BCC Detection and Analysis Pipeline

Project Overview

This project implements a robust, modular pipeline for the analysis of Basal Cell Carcinoma (BCC) in whole slide images (WSIs). The pipeline is designed for reproducible, scalable, and interpretable feature extraction and classification, using both deep learning and classical machine learning techniques. It is containerized for easy deployment and reproducibility.

Pipeline Summary

1. **Tissue Segmentation**: Isolate tissue regions from background in WSIs.
2. **Patch Extraction**: Divide tissue regions into smaller image patches.
3. **Feature Extraction**: Extract deep features (EfficientNetB7) and color features from each patch.
4. **Clustering (FCM)**: Group patches using Fuzzy C-Means clustering.
5. **Dimensionality Reduction (PCA)**: Reduce feature dimensionality for downstream analysis.
6. **Classification**: Use extracted features and labels for BCC/non-BCC and risk-level classification (outside this pipeline).
7. **Containerization**: Run the pipeline in a portable Ubuntu-based Docker container.

Directory Structure

```
project-root/
├── src/
│   ├── preprocessing/
│   │   ├── tissue_segmentation.py
│   │   └── patch_extraction.py
│   ├── feature_extraction/
│   │   ├── efficientnet.py
│   │   ├── color_features.py
│   │   ├── fcm_clustering.py
│   │   └── dimensionality.py
│   └── config.py
├── scripts/
│   ├── optimized_pipeline.py
│   ├── split_data.py
│   └── extract_features.py
├── requirements.txt
├── Dockerfile
├── .dockerignore
└── README.md
```

Step-by-Step Pipeline Description

1. **Data Splitting** (`scripts/split_data.py`)

- **Purpose:** Split the dataset into training, validation, and test sets with a controlled distribution of high-risk BCC, low-risk BCC, and non-BCC images.
- **How:**
  - Reads `labels.csv` to determine cancer status and risk level for each image.
  - Copies selected images into `train/`, `val/`, and `test/` folders.
- **Usage:**

```
python3 scripts/split_data.py --input_dir <input_images> --labels_file
<labels.csv> --output_dir <output_split_dir>
```

2. **Tissue Segmentation** (`src/preprocessing/tissue_segmentation.py`)

- **Purpose:** Identify and segment tissue regions in each WSI, removing background.
- **How:**
  - Applies image processing techniques to generate a binary tissue mask.
- **Usage:**
  - Called automatically by the pipeline for each image.

3. **Patch Extraction** (`src/preprocessing/patch_extraction.py`)

- **Purpose:** Divide the segmented tissue into smaller, manageable patches (e.g., 224x224 pixels).
- **How:**
  - Extracts patches only from tissue regions, using configurable patch size and overlap.
  - Saves each patch as a `.npy` file with coordinates in the filename.
- **Patch Existence Check:**
  - Before extracting, the pipeline checks if patch files for an image already exist and skips patching if so.

4. **Feature Extraction**

a. Deep Features (`src/feature_extraction/efficientnet.py`)

- **Purpose:** Extract high-level features from each patch using EfficientNetB7 (pretrained on ImageNet).
- **How:**
  - Processes batches of patches and outputs feature vectors.

b. Color Features (`src/feature_extraction/color_features.py`)

- **Purpose:** Extract color statistics (mean, std, histograms, etc.) from each patch in various color spaces.
- **How:**
  - Returns a dictionary of color features for each patch.

c. Feature Concatenation

- **Purpose:** Combine deep and color features into a single feature vector per patch.
- **How:**
  - Ensures both are 2D arrays and concatenates them along the feature axis.

5. **Clustering (FCM)** (`src/feature_extraction/fcm_clustering.py`)

- **Purpose:** Group patches into clusters using Fuzzy C-Means, allowing soft assignment to multiple clusters.
- **How:**
  - Outputs cluster membership scores for each patch.

6. **Dimensionality Reduction (PCA)** (`src/feature_extraction/dimensionality.py`)

- **Purpose:** Reduce the dimensionality of the feature vectors while retaining most of the variance.
- **How:**
  - Applies PCA to the cluster features.

7. **Saving Features**

- **Purpose:** Store the final reduced feature vectors for each image for downstream analysis.
- **How:**
  - Saves as `.npy` files in the output directory.

8. **Classification (Downstream, Not in Pipeline)**

- **Purpose:** Use the extracted features and `labels.csv` to train/test classifiers for BCC/non-BCC and risk-level prediction.
- **How:**
  - Match feature files to labels for supervised learning.

Main Scripts and Their Roles

- **`scripts/split_data.py`**: Splits images into train/val/test using `labels.csv`.
- **`scripts/optimized_pipeline.py`**: Runs the full pipeline (segmentation, patching, feature extraction, clustering, PCA, saving features).
- **`scripts/extract_features.py`**: (Optional) Standalone feature extraction from pre-extracted patches.
- **`src/preprocessing/tissue_segmentation.py`**: Implements tissue segmentation logic.
- **`src/preprocessing/patch_extraction.py`**: Implements patch extraction logic.
- **`src/feature_extraction/efficientnet.py`**: Deep feature extraction using EfficientNetB7.
- **`src/feature_extraction/color_features.py`**: Color feature extraction.
- **`src/feature_extraction/fcm_clustering.py`**: Fuzzy C-Means clustering.
- **`src/feature_extraction/dimensionality.py`**: PCA-based dimensionality reduction.
- **`src/config.py`**: Centralized configuration for pipeline parameters.

Running the Pipeline (Locally or in Docker)

A. Local Setup

1. Install dependencies:

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

2. Run the pipeline:

```
python3 scripts/optimized_pipeline.py --input_dir <input_images> --
output_dir <output_dir>
```

B. Dockerized Setup

1. Build the Docker image:

```
docker build -t bcc-pipeline .
```

2. Run the container, mounting your data:

```
docker run -it --rm -v /path/to/your/data:/data bcc-pipeline
```

3. Inside the container, run the pipeline:

```
python3 scripts/optimized_pipeline.py --input_dir /data/images --
output_dir /data/output
```

Notes and Best Practices

- **Data is not included in the Docker image**; mount it at runtime.
- **Patch existence check** prevents redundant computation.
- **Labels are used for splitting and downstream classification, not for feature extraction.**
- **Logs** are written for all major steps and errors.
- **Modular design** allows you to swap or extend components easily.

Contact & Support

For questions, issues, or contributions, please open an issue or contact the maintainer.