M.A.I.D: Modular AI-based Imaging & Diagnostics

M.A.I.D is a **graphical user interface (GUI)** driven toolkit for **cell-line image classification** and **analysis**, designed for biomedical researchers who want to preprocess images, train convolutional neural networks, run inference, and interpret classification results *without heavy coding overhead*.

***This software is provided for research and educational purposes only**. It is **not** approved for clinical or diagnostic use. All **data privacy** and **regulatory compliance** is the user's responsibility.

Table of Contents

Features

- **Plugin-based GUI (PyQt5):** Each major step of the deep-learning pipeline (e.g., data preprocessing, model training) is modularized into “tabs” or plugins.
- **Advanced Deep-Learning Workflows (PyTorch Lightning):** Seamlessly train a variety of CNN architectures (ResNet, EfficientNet, etc.) with advanced features like MixUp, CutMix, LR finder, partial freezing layers, and more.
- **Data Exploration:** Quickly visualize class distributions, sample images, and view or log folder structure.
- **Interpretability:** Support for Grad-CAM/Grad-CAM++, SHAP analysis, confusion matrices, classification reports, and more.
- **High Customizability:** Toggle data augmentations (random crop, flipping, brightness/contrast, elastic transforms), partial freeze layers, or multi-GPU training with ease.
- **Transparent Source Code:** The code is structured so that advanced users can modify or extend functionality with minimal friction.

Installation and Dependencies

1. Create a Virtual Environment (Recommended)

```
bash
Copy
python -m venv venv
source venv/bin/activate  # on Linux/Mac
# or venv\Scripts\activate on Windows
```

2. Install Requirements

Make sure you have `Python 3.8+` installed. Then:

```
bash
Copy
pip install -r requirements.txt
```

If you use GPU acceleration, ensure PyTorch is installed with the correct CUDA version.
Some optional dependencies:

- **`pytorch_optimizer`** if you want to use the LAMB optimizer.
- **`optuna`** if you want to run hyperparameter tuning.

3. Launch the App

```
bash
Copy
python main.py
```

This will open the M.A.I.D GUI window with all available plugins as separate tabs.

Quick Start

1. **Launch `main.py`:** A window titled **"M.A.I.D"** will open, with multiple tabs (e.g., *Preprocessing*, *Data Exploration*, *Training*, *Evaluation/Inference*, etc.).
2. **Preprocessing Tab:**
    - Select a single image or bulk folder to apply transformations (thresholding, morphology, noise removal, overlays, etc.).
3. **Data Exploration Tab:**
    - Point to a dataset folder with subfolders for each class (e.g., `data/classA/`, `data/classB/`).

    - Click **Analyze** to view class distribution chart and sample images.
4. **Training Tab:**
    - Select your dataset folder and configure training parameters (augmentations, architecture, optimizer, scheduler, etc.).
    - Press **Start Training** to begin. You can monitor the progress in the log.
5. **Evaluation/Inference Tab:**
    - Browse for a trained checkpoint (`.ckpt`) from your training results.
    - Load an image or a folder of images to run inference.
    - Optionally enable Grad-CAM / Grad-CAM++ overlays or SHAP interpretability.

Project Structure

Inside this repository, you will find several Python scripts that make up the application:

```
css
Copy
├── main.py
├── base_plugin.py
├── plugins/
│   ├── plugin_preprocessing.py
│   ├── plugin_data_exploration.py
│   ├── plugin_training.py
│   ├── plugin_inference.py
│   └── ...
├── check_dependencies.py
├── requirements.txt
└── README.md  <-- (You are here!)
```

Key Files:

- **`main.py`**
  Launches the PyQt5 application, loads all plugins, and creates the main window with multiple tabs.
- **`base_plugin.py`**
  The abstract base class that all plugins derive from. Each plugin must specify a `plugin_name` and implement `create_tab()`.
- **`plugin_preprocessing.py`**
  Contains the Preprocessing plugin: single/bulk image transformations, morphological ops, thresholding, channel overlays, etc.
- **`plugin_data_exploration.py`**
  Implements the Data Exploration plugin: scanning data folders, summarizing class distributions, plotting histograms, etc.
- **`plugin_training.py`**
  Implements the Training plugin: letting you choose architecture, hyperparameters, data augmentations, partial freeze, etc. Runs a training loop with PyTorch Lightning.

- **`plugin_inference.py`**
  Provides an Evaluation/Inference plugin: load a `.ckpt` checkpoint, run predictions on single images or entire folders, visualize Grad-CAM or Grad-CAM++ heatmaps, compute metrics, etc.

Using the Toolkit

1. Preprocessing Plugin

Purpose: Quickly preprocess biomedical images (e.g., thresholding, morphological operations, channel extraction) either **one-by-one** or **in bulk**.

Key Steps:

1. **Single Image Section:**
   - **Browse Image...**: Load a single image for preview.
   - **Apply Preprocessing**: Check or configure operations like histogram equalization, threshold, invert, morphological ops, denoising, or red channel enhancement.
   - **Save Processed Image**: Exports the processed result.
2. **Bulk Preprocessing Section:**
   - **Input Folder** and **Output Folder**: Recursively processes all images in the input, applying the same transformations used in single-image mode.
   - **Output Format & Suffix**: Convert all images to e.g. `.jpg` with a custom suffix.
   - **Process Bulk Images**: Launches a background thread so the UI remains responsive.
3. **Overlay**:
   - If you have separate channel files (e.g., `-ch01`, `-ch02`), this can merge them into a color-coded overlay (red-green).
   - Additional toggles for histogram equalization, CLAHE, or normalization before merging.
4. **Filtering**:
   - Copy only images containing a certain substring (e.g., `ch01`) to a new folder.

Tips:

- Use **single image mode** to experiment with transformations. Then replicate the same pipeline to **bulk** mode.
- Check the **log** for any file reading issues.

---

2. Data Exploration Plugin

Purpose: Explore your dataset quickly to see if your classes are balanced or if certain classes have too few images.

Usage:

1. **Browse Data Folder**: Choose the parent dataset folder where each subfolder is a class (e.g., `data/ClassA`, `data/ClassB`).
2. **Max Images/Class**: In the gallery, how many samples to show per class.
3. **Analyze Dataset**:
    - A bar chart shows class distribution (e.g., ClassA: 120 images, ClassB: 85 images, etc.).
    - A scrollable gallery displays up to N sample images per class.
4. **Logging**: The text log field notes how many images were found, any folders with zero images, etc.

This step helps ensure your data is structured correctly before training.

---

3. Training Plugin

Purpose: Train convolutional neural networks on your image dataset with advanced PyTorch Lightning features.

Usage Workflow:

1. **Dataset Folder**: The same parent directory where subfolders represent classes.
2. **Validation and Test Splits**: Adjust the percentage splits. (E.g., 15% for validation, 15% for test, leaving 70% for training.)
3. **Architecture**: Choose a backbone (e.g., ResNet18, MobileNet, EfficientNet, etc.).
4. **Augmentations**: Check boxes for brightness/contrast, Hue/Sat, noise injection, random flips, random crop, etc.
5. **Hyperparameters**:
    - **LR**: Initial learning rate.
    - **Momentum**: For SGD (ignored for Adam).
    - **Optimizer**: e.g., Adam, SGD, AdamW, LAMB (if installed).
    - **Scheduler**: e.g., StepLR, ReduceLROnPlateau, CosineAnnealing, or none.
    - **Epochs**: Number of training epochs.
    - **Batch**: Mini-batch size for training.
    - **(Optional) Weighted Loss**: If classes are imbalanced, can apply weighting.
6. **Early Stopping / Checkpoints**: Automated stopping when validation stops improving.

7. **Loss Function**: Cross-entropy, Focal, BCE, BCE single-logit, etc.
8. **Partial Freezing**: Freeze entire backbone or freeze selected ResNet/ConvNeXt layers only.
9. **Start Training**:
    - A background thread will run so the UI remains responsive.
    - View logs for training progress, confusion matrix, classification report, etc.
    - The best model checkpoint is saved in your working directory (e.g., `lightning_logs/` or `best-checkpoint.ckpt`).

Optuna Hyperparameter Tuning (optional):

- Click **Tune with Optuna** and specify the number of trials to automatically search for the best LR, dropout, or other parameters.
- Not recommended to optimize on the *test set*—the plugin allows an option to use the test metric, but the default is the validation metric.

---

4. Evaluation/Inference Plugin

Purpose: Evaluate a trained model on new images or entire folders, get classification results, Grad-CAM overlays, confusion matrices, etc.

Workflow:

1. **Checkpoint**: Browse for your `.ckpt` file from the training stage.
2. **Single Image Inference**:
    - **Test Image**: Select an image.
    - **Top-k**: E.g., 3 to see top-3 predictions with probabilities.
    - **Min Confidence**: Filter out predictions below a threshold.
    - **Grad-CAM++**: Enable to get a heatmap overlay on the predicted class's activation regions.
3. **Evaluation**:
    - In the *Batch Inference* section, specify an **input folder** and optionally a ground-truth CSV for computing accuracy, confusion matrix, etc.
    - **Generate Grad-CAM** for every image in bulk.
    - **Export CSV** of predictions, plus a "misclassified.csv" to see which images are mis-labeled.
    - The plugin can create a confusion matrix, classification report, ROC curve, PR curve, calibration curve, SHAP analyses (if shap is installed), etc.

Extending with Plugins

M.A.I.D is built around a *plugin architecture*. Each plugin inherits from `BasePlugin`:

```
python
Copy
class BasePlugin:
    def __init__(self):
        self.plugin_name = "Base Plugin"

    def create_tab(self) -> QWidget:
        raise NotImplementedError("Plugins must implement
create_tab()")
```

- **`plugins/`** folder contains individual Python modules, each implementing a `Plugin(BasePlugin)` class with a `create_tab()` function that returns a PyQt widget.
- You can add new plugins by:
    1. Creating a new `.py` file in `plugins/`.
    2. Defining `class Plugin(BasePlugin): ...`
    3. Overriding the logic for your custom step (e.g., specialized image transformations, advanced metrics, etc.).
- The main application (`main.py`) automatically detects and loads all `Plugin` classes from `plugins/` at startup.

License

MIT License

If you find this toolkit useful for your work, please consider citing or sharing feedback. For questions, issues, or contributions, open an Issue or Pull Request on GitHub. Remember that **M.A.I.D** is *not* FDA-approved or CE-marked for clinical use.