Foqus Technologies Inc. Machine Learning Task

Introduction

In MRI scans, acquiring full k-space data (raw frequency data) takes time, so modern techniques undersample k-space to speed up scans. However, this leads to incomplete data, and the goal is to reconstruct high-quality images from this undersampled data.

To do so, we may wish to use a contrastive learning approach to pre-train foundation models that can better represent raw MRI data. In this technical task, you will implement a simple contrastive learning approach to train a general feature extractor for MRI data.

A few things to keep in mind throughout this exercise:

- **We care more about your thinking and process than your results**, so don't feel that you need to spend a lot of time tuning things. If you wish, you are welcome to describe what you *would* do.
- **Code quality counts**; use your judgement and apply best practices. Pretend that your code is going to be part of a production ML pipeline. Some tips:
  - Write clean, modular, and well-documented code.
  - Use efficient algorithms to optimize performance.
  - Ensure readability with meaningful variable names and structured functions.
  - Include error handling where necessary.
  - Provide comments to explain your implementation.
- While this is meant to be an individual exercise, **you can ask clarifying questions if you need to**.
- **You may use AI assistants** to help with coding and debugging. However, ensure that you **understand and justify the solutions** you implement.
- **Try to be concise** in your writeup, as we're looking for quality over quantity.
- You are **free to modify any of the provided code**.

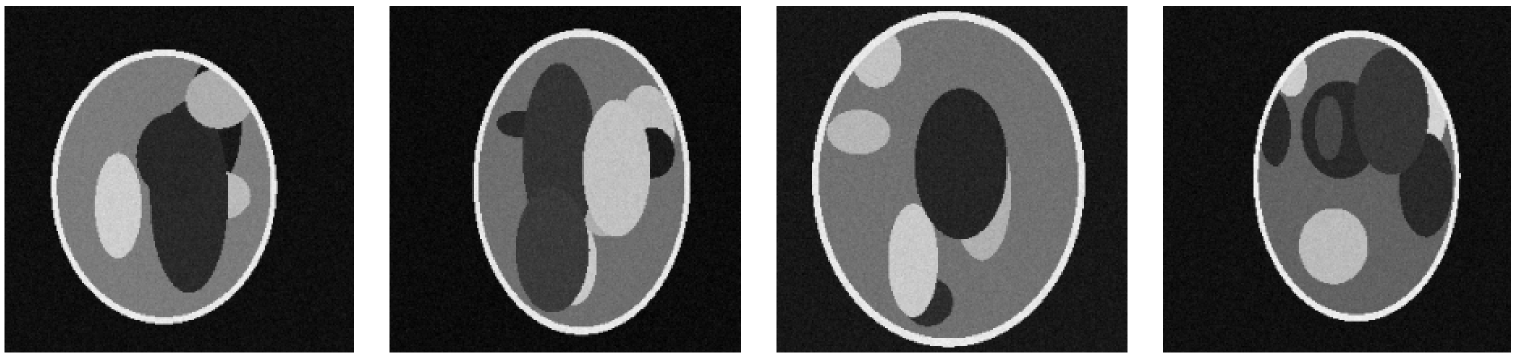
Getting Started

The technical task is provided as a zipped folder containing these instructions and some starter code. For the provided starter code, **the base requirements are Python>=3.10, numpy>=2.0.0, and torch>=2.8.0.** You may change these if you wish, but please use at least Python 3.10.

We use Git to manage our code, and we expect you will as well. Please implement your changes in a Git repository - to submit it, you can provide us with a link to your repository or a zipped copy of it.

Task 1 - Dataset Generation

First things first - we need some data to work with. We will be working with randomly-generated phantom MRI data consisting of several small ellipses contained within a larger “head” ellipse. Some examples of these images are shown below:



In accelerated MRI applications, images are acquired using several **receiver coils**, each of which acquires a separate channel (i.e., we are working with multi-channel data). To simulate this, we will multiply the magnitude images shown above by a series of **complex-valued coil sensitivity maps,** which encode the spatial sensitivities of the various receiver coils. The result is a complex-valued array with shape (num_coils, num_rows, num_columns) for each of the generated images. In PyTorch, we want this to be a real-valued tensor, and so we convert the real/imaginary parts to separate channels, yielding a tensor with shape (num_coils, num_rows, num_columns, 2).

The process described above has been implemented for you in the file phantom.py. Your task will be to further modify the images through data augmentation, undersampling, and conversion to a form that we can use for training.

Instructions

1. Implement the following transformations in the provided transforms.py:
   a. transforms.Normalize
   b. transforms.RandomUndersample
   c. transforms.Augmentation
   d. transforms.ToCompatibleTensor

   Further instructions are provided in the docstrings of the classes.
2. In a file of your choosing, create an instance of datasets.RandomPhantomDataset using your new transformations. For undersampling, use 4x acceleration with 8% center fraction. Visualize the resulting images.
3. (OPTIONAL) You may implement other transformations, but **only do so if you have a good reason to.**

Tips

- PyTorch includes some built-in methods for handling complex data:

    - torch.view_as_complex
    - torch.view_as_real
- We have provided Fourier transform functions for convenience:
    - transforms.fft
    - transforms.ifft
- Some transformation classes may not have __init__ methods, but you are free to add/modify the __init__ methods as you see fit.

Deliverables

1. transforms.py, with the aforementioned transformations implemented.
2. In your writeup, include:
    a. Explanations of any important choices you made when implementing the transformations.
    b. An explanation of the **order** in which you applied your transformations and why.
    c. Example images created by the instance of datasets.RandomPhantomDataset with your transformations applied.
        i. **NOTE:** once you have your visualizations, you **do not** have to keep the code you used to generate them (unless you want to).
    d. An explanation of any additional/optional steps you took.

Task 2 - Model Architecture

We now need a model to work with. This model must:

- Accept a batch of coil images as input (i.e., a batch of outputs from your dataset in Task 1).
- Produce a fixed-size embedding (i.e., vector) for each of the input images.

Instructions

1. In model.py, implement an embedding model of your choosing. It need not be overly complicated, and you are free to design it as you see fit.

Tips

- You are free to use online resources to help in your design process.
- Similarly, you may use an existing architecture, but please reference it and explain your choice.

Deliverables

1. model.py, with your model implemented therein.
2. In your writeup, provide an explanation of your model design or choice of existing model architecture.
    a. If applicable, you may wish to describe the embedding space of your model.

Task 3 - Training

We now have a dataset and model, so we can get started on training. We will be using a **contrastive learning** approach where we attempt to embed similar images close together and dissimilar images far apart.

To do so, you have been provided with the class datasets.RandomPhantomTripletDataset. This dataset is similar to datasets.RandomPhantomDataset, but it returns 3 images: the first two are the same base image with different transformations applied, and the third image is a different base image altogether. We will train our model to closely embed the first two images and separate them from the third image.

Instructions

1. Create a training dataset using datasets.RandomPhantomTripletDataset. This dataset accepts two lists of transforms:
    a. **Use 4x acceleration with 8% center fraction** for one of the transform lists and **8x acceleration with 4% center fraction** in the other.
    b. Include all other transformations in both lists.
2. Create a second RandomPhantomTripletDataset for validation. This dataset should include the same transforms, but **should not** include augmentations. Additionally, be sure to set *offset=len(training_dataset)* and *deterministic=True* so that the validation samples are always the same and do not overlap the training samples.
3. Implement a simple training loop in train.py.
    a. Load batches of samples from your training dataset.
        i. Each batch includes three tensors: same1, same2, and diff
    b. Run each of the tensors through your embedding model
    c. Compute a loss that minimizes the distance between same1 and same2 while maximizing the distance between same1/2 and diff.
        i. The meaning of “distance” is up to you to decide.
    d. Update weights and all that jazz.
4. Implement a validation process that runs at the end of each training epoch.
5. Train your model and plot the training and validation curves.

Tips

- You are free to change dataset parameters for simplicity/speed - for example, the number of samples or number of coils. Please do so within reason (e.g., an image size of 10x10 is unreasonable).
- If you find that your model does not train effectively, you can try to fix it, but if that fails you can also try to explain *why* your model might be failing.
- You need not train to convergence, but please train long enough for the training/validation curves to be meaningfully interpreted.

Deliverables

1. train.py, with an implementation of your training and validation processes. Include instructions for running the training process.
2. In your writeup, include:
   a. An explanation of your design choices when creating the training loop.
   b. An explanation of your loss function and the types of embeddings it uses.
   c. Training/validation curves, with a discussion of your results.
   d. Any suggestions for improvements to your training process.

Task 4 - Cleanup

Now that we have some working code, the last step is to improve our overall project. This is important for maintainability and usability; we don't want to incur tech debt through quick-and-dirty implementations that have to be rewritten in the future.

Instructions

This is an open-ended task - there are no specific instructions and there is no wrong answer. We're looking to see what is, in your opinion, a well-structured and maintainable project.

Tips

- Some things you may want to consider:
  - Refactoring some of your code to improve modularity/reusability. You can refactor ours too if you'd like!
  - Module structure and file locations - feel free to move things around to your liking.
  - Git-related configuration.
  - Installation and package management.
  - Documentation.
  - Testing, linting, code formatting, or type checking.
  - Test automation.
- Some improvements are harder than others; if you find yourself spending too much time, you can include a *proposal* for changes you would like to see in your writeup. This should be more than a general statement (e.g., "add tests") - try to be specific about what you would implement and why.

Deliverables

1. In your writeup, include a description of the steps you took to improve the project and your reasons for taking said steps.
2. Provide your final Git repository with all tasks completed.
   a. Include your writeup as a PDF file in the repository.

b. You may either (a) share a link to a Git repository, or (b) provide us with a zipped copy of your repository.