Chapter 1

Maintenance Manual

1.1 System Overview

The maintenance manual provides the details of the ETCAPS implementation as well as instructions for future work. The system implements several neural network architectures including Equivariant Transformer (ET), ETCaps, Self Routing Capsules (SRCaps), and ResNet20.

1.2 Installation Instructions

1.2.1 System Requirements

- **Operating System**: Windows, Linux, or macOS
- **Hardware**: CUDA-compatible GPU, High Performance Computer Cluster for training
- **RAM**: 16GB minimum
- **Disk Space**: 10GB for code, datasets, and model checkpoints

1.2.2 Software Dependencies

- Python 3.8
- PyTorch 1.8 with CUDA support
- Python packages: see req.txt. Omitted for lack of space.

1.2.3 Installation Steps

For installation instructions, see the user manual.

1.3 System Architecture

1.3.1 Directory Structure

```
Equivariant-Transformer-Capsule-Networks/
 main.py                  # Main entry point for training models
 eval_classification.py  # Script for evaluating classification accuracy
 class_acc.py             # Classification accuracy utility functions
 train.sh                 # SLURM job submission script
 src/                     # Core source code
    coordinates.py       # Coordinate manipulation utilities
    datasets.py          # Dataset loading and preprocessing
    grid_sampler.py      # Grid sampling for transformations
    loss.py              # Loss functions
    models.py            # Model architectures
    networks.py          # Network components
    norb.py              # smallNORB dataset handling
    resnet.py            # ResNet implementation
    train.py             # Training functions
    transformers.py      # Transformer implementations
 liederiv/                # Lie derivative evaluation code
    exps_e2e.py          # End-to-end equivariance evaluation
    lee/                 # Lie derivative implementation
```

1.3.2 Model Checkpoints Structure

Model checkpoints are saved in directories named after the dataset, model, and experimental condition:

```
cifar10/                    # Dataset name
 et/                       # Model name
    cifar10_best_32_1.pth  # Best checkpoint (32 caps, depth 1)
 resnet20/                 # ResNet20 model
    cifar10_best_32_1.pth
 srcaps/                   # SRCaps model
     cifar10_best_32_1.pth
```

1.3.3 Temporary Files

During training and evaluation, the following temporary files may be created:

- `*.pth.tmp` - Temporary checkpoints during model saving
- `*.log` - Log files for training progress

1.3.4 Dataset Preparation

The first time you run the training script, datasets will be automatically downloaded. Alternatively, you can pre-download them to the `data/` directory.

1.4 Source Code Documentation

1.4.1 Key Source Files

Table 1.1: Source Code Files and Their Roles

File	Role
main.py	Entry point for training models with command-line argument parsing
eval_classification.py	Script for evaluating classification accuracy on test sets
src/models.py	Model architecture definitions for ET, ETCaps, SRCaps, and ResNet20
src/transformers.py	Implementations of equivariant transformers
src/networks.py	Equivariant Transformer components
src/train.py	Training loop functions
src/datasets.py	Dataset loading and preprocessing utilities
liederiv/exps_e2e.py	End-to-end equivariance evaluation using Lie derivatives

1.4.2 Crucial Constants

Table 1.2: Important Constants in the Codebase

Constant	Location
Dataset configurations	`src/datasets.py:DATASET_CONFIGS`
Viewpoint experiment types	`src/datasets.py:VIEWPOINT_EXPS`
Coordinate system limits	`src/transformers.py` (in each transformer class)
Learning rate schedules	`main.py` (in main_worker function)

1.5 Memory and Space Requirements

1.5.1 Disk Space Requirements

- CIFAR-10 Dataset: 340MB
- SVHN Dataset: 61MB

- smallNORB Dataset: 1.67GB
- Model Checkpoints: 3-6 MB per model
- Total: 1 GB for basic setup, 10 GB with multiple trained models and results

1.5.2 Memory Requirements

- Training ET model: 2-3 GB GPU memory
- Training ETCaps model: 5-6 GB GPU memory
- Training SRCaps model: 4-5 GB GPU memory
- Training ResNet20 model: 1-2 GB GPU memory
- RAM usage: 8 GB during training with batch size 64

1.6 Main Classes and Methods

1.6.1 Model Architectures

- **ETCaps** (in `src/models.py`): Equivariant Transformer Capsule Network model
- **ET** (in `src/models.py`): Equivariant Transformer model with ResNet backbone
- **SRCaps** (in `src/models.py`): Self Routing Capsule model with ResNet backbone.
- **ResNet20** (in `src/resnet.py`): 20-layer ResNet model with average pooling and final classification layer

1.6.2 Other Components

- **Transformer** (in `src/transformers.py`): Base class of a Transformer. Other specific equivariant transformers inherit from this class, using their specific canonical coordinates.
- **TransformerSequence** (in `src/transformers.py`): Combines a collection of transformers in sequence
- **TransformerLayer** (in `src/networks.py`): Wrapper class for `TransformerSequence` that implements ET layer in ETCAPS architecture.

1.6.3 Training and Evaluation Functions

- **train_epoch** (in `src/train.py`): Function for training a single epoch
- **validate** (in `src/train.py`): Function for validation during training
- **test** (in `train.py`): Function for calculating classification accuracy
- **get_metrics** (in `liederiv/lee/lie_derivs.py`): Computes and returns equivariance errors for all transformations.

1.7 Future Improvements

- Implement additional equivariant transformations for more transformation groups, including 3D transformation groups.
- Add support for self-supervised learning approaches
- Implement learned coordinate system transforms so that no prior knowledge of domain is required.
- Add support for more datasets including 3DIEBench.
- Add a hyperparameter tuning pipeline.

1.8 Known Issues and Bugs

- Training can sometimes be unstable.
  - Try to tune the hyperparameters for faster and more stable convergence.
- **CUDA Out of Memory**: This error typically occurs when the GPU runs out of memory during training. To mitigate this:
  - **Reduce Batch Size**: Decreasing the number of samples processed simultaneously can significantly lower memory usage.
  - **Simplify Model Architecture**: Reducing the number of capsules or the depth of the network can help reduce memory usage.
- **Training Divergence**: If the training process becomes unstable or the loss increases uncontrollably:
  - **Lower Learning Rate**: A high learning rate can cause the model to overshoot minima, leading to divergence.
  - **Increase Weight Decay**: Adding regularisation can prevent overfitting and stabilize training by penalizing large weights.

- **Dependency Errors**: You might encounter errors such as:

  ```
  ModuleNotFoundError: No module named 'typing_extensions'
  ImportError: numpy.core.multiarray failed to import
  ```

  To resolve these:

  - **Install Missing Packages**: Use `pip install typing-extensions` or `pip install numpy` to install the required modules.
  - **Ensure Compatibility**: Make sure that the versions of installed packages are compatible with each other and with your Python environment.
  - **Reinstall in Virtual Environment**: If issues persist, consider setting up a new virtual environment and reinstalling dependencies to avoid conflicts.
  - **Mismatched Model Architecture during model loading**: You might encounter errors such as:

    ```
    RuntimeError: Error(s) in loading state_dict for <ModelClass>:
        Missing key(s) in state_dict: "layer1.weight", "layer2.bias".
        Unexpected key(s) in state_dict: "module.layer1.weight", "module.layer2.bia
        ...
    ```

    In this case, ensure that you set the model hyperparameters (in the command line arguments) to exactly what you did during training.

1.9 Extending the Framework

1.9.1 Adding New Models

To add a new model:

1. Create a new class in `src/models.py` that inherits from `nn.Module`
2. Implement the `__init__` and `forward` methods
3. Add the model to the model factory in `src/models.py`
4. Update the command-line argument parsing in `main.py`

1.9.2 Adding New Datasets

To add a new dataset:

1. Create dataset loading functions in `src/datasets.py`
2. Add the dataset to the dataset factory in `src/datasets.py`
3. Update the command-line argument parsing in `main.py`

1.9.3 Adding New Equivariance Metrics

To add new equivariance metrics:

1. Implement the metric in `liederiv/lee/lie_derivs.py`
2. Add the metric to the evaluation function in `liederiv/exps_e2e.py`
3. Update the CSV output format to include the new metric