Code Documentation

Project Title

Advancing Precision Oncology: A Deep Learning Framework for Predicting Drug Sensitivity

---

1. Overview of Codebase

This repository implements a modular and interpretable deep learning pipeline for **drug response prediction**, focused on estimating IC50 values using a **hybrid GCN-CNN model** with **cross-attention fusion**. It integrates molecular graph representations of drugs (via SMILES strings) and gene expression profiles of cancer cell lines.

Key Features:

- **SMILES → Graph conversion** using RDKit
- **Gene expression preprocessing** with Z-score normalisation and outlier handling
- **Dataset packaging** with PyTorch Geometric
- **GCN + CNN architecture** with attention fusion
- **Training + Hyperparameter tuning + Evaluation**
- **Web deployment** using Gradio interface

---

2. Code Structure & Functional Highlights

Component	Description
`convert_smile_to_graph()`	Parses SMILES into RDKit molecular graphs
`encode_atom_features()`	Extracts atomic descriptors: atom type, valency, hybridisation
`preprocess_gene_expression()`	Cleans and Z-score normalises gene expression
`prepare_drug_cellline_dataset()`	Combines molecular graphs and gene vectors
`DrugResponseModel`	Hybrid architecture: GCN + 1D CNN + cross-attention
`train_and_evaluate_model()`	Trains, validates, and evaluates model with logging
`evaluate_per_drug()`	Computes metrics per compound
`save_model_and_results()`	Persists model weights and metrics

Component	Description
`fetch_hyperparam_results()`	Loads and visualises tuning results
`predicting_and_evaluate()`	Runs full evaluation on holdout test set

---

3. Detailed Pipeline Walkthrough

Step 1: Drug Graph Construction

- `convert_smile_to_graph()` converts SMILES (e.g., Gefitinib) into RDKit-based molecular graphs.
- `encode_atom_features()` + `one_hot_encode()` transform atoms into numerical features for GCN input.

Step 2: Gene Expression Preprocessing

- `preprocess_gene_expression()` applies log transform, Z-score normalisation.
- `detect_outliers_iqr()` filters noisy or outlier genes to improve generalisation.

Step 3: Dataset Assembly

- `prepare_drug_cellline_dataset()` merges drug and gene data into graph + vector formats.
- Uses `DrugGeneDataset` to format PyTorch Geometric `Data` objects.
- Batched with `get_data_loaders()`.

Step 4: Deep Learning Model

- `DrugResponseModel`: GCN (for drug) + CNN1D (for cell line) + Cross-attention fusion.
- Managed by `train_and_evaluate_model()` and `train()` for parameter updates.

Step 5: Hyperparameter Tuning

- Explores optimisers (Adam, RMSprop, RAdam, AdamW), loss functions (MSE, HuberLoss(delta=0.5), SmoothL1Loss), learning rates (1e-4, 5e-4).
- Evaluated using:
  - `compute_mse()`, `compute_rmse()`
  - `compute_pearson_correlation()`, `compute_spearman_correlation()`
  - `compute_r2_score()`
- Logged using CSV summaries and `fetch_hyperparam_results()`.

Step 6: Model Saving & Logging

- `save_model_and_results()` persists weights, logs, and metadata.

Step 7: Final Evaluation

- `predicting_and_evaluate()` evaluates on hold-out data.
- Metrics + plots:
  - `plot_training_and_test_loss()`
  - `plot_residuals()`

Step 8: Per-Drug Analysis

- `evaluate_per_drug()` groups metrics by drug for bias/imbalance analysis.
- Output: Ranked DataFrame of drug-wise performance.

Step 9: Gradio Deployment

- Lightweight web app allows users to:
  - Select drug + cell line
  - Internally convert inputs
  - Get IC50 predictions with qualitative labels (Strong/Moderate/Weak response)
- Backend calls trained model + preprocessing utilities

---

4. Evaluation Metrics

Metric	Purpose
MSE / RMSE	Regression accuracy
Pearson / Spearman Correlation	Captures pattern alignment
R^2 Score	Measures variance explanation (used cautiously due to low IC50 variance)

Diagnostic Plots

- **Training vs Test Loss**
- **Residual Distributions**

---

5. Model Development & Testing

- Extensive hyperparameter tuning was conducted on validation splits.

- Final model tested on **unseen drug-cell line pairs** using `predicting_and_evaluate()`.
- All runs are reproducible with stored seeds, saved checkpoints, and logs.

---

6. Gradio Deployment

- `app.py`: Runs local Gradio server
- Dropdowns for drug and cell line selection
- Outputs: predicted IC50 + qualitative label

To Run Locally:

```
# Clone the repo
git clone https://github.com/tanayab/DrugResponse.git
cd DrugResponse/WebApp/

#(Optional) Create a Python Virtual Environment:
python3 -m  venv venv
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Launch the Gradio app
python app.py
```

Summary

This end-to-end pipeline: 1. Converts SMILES into graphs with atom features 2. Preprocesses gene expression with robust normalisation 3. Fuses both modalities using cross-attention in a deep network 4. Trains and evaluates using statistical metrics and visual diagnostics 5. Analyses per-drug performance for interpretability 6. Exposes predictions through a Gradio web interface

The codebase is: - **Modular & Reproducible** - **Highly Commented** - **Optimised for Biomedical Deep Learning**