Basal Cell Carcinoma Detection System Development Prompt

Project Overview

Develop a comprehensive deep learning system for automated detection of Basal Cell Carcinoma (BCC) from high-resolution TIFF images. The system should be based on the architecture described in the research paper but adapted for TIFF images (250-400MB in size) instead of Whole Slide Images (WSIs).

Core Objectives

1. Process large TIFF images (250-600MB) efficiently
2. Implement the multi-stage detection pipeline based on the research paper
3. Create a modular, maintainable codebase with clear separation of concerns
4. Ensure proper data storage and organization for each processing stage
5. Provide visualization and interpretability tools for clinical use
6. Implement robust evaluation metrics to assess model performance

System Architecture

The system should follow the four-stage architecture outlined in the paper:

1. **Preprocessing** - Convert TIFF images to manageable patches
2. **Feature Extraction** - Using EfficientNet-B7 and Fuzzy C-Means clustering
3. **Classification** - Patch-level BCC detection
4. **Aggregation** - Produce final image-level diagnosis

Project Structure

```
bcc-detection/
│
├── data/
│   ├── raw/                  # Original TIFF images
│   ├── processed/            # Preprocessed data
│   │   ├── segmented/        # Tissue-segmented images
│   │   └── patches/          # Extracted patches (224×224)
│   ├── features/             # Extracted features
│   │   ├── deep/             # Features from EfficientNet-B7
│   │   ├── color/            # Color-based features
│   │   └── combined/         # Combined feature vectors
│   ├── predictions/
│   │   ├── patch_level/      # Patch classification results
│   │   ├── spatial/          # After spatial coherence enhancement
│   │   └── slide_level/      # Final image-level predictions
│   └── visualizations/       # Generated heatmaps and visual outputs
│
├── models/
```

```
│   ├── pretrained/           # Pretrained EfficientNet-B7 weights
│   ├── checkpoints/          # Model checkpoints during training
│   └── final/                # Final trained models
│
├── src/
│   ├── preprocessing/
│   │   ├── __init__.py
│   │   ├── tissue_segmentation.py  # Tissue segmentation module
│   │   ├── patch_extraction.py     # Patch extraction module
│   │   └── utils.py                # Preprocessing utilities
│   │
│   ├── feature_extraction/
│   │   ├── __init__.py
│   │   ├── efficientnet.py         # EfficientNet-B7 implementation
│   │   ├── color_features.py       # Color feature extraction
│   │   ├── fcm_clustering.py       # Fuzzy C-Means clustering
│   │   └── dimensionality.py       # PCA implementation
│   │
│   ├── classification/
│   │   ├── __init__.py
│   │   ├── model.py                # Classification model architecture
│   │   ├── training.py             # Training procedures
│   │   └── inference.py            # Inference procedures
│   │
│   ├── aggregation/
│   │   ├── __init__.py
│   │   ├── patch_aggregation.py    # Patch-to-image aggregation
│   │   ├── spatial_coherence.py    # Spatial coherence enhancement
│   │   └── confidence.py           # Confidence scoring
│   │
│   ├── utils/
│   │   ├── __init__.py
│   │   ├── data_handling.py        # Data loading/saving utilities
│   │   ├── visualization.py        # Visualization tools
│   │   ├── metrics.py              # Evaluation metrics
│   │   └── optimization.py         # Performance optimization tools
│   │
│   └── config.py                   # Configuration parameters
│
├── notebooks/
│   ├── 1_data_exploration.ipynb    # Data exploration and analysis
│   ├── 2_model_development.ipynb   # Model development and testing
│   ├── 3_performance_analysis.ipynb# Performance analysis
│   └── 4_case_studies.ipynb        # Case studies and examples
│
├── scripts/
│   ├── download_pretrained.py      # Download pretrained models
│   ├── preprocess_dataset.py       # Preprocess all images
│   ├── train_model.py              # Train the model
│   ├── evaluate_model.py           # Evaluate model performance
│   └── predict.py                  # Run predictions on new images
│
├── tests/
```

```
│   ├── test_preprocessing.py
│   ├── test_feature_extraction.py
│   ├── test_classification.py
│   └── test_aggregation.py
│
├── docs/
│   ├── architecture.md            # System architecture documentation
│   ├── usage.md                   # Usage instructions
│   └── api/                       # API documentation
│
├── requirements.txt               # Project dependencies
├── setup.py                       # Package installation
├── README.md                      # Project overview
└── .gitignore                     # Git ignore file
```

Implementation Details

1. Preprocessing Module

TIFF Image Handling

Unlike WSIs, TIFF images don't have multi-resolution pyramid structure. Therefore:

1. Implement efficient TIFF loading using libraries like tifffile or PIL with memory mapping
2. Implement tile-based processing to handle the large image sizes without loading entire images into memory
3. Create a downsampling function to generate lower-resolution versions for initial analysis

Tissue Segmentation

Adapt the tissue segmentation approach from the paper:

1. Implement color deconvolution to separate hematoxylin and eosin components
2. Apply Otsu's thresholding to separate tissue from background
3. Implement morphological operations (opening/closing) to refine segmentation
4. Perform connected component analysis to remove small artifacts

This module should output binary masks identifying tissue regions.

Patch Extraction

Implement the patch extraction process:

1. Divide segmented tissue into 224×224 pixel patches
2. Implement filtering to discard patches with less than 70% tissue content
3. Store spatial coordinates of each patch for later aggregation
4. Implement 50% overlap between patches as described in the paper
5. Save patches with appropriate metadata (coordinates, source image, etc.)

2. Feature Extraction Module

EfficientNet-B7 Implementation

1. Implement EfficientNet-B7 using TensorFlow or PyTorch
2. Load pretrained ImageNet weights
3. Implement adaptive data augmentation (rotations, flips, color jittering)
4. Configure progressive resolution training as described
5. Extract deep features using global average pooling
6. Apply PCA dimensionality reduction to preserve 99% variance

Color Feature Extraction

1. Implement conversion to multiple color spaces (HSV, Lab, YCbCr)
2. Calculate statistical features for each color channel
3. Generate color histograms and co-occurrence matrices
4. Implement Fuzzy C-Means clustering with 3 clusters
5. Store membership values as additional features

Feature Integration

1. Concatenate reduced EfficientNet features with FCM membership values
2. Save combined feature vectors for classification

3. Classification Module

Feature Concatenation and Model Architecture

1. Implement the fully connected network with:
    - First dense layer: 512 neurons, ReLU activation
    - Second dense layer: 256 neurons, ReLU activation
    - Third dense layer: 128 neurons, ReLU activation
    - Dropout layers after each dense layer (rate=0.3)
    - Batch normalization after each dense layer
    - Softmax output layer for binary classification

Model Training

1. Implement weighted binary cross-entropy loss
2. Configure Adam optimizer with 1e-4 initial learning rate and cosine decay
3. Implement early stopping and model checkpointing
4. Use stratified batch sampling for class balance
5. Implement mixed precision training for performance optimization
6. Save model checkpoints and training history

Patch Classification

1. Implement batch prediction for efficient processing

2. Store patch-level predictions with confidence scores

4. Aggregation Module

Spatial Coherence Enhancement

1. Implement weighted majority voting within 3×3 neighborhoods
2. Use confidence scores as weights
3. Store spatially-enhanced predictions

Image-Level Aggregation

1. Implement confidence weighting of patch predictions
2. Develop cluster analysis to identify contiguous BCC regions
3. Implement threshold-based classification (threshold ~0.3)
4. Calculate final confidence scores from weighted averages

Uncertainty Visualization

1. Generate confidence heatmaps for the entire image
2. Highlight regions of uncertainty for expert review
3. Create visual overlays of predicted BCC regions on original images

5. Optimization Techniques

Mixed Precision Training

1. Implement FP16 forward pass
2. Store FP32 master weights
3. Configure loss scaling to prevent gradient underflow
4. Implement dynamic loss scaling

Parallel Processing

1. Implement multi-threading for patch processing
2. Configure batch prediction to leverage GPU parallelism
3. Optimize memory usage during inference

Progressive Resolution Analysis

1. Implement low-resolution initial pass
2. Configure selective high-resolution analysis of potential BCC regions
3. Implement adaptive thresholding based on probability distribution

Evaluation Metrics

Implement and report the following metrics:

1. Accuracy, precision, recall, and F1-score

2. ROC curve and AUC
3. Confusion matrix
4. Processing time per image
5. Memory usage

Visualization Tools

Develop visualization tools for:

1. Segmentation results
2. Patch extraction coverage
3. Feature activations from EfficientNet
4. FCM clustering results
5. Prediction confidence heatmaps
6. Final BCC region identification

Documentation Requirements

1. Clear API documentation for each module
2. Detailed installation and setup instructions
3. Usage examples for each stage of the pipeline
4. Performance benchmarks and optimization guidelines
5. Troubleshooting guide for common issues

Deliverables

1. Complete source code implementing the architecture
2. Pretrained model weights
3. Documentation covering all aspects of the system
4. Example notebooks demonstrating the workflow
5. Test suite verifying component functionality
6. Performance evaluation report

Technical Considerations for TIFF Images

1. TIFF images (250-600MB) will require efficient memory handling
2. Implement memory mapping or chunked reading for large files
3. Consider using dask arrays for out-of-memory processing
4. Optimize I/O operations by minimizing disk reads/writes
5. Implement parallel processing where possible to handle large files efficiently
6. Consider cloud storage integration for scalable deployment

This prompt provides the framework for implementing the BCC detection system based on the research paper while adapting it for large TIFF images instead of WSIs.