Intelligent Radiologist Assistant (Lungs Tumor segmentation)

Low Level Design (LLD)

Author: Tanjina Proma

Date: 12/09/2024

Table of Contents

1 Introduction

1.1 What is Low-Level-Design document?

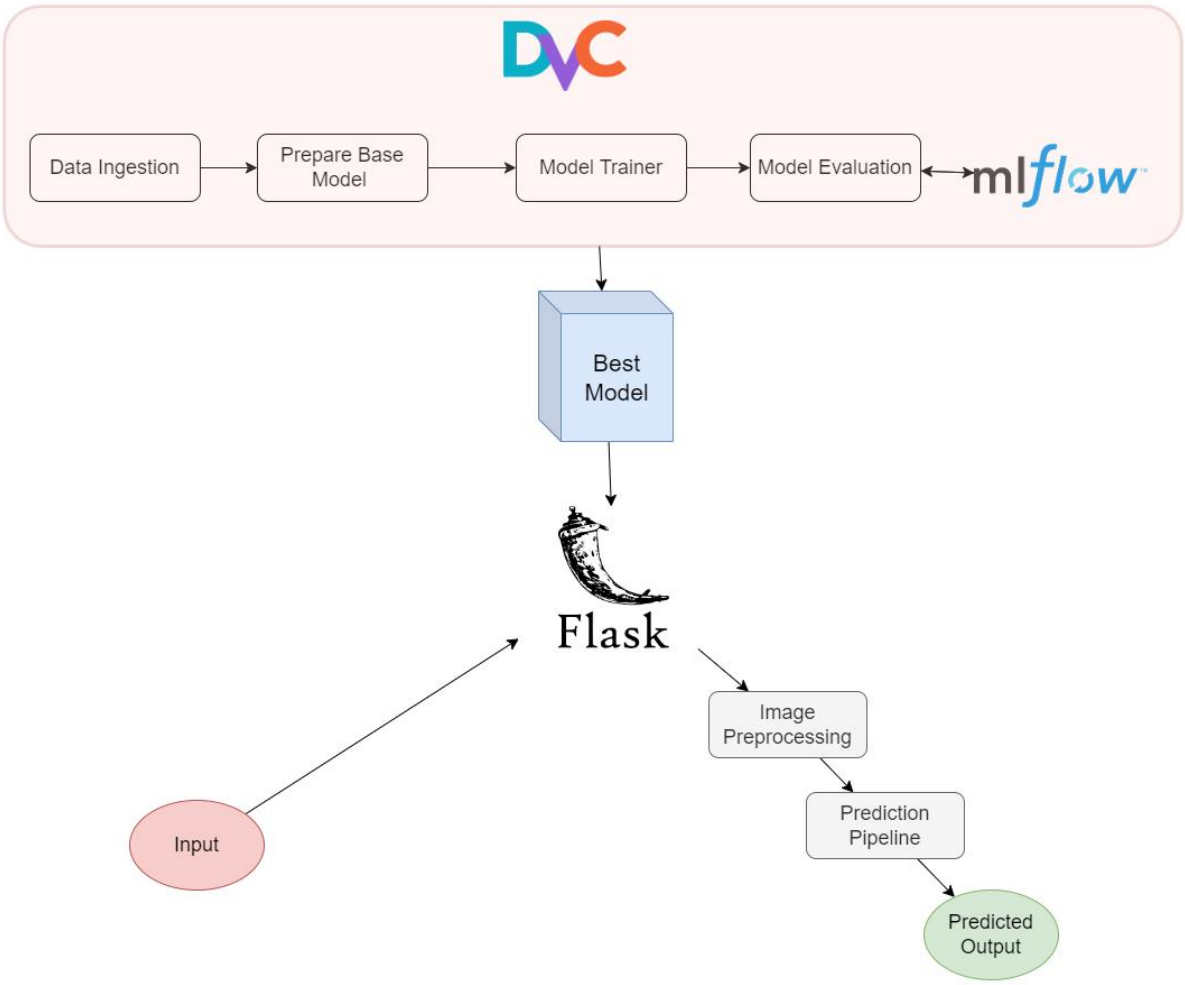
Low-Level Design (LLD) refers to the phase in software or system development where the actual implementation details of the system are planned out. This phase comes after the high-level design and involves creating detailed design specifications that describe how the system will be built.

The output of the LLD phase is often in the form of diagrams, charts, documents, and sometimes pseudocode that provides a blueprint for developers to start coding. It serves as a bridge between high-level architectural concepts and actual coding, making it easier for the development team to understand the implementation requirements.

1.2 Scope

The scope for Lung Tumor Classification from Chest CT Scans using VGG16 Model in TensorFlow, Python, Docker, MLflow, DagsHub, and DVC encompasses a comprehensive and scalable solution for automated diagnosis and treatment planning in lung cancer detection. Leveraging TensorFlow and Python, the project aims to develop a deep learning model based on the VGG16 architecture, capable of accurately classifying lung tumors as malignant or benign. Docker facilitates seamless deployment of the model, ensuring consistent execution across different environments. MLflow enables efficient experiment tracking, hyperparameter optimization, and model packaging, while DVC ensures versioning and reproducibility of datasets and model artifacts. Integration with DagsHub provides collaborative version control and continuous integration capabilities, streamlining project management and collaboration. The system promises to enhance diagnostic accuracy, streamline radiology workflows, and ultimately contribute to improved patient outcomes by enabling early detection and personalized treatment strategies for lung cancer patients.

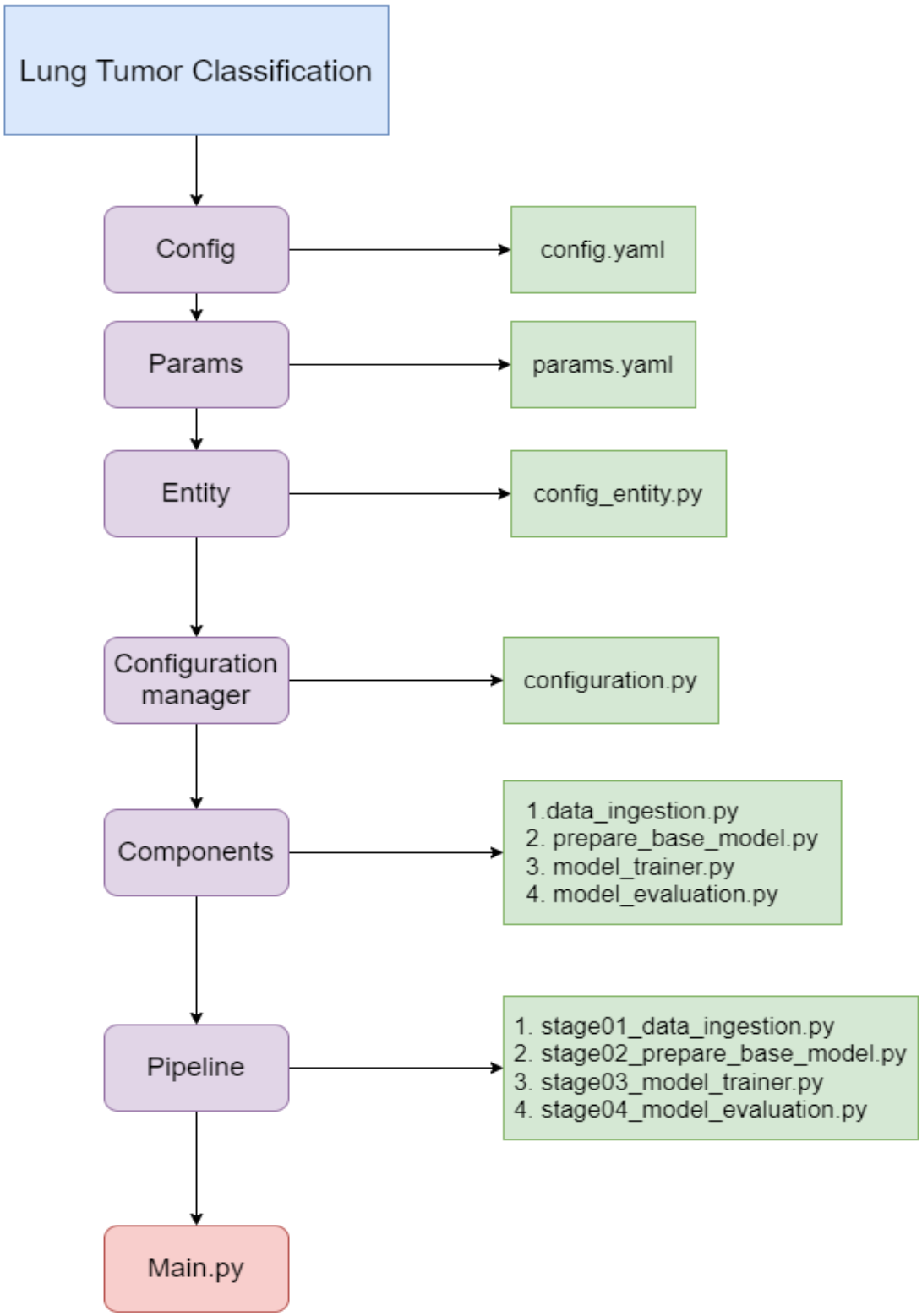
2 Architecture



3 Architecture Description

3.1 Data Acquisition and Preprocessing:

- **Data Collection Script:** Python script to download and preprocess chest CT scan datasets, utilizing libraries like PyDicom for DICOM image handling.
- **Preprocessing Pipeline:** Python functions for standardizing voxel sizes, resampling, intensity normalization, and augmentation using libraries like OpenCV and NumPy.
- **DVC Integration:** DVC to track dataset versions and reproduce preprocessing steps across different environments.


Lung Tumor Classification
Config
config.yaml
Params
params.yaml
Entity
config_entity.py
Configuration manager
configuration.py
Components
1.data_ingestion.py
2. prepare_base_model.py
3. model_trainer.py
4. model_evaluation.py
Pipeline
1. stage01_data_ingestion.py
2. stage02_prepare_base_model.py
3. stage03_model_trainer.py
4. stage04_model_evaluation.py
Main.py

3.2 Model Development:

- **Model Definition:** Python script defining the VGG16 model architecture using TensorFlow's Keras API.
- **Transfer Learning Script:** Python script to load pre-trained VGG16 weights, freeze convolutional layers, and modify fully connected layers for fine-tuning.
- **Training Script:** Python script to train the model using TensorFlow's **Model.fit()** function, integrating MLflow for experiment tracking.
- **Hyperparameter Tuning:** Utilize MLflow's hyperparameter optimization capabilities to search for optimal learning rates, batch sizes, etc.

3.3 Model Evaluation:

- **Evaluation Script:** Python script to evaluate model performance using metrics like accuracy, precision, recall, and AUC-ROC.
- **Visual Interpretation:** Python functions to overlay model predictions on CT scan images for qualitative assessment.

3.4 Model Deployment:

- **Dockerfile:** Definition of Dockerfile to create a reproducible Docker image encapsulating the model and its dependencies.
- **MLflow Deployment:** Utilize MLflow's model serving capabilities to deploy the trained model as a REST API endpoint.
- **Integration with DVC:** Incorporate DVC to manage model versions and ensure consistency between training and deployment environments.

3.5 Continuous Integration and Deployment (CI/CD):

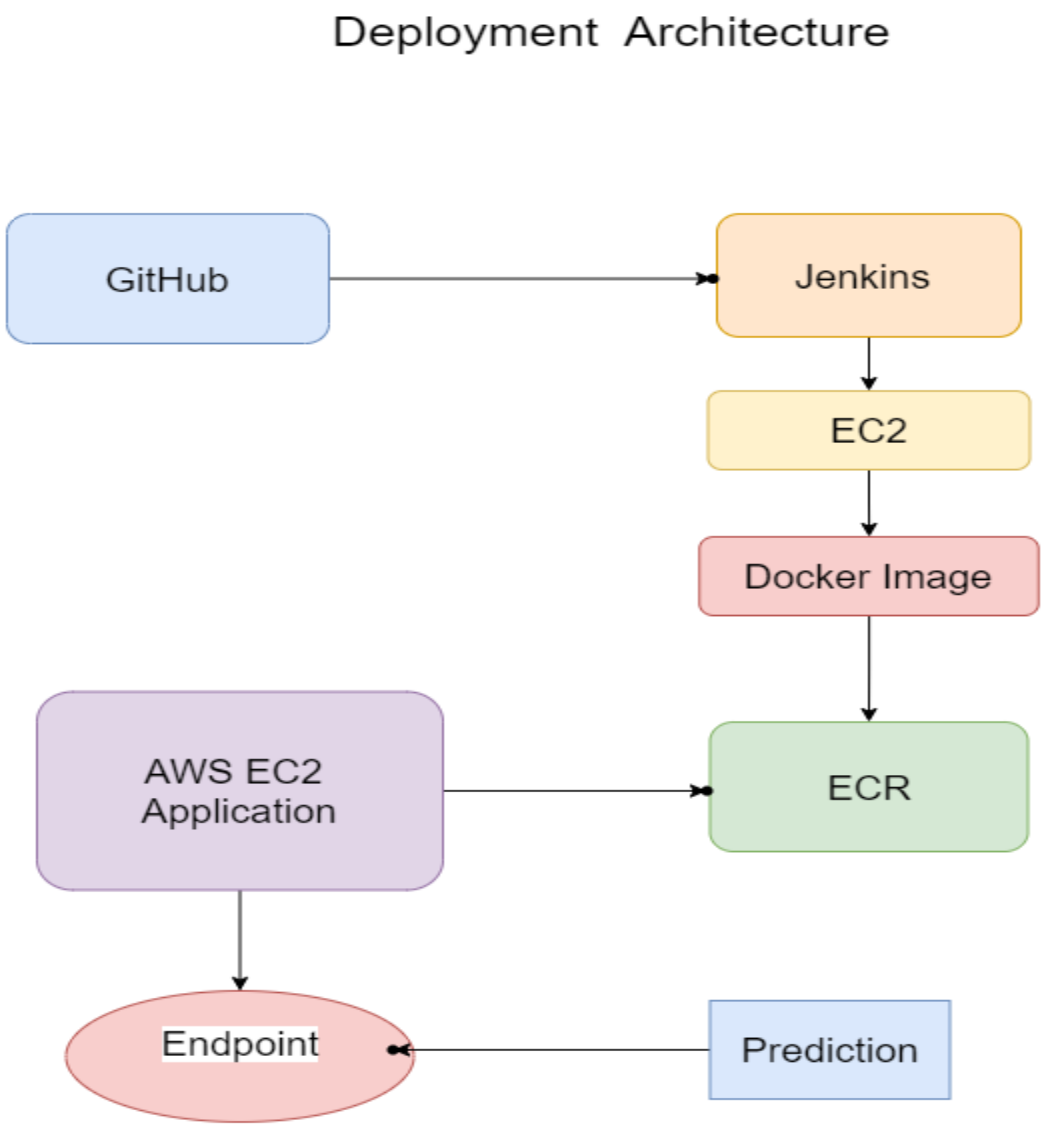
- **CI/CD Pipeline Definition:** Define CI/CD pipeline configurations using Jenkins.
- **Automated Testing:** Include unit tests and integration tests to ensure the reliability of code changes.
- **Automatic Deployment:** Trigger deployment to production or staging environments upon successful completion of tests.

3.6 Monitoring and Maintenance:

- **Model Monitoring Script:** Python script to monitor model performance, detecting changes in accuracy or drift in predictions.
- **Scheduled Retraining:** Schedule periodic retraining of the model using DVC to incorporate new data and maintain performance.

- **Error Logging and Reporting:** Implement error logging mechanisms to track and report any issues in the production environment.

Deployment Architecture



4. Unit Test Cases

4.1 Data Preprocessing Tests:

- Test that the preprocessing pipeline correctly resamples CT scans to a uniform resolution.
- Verify that intensity normalization functions properly normalize voxel values across different scans.
- Ensure that data augmentation techniques maintain the integrity of tumor annotations.

4.2 Model Development Tests:

- Test that the VGG16 model architecture is defined correctly with appropriate layers.
- Verify that transfer learning script correctly loads pre-trained VGG16 weights and modifies fully connected layers.
- Ensure that the training script initializes the model, optimizer, and loss function properly.

4.3 Model Evaluation Tests:

- Test that the evaluation script correctly computes performance metrics such as accuracy, precision, recall, and AUC-ROC.
- Verify that visual interpretation functions overlay model predictions accurately on CT scan images.
- Ensure that the model evaluation is consistent across different datasets and environments.

4.4 Model Deployment Tests:

- Test that the Dockerfile creates a functional Docker image with all dependencies installed.
- Verify that MLflow deployment script correctly serves the trained model as a REST API endpoint.
- Ensure that the deployed model produces consistent predictions when queried with sample data.

4.5 Continuous Integration and Deployment (CI/CD) Tests:

- Test that CI/CD pipeline configurations are defined correctly and trigger automated testing and deployment.
- Verify that unit tests and integration tests execute successfully in the CI/CD pipeline.

- Ensure that automated deployment to production or staging environments is triggered upon successful completion of tests.

4.6 Monitoring and Maintenance Tests:

- Test that model monitoring script correctly detects changes in performance metrics or drift in predictions.
- Verify that scheduled retraining script initiates periodic retraining of the model using new data.
- Ensure that error logging mechanisms capture and report any issues in the production environment.