BCanD Model

Dataset link:

https://www.kaggle.com/datasets/kmader/mias-mammography

Phase 1: Environment Setup

1.1 Install Required Tools

- Install Python (3.9 or above recommended).
- Install libraries:
-

bash

Copy code

```
pip install tensorflow keras numpy pandas matplotlib opencv-python scikit-learn
```

- Recommended: GPU setup with CUDA and cuDNN for TensorFlow or PyTorch if using a large dataset.

1.2 Prepare the Dataset

- Download the **MIAS dataset** from MIAS repository.
- Organize images into three folders:

css

Copy code

```
dataset/
  normal/
  benign/
  malignant/
```

1.3 Data Preprocessing

- Convert .pgm files to .jpg (if needed for compatibility) or load them as is.
- Augment data using rotations and other transformations to expand the dataset.
    - Rotate each image at intervals (0°, 45°, ..., 315°) to create more samples.
    - Use Python's **ImageDataGenerator** or custom augmentation pipelines.

Phase 2: Mass Detection (YOLOv4)

2.1 Annotate Images

- Use annotation tools like **LabelImg** to draw bounding boxes around masses in the mammograms.
- Save annotations in YOLO format:
    - Text files where each line contains:
      <class_id> <x_center> <y_center> <width> <height>
      (normalized to image size).

2.2 Prepare YOLOv4

- Download YOLOv4 pre-trained weights and config files from YOLO GitHub Repo.
- Adjust the configuration file (yolov4.cfg) for your dataset:
    - Update the number of classes (classes=2 for benign/malignant).
    - Adjust anchors and max batches according to dataset size.

2.3 Train YOLOv4

- Use **Darknet** or TensorFlow for training.
- Convert annotations and train YOLOv4 on the prepared dataset.
- Evaluate using metrics like **Intersection over Union (IoU)** and **F1-score**.

Phase 3: Mass Segmentation (U-Net)

3.1 Preprocess Detected Regions

- Apply **CLAHE (Contrast-Limited Adaptive Histogram Equalization)** to improve image contrast:

python

Copy code

```
import cv2

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))

enhanced_image = clahe.apply(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY))
```

3.2 Build U-Net Architecture

- Use TensorFlow/Keras to define the U-Net:

python

Copy code

```
def unet_model(input_size=(128, 128, 1)):
  inputs = tf.keras.layers.Input(input_size)
  # Encoder
  c1 = layers.Conv2D(64, (3, 3), activation='relu', padding='same')(inputs)
  p1 = layers.MaxPooling2D((2, 2))(c1)
  # Bottleneck
  b = layers.Conv2D(128, (3, 3), activation='relu', padding='same')(p1)
  # Decoder
  u1 = layers.Conv2DTranspose(64, (2, 2), strides=(2, 2), padding='same')(b)
  outputs = layers.Conv2D(1, (1, 1), activation='sigmoid')(u1)
  return tf.keras.models.Model(inputs, outputs)


model = unet_model()
```

3.3 Train U-Net

- Train on segmented mass images (from YOLO).
- Compile the model:

python

Copy code

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(train_data, train_masks, validation_data=(val_data, val_masks), epochs=50,
batch_size=16)
```

Phase 4: Mass Classification (CNN)

4.1 Resize Segmented Images

- Resize U-Net output to 40x40 pixels using bicubic interpolation:

python

Copy code

```
resized_img = cv2.resize(segmented_img, (40, 40), interpolation=cv2.INTER_CUBIC)
```

4.2 Build CNN

- Use AlexNet-like architecture for binary classification:

python

Copy code

```
def cnn_model(input_shape=(40, 40, 1)):
  model = tf.keras.models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(2, activation='softmax')  # benign vs malignant
  ])
  return model


model = cnn_model()
```

4.3 Train the CNN

- Compile and train:

python

Copy code

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, validation_data=(val_images, val_labels), epochs=30, batch_size=32)
```

Phase 5: Integration and Evaluation

5.1 Combine the Stages

- Create a pipeline:
    1. **Detection**: Use YOLO to locate ROIs.
    2. **Segmentation**: Apply U-Net to segment masses within detected ROIs.

3. **Classification**: Use the CNN to classify segmented masses as benign or malignant.

5.2 Evaluate the Model

- Use cross-validation and these metrics:
    - **Detection**: IoU, F1-score.
    - **Segmentation**: Dice coefficient, Jaccard index.
    - **Classification**: Accuracy, ROC-AUC, MCC.

5.3 Compare with State-of-the-Art

- Compare your model's performance with the results from the paper and other methods to validate improvements.

Phase 6: Deployment

6.1 Backend Deployment

- Use Flask or FastAPI to create an API for uploading mammograms and getting classification results.

6.2 Frontend Interface

- Use **Streamlit** for an interactive web interface:
    - Users can upload an image and see the detection, segmentation, and classification results in real-time.

6.3 Deployment on the Cloud

- Deploy the application using platforms like:
    - **Heroku** for small-scale deployment.
    - **AWS/GCP/Azure** for production-grade deployment with GPUs.

Optional Enhancements

- Incorporate explainability tools like Grad-CAM to visualize which parts of the image influenced the classification decision.
- Optimize for inference speed to handle real-time clinical use.
