Task 2 Deliverable — Model Architecture (MRIEmbeddingModel)

Overview

This document describes the Task■2 model that maps batches of multi■coil complex MRI inputs into fixed■size embeddings. The implementation emphasizes simplicity, readability, and production■friendliness and is designed to work directly with Task■3 distance■based objectives.

I/O Contract

Input: x ∈ ■^(B × C_in × H × W), where C_in = 2 × num_coils (real and imaginary channels stacked by Task■1). Output: y ∈ ■^(B × D), where D = embed_dim (L2■normalized by default).

Architecture Summary

- Stem: LazyConv2d → GroupNorm → GELU (adapts automatically to C_in).
- Backbone (4 stages): Conv2d(stride=2) → GroupNorm → GELU → (optional Dropout).
- Head: GlobalAveragePool → Linear(projection to embed_dim) → (optional 2■layer MLP).
- Output: optional L2 normalization (enabled by default).

Key Design Choices & Rationale

- Compact CNN: satisfies the brief’s emphasis on a simple, not■overly■complicated model while being fast and stable.
- GroupNorm: robust to small batch sizes common in medical■imaging pipelines.
- LazyConv2d: avoids hard■coding the coil count; works with any C_in produced by Task■1.
- Global Average Pooling: parameter■efficient spatial aggregation.
- L2■normalized embeddings: distances (cosine/Euclidean) behave well for contrastive objectives in Task■3.
- Deterministic default: dropout is 0.0 by default to keep tests reproducible; enable during training if desired.

Initialization & Implementation Details

Convolutional layers use Kaiming initialization with ReLU gain (compatible with GELU). Linear layers use Xavier uniform. For the first LazyConv2d, parameters are initialized only after the first forward pass; the code guards against initializing unmaterialized lazy parameters.

Public API

File: model.py; Class: MRIEmbeddingModel(embed_dim=256, widths=(32,64,128,256), dropout=0.0, normalize=True, use_mlp_head=False). Methods: forward(x[, normalize]) → (B, D); embed(x, normalize=True) → (B, D) under no■grad.

Sanity Checks (PyTest)

A small test suite validates output shapes, L2■normalization behavior, gradient flow, and LazyConv2d adaptation to different channel counts. Command: PYTHONPATH=. ./.venv/bin/python -m pytest -q. All tests pass.

Evaluation Criteria Compliance

- Accepts a batch of coil images and outputs a fixed■size embedding vector.
- Clean, readable, and production■friendly code with clear I/O contract and documentation.
- Simple architecture (compact CNN) that is easy to extend (e.g., optional MLP head).
- Rationale for design choices is documented for the deliverable write■up.

Usage Example

>>> import torch >>> from model import MRIEmbeddingModel >>> x = torch.randn(2, 16, 128, 128) # 8 coils → 16 channels (real+imag) >>> model = MRIEmbeddingModel(embed_dim=256, normalize=True) >>> y = model(x) # y.shape == (2, 256)

End of Document

Additional Notes on Architecture Choice

The current compact CNN architecture was deliberately chosen to align with the Task 2 instructions, which emphasize that the model "need not be overly complicated" and should prioritize clarity and production-friendliness. This architecture satisfies those requirements while providing stable training, flexible input channel handling, and well-behaved embeddings for Task 3 objectives.

Alternative Architectures (for reference):
- **ResNet-18**: A widely used CNN backbone that could be adapted by modifying the input layer to accept 2×coils channels. It provides stronger representational power and is commonly used in metric learning tasks. However, it adds complexity and batch-normalization sensitivity to small batches.
- **ConvNeXt-Tiny**: A modern CNN design that improves performance compared to ResNet at similar computational cost. Slightly heavier, but production-ready.
- **Vision Transformers (e.g., DeiT-Tiny)**: Offer global receptive fields and flexibility, but typically require more data to perform well and are less efficient on small synthetic datasets.

Conclusion:
For the purposes of this interview task, the provided compact CNN is sufficient and appropriate. It is lightweight, easy to interpret, and satisfies the evaluation criteria. Alternatives like ResNet-18 or ConvNeXt could be explored in larger-scale setups, but are not strictly necessary here.