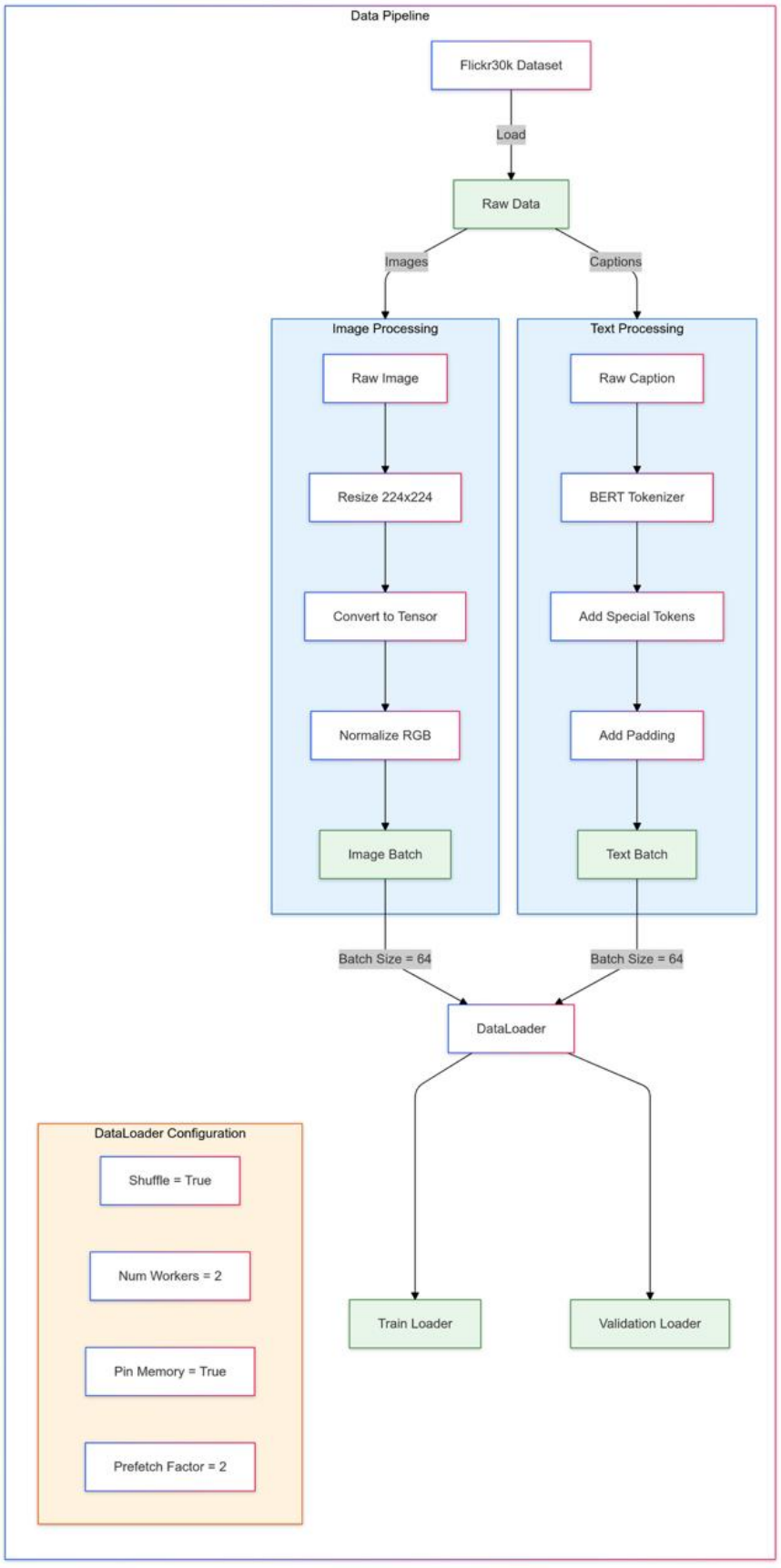
   Code Reference: The `TextTokenizer` module processes captions, converting them into model-compatible embeddings.

3. **Data Loader**:
    - Data loaders shuffle the dataset and prepare batches for training and validation.
    - Configurations include `batch_size=64` and optimizations like pinned memory and prefetching for faster data loading.

    Code Reference: The `DataLoaderConfig` handles the configuration for both train and validation loaders.

Figure Placeholder: Data Pipeline Schematic

Filename: `figures/data_pipeline.png`


Data Pipeline
Flickr30k Dataset
Load
Raw Data
Images
Captions
Image Processing
Raw Image
Resize 224x224
Convert to Tensor
Normalize RGB
Image Batch
Text Processing
Raw Caption
BERT Tokenizer
Add Special Tokens
Add Padding
Text Batch
Batch Size = 64
Batch Size = 64
DataLoader
DataLoader Configuration
Shuffle = True
Num Workers = 2
Pin Memory = True
Prefetch Factor = 2
Train Loader
Validation Loader

Caption: Illustration of image and text processing pipelines leading to batched inputs for ResNet-18 and GPT-2 encoders.

---

2. Model Architecture

The architecture integrates frozen pre-trained encoders for images and text with learnable linear adapters that map unimodal features to a shared latent space.

1. **Pre-trained Encoders**:
   - **ResNet-18** extracts 512-dimensional feature vectors from images.
   - **DistilBERT** (a smaller variant of GPT-2) generates 768-dimensional embeddings from text captions.

   Code Reference: The `ImageEncoder` and `TextEncoder` modules implement these encoders with frozen weights.

2. **Linear Adapters**:
   - Learnable linear adapters project unimodal features into a shared $d_e = 768$-dimensional latent space.
   - Variants include:
     - Linear adapters ($d_x \to d_e$).
     - MLP adapters ($d_x \to 1024 \to d_e$).
     - Bottleneck adapters ($d_x \to 256 \to d_e$).

   Code Reference: The `AdapterArchitecture` module defines these adapter variants.

3. **Contrastive Loss**:
   - A contrastive loss function aligns representations from the same image-caption pair in the latent space while separating unrelated pairs. The similarity score is scaled by a temperature parameter τ.

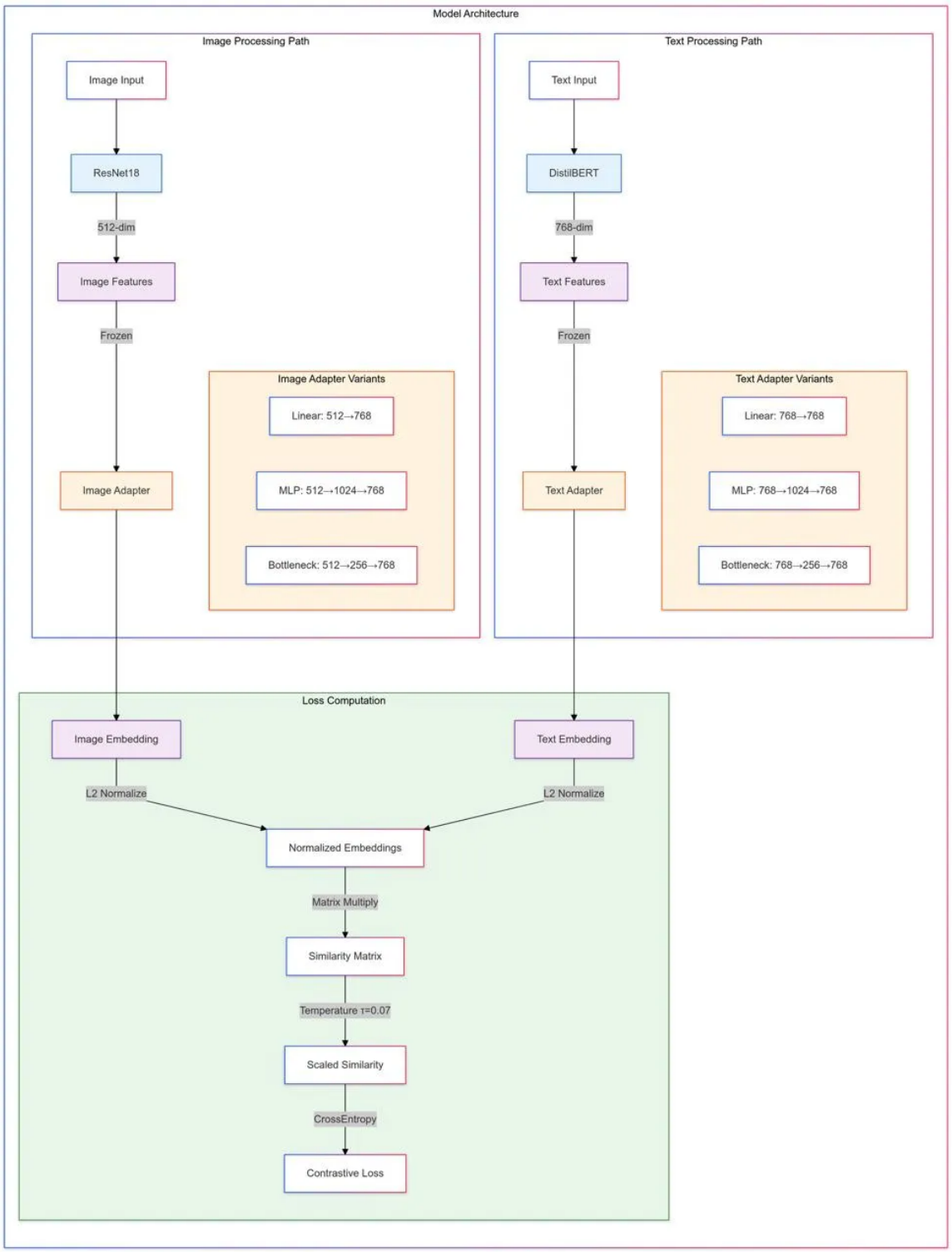
   Loss Formula:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=1}^{N} \log \frac{\exp\left(\left\langle W_x X_{\text{enc}}\left(x^{(i)}\right), W_y Y_{\text{enc}}\left(y^{(i)}\right)\right\rangle / \tau\right)}{\sum_{j=1}^{N} \exp\left(\left\langle W_x X_{\text{enc}}\left(x^{(i)}\right), W_y Y_{\text{enc}}\left(y^{(j)}\right)\right\rangle / \tau\right)}$$

Code Reference: The `LossComponents` module computes the contrastive loss for training.

Figure Placeholder: Model Architecture Diagram

Filename: `figures/model_architecture.png`



Caption: Schematic of the image and text processing paths, adapters, and the shared multimodal latent space.

3. Training Procedure

1. **Training Loop**:
    - Batches of images and captions are processed by their respective encoders and adapters.
    - The model computes similarity matrices for paired and unpaired examples.
    - Gradients are computed for the contrastive loss, and parameters of the adapters are updated.

    Code Reference: The `TrainingLoop` module orchestrates data processing, loss computation, and parameter updates.

2. **Evaluation**:
    - Alignment metrics, such as kernel similarity, compare the learned multimodal representation with DinoV2 embeddings.
    - Metrics are tracked across training epochs to monitor convergence.

    Code Reference: The `AlignmentMetrics` module computes kernel alignment metrics.

Figure Placeholder: Training Loss and Alignment Metrics

Filename: `figures/training_loss_metrics.png`

Caption: Plots of training loss and kernel alignment scores across epochs.

---

4. Evaluation Metrics

We evaluate the quality of the learned multimodal representation using kernel alignment metrics:

1. **Unimodal Kernel Similarity**:
    - Measure similarity between DinoV2 embeddings and individual image/text kernels before alignment.
2. **Aligned Kernel Similarity**:
    - Compare the aligned multimodal kernel with DinoV2 embeddings after training.

Code Reference: The `KernelMetrics` module computes similarity scores and tracks progress.

Figure Placeholder: Kernel Similarity Heatmaps

Filename: `figures/kernel_similarity_heatmaps.png`

Caption: Heatmaps showing representational similarity between multimodal and DinoV2 embeddings.

---

5. Implementation Details

- **Frameworks**:
  - PyTorch for model definition and training.
  - Hugging Face Transformers for text encoding.
  - Weights & Biases for experiment tracking and visualization.
- **Hyperparameters**:
  - Batch size: 64
  - Learning rate: $1e{-}4$
  - Temperature parameter: $\tau = 0.07$
  - Number of epochs: 50
