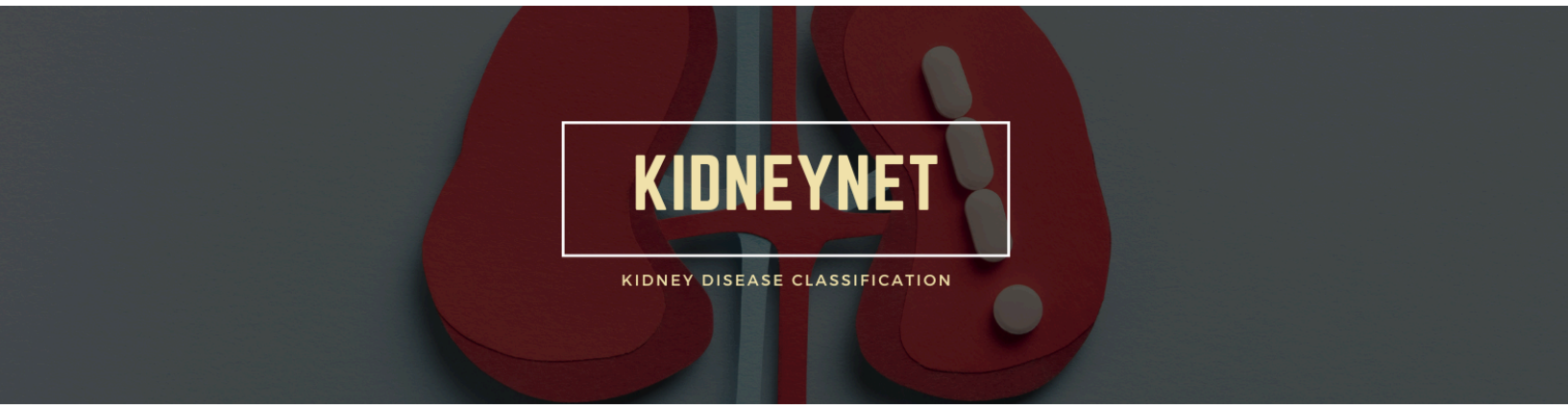
SNEHA KURMI



This document serves as a record of my project development process. Here, you can find:

- Doubts I encountered while building the project
- Questions I needed to answer through Google searches
- Errors I faced, along with the solutions I implemented

This documentation is particularly detailed because it's my first time using many of the project's libraries and tools, such as MLflow and DVC.

Navigating the Document:

- **For a quick overview:** Read the "Project Flow" section or the summaries provided on the first page for each stage.
    - Setup
    - Stage 01: Data Ingestion
    - Stage 02: Base Model
    - Stage 03: Model Training
    - Stage 04: Model Evaluation
    - Pipeline Version Control - DVC
    - Frontend
    - Docker

- **Looking for specific information?** Use the search functions within Google Docs:
    - **Errors I encountered:** Type " ❗ Error" in the search bar.
    - **Questions I had while coding this project:** Type " ❓ " in the search bar.

Set Up: Automating Project File Structure

template.py

template.py > ...

```python
import os
from pathlib import Path
import logging

# logging string
logging.basicConfig(level=logging.INFO, format='[%(asctime)s]: %(mes

project_name = 'cnnClassifier'

list_of_files = [
    ".github/workflows/.gitkeep",
    f"src/{project_name}/__init__.py",
    f"src/{project_name}/components/__init__.py",
    f"src/{project_name}/components/data_ingestion.py",
    f"src/{project_name}/utils/__init__.py",
    f"src/{project_name}/utils/file_op_utils.py"
    f"src/{project_name}/config/__init__.py",
    f"src/{project_name}/config/configuration.py",
    f"src/{project_name}/pipeline/__init__.py",
    f"src/{project_name}/pipeline/stage1_data_ingestion.py",
    f"src/{project_name}/entity/__init__.py"
```

Fig: template script code and output below

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
e c:/Users/Sneha/Documents/Projects/Kidney_Disease_Classification/template.py
[2024-04-30 23:12:15,058]: Creating directory: .github\workflows for the file: .gitkeep:
[2024-04-30 23:12:15,058]: Creating empty file: .github\workflows\.gitkeep:
[2024-04-30 23:12:15,061]: Creating directory: src\cnnClassifier for the file: __init__.py:
[2024-04-30 23:12:15,061]: Creating empty file: src\cnnClassifier\__init__.py:
[2024-04-30 23:12:15,061]: Creating directory: src\cnnClassifier\components for the file: __init__.py:
[2024-04-30 23:12:15,061]: Creating empty file: src\cnnClassifier\components\__init__.py:
[2024-04-30 23:12:15,063]: Creating directory: src\cnnClassifier\utils for the file: __init__.py:
[2024-04-30 23:12:15,063]: Creating empty file: src\cnnClassifier\utils\__init__.py:
[2024-04-30 23:12:15,063]: Creating directory: src\cnnClassifier\config for the file: __init__.py:
[2024-04-30 23:12:15,063]: Creating empty file: src\cnnClassifier\config\__init__.py:
[2024-04-30 23:12:15,063]: Creating directory: src\cnnClassifier\config for the file: configuration.py:
[2024-04-30 23:12:15,063]: Creating empty file: src\cnnClassifier\config\configuration.py:
[2024-04-30 23:12:15,067]: Creating directory: src\cnnClassifier\pipeline for the file: __init__.py:
[2024-04-30 23:12:15,067]: Creating empty file: src\cnnClassifier\pipeline\__init__.py:
[2024-04-30 23:12:15,067]: Creating directory: src\cnnClassifier\entity for the file: __init__.py:
[2024-04-30 23:12:15,067]: Creating empty file: src\cnnClassifier\entity\__init__.py:
[2024-04-30 23:12:15,069]: Creating directory: src\cnnClassifier\constants for the file: __init__.py:
[2024-04-30 23:12:15,069]: Creating empty file: src\cnnClassifier\constants\__init__.py:
[2024-04-30 23:12:15,069]: Creating directory: config for the file: config.yaml:
[2024-04-30 23:12:15,069]: Creating empty file: config\config.yaml:
[2024-04-30 23:12:15,069]: Creating empty file: dvc.yaml:
[2024-04-30 23:12:15,069]: Creating empty file: params.yaml:
[2024-04-30 23:12:15,069]: Creating empty file: requirements.txt:
[2024-04-30 23:12:15,069]: Creating empty file: setup.py:
[2024-04-30 23:12:15,069]: Creating directory: research for the file: trails.ipynb:
[2024-04-30 23:12:15,069]: Creating empty file: research\trails.ipynb:
```

The `__init__.py` files are used in Python to mark a directory as a Python package, which means it can be imported as a module. Here's a bit more detail:

- When Python encounters a directory containing an `__init__.py` file, it treats that directory as a package.
- This file is often empty, but it can also contain valid Python code.
- This code will be executed when the package is imported, allowing you to initialize package-level variables or perform other setup tasks

In the context of my code, the `__init__.py` files are being created in several directories under the `src/cnnClassifier` directory. This allows each of these directories to be imported as a module in Python, which can be useful for organizing your code into logical, reusable components.

In a machine learning project, you might have different modules for data preprocessing, model training, evaluation, etc. By using `__init__.py` files and organizing these components into packages, you can keep your project organized and modular.

Setting Up a Virtual Environment for a New Project

```
C:\Windows\System32\cmd.e

C:\Users\Sneha\Documents\Projects\KidneyNet>conda create -n kidney python==3.8
Collecting package metadata (current_repodata.json): done
Solving environment: unsuccessful attempt using repodata from current_repodata.json, retrying with next repodata source.
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Sneha\anaconda3\envs\kidney

  added / updated specs:
    - python==3.8


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2024.3.11  |       haa95532_0         128 KB
    pip-23.3.1                 |   py38haa95532_0         2.8 MB
    python-3.8.0               |       hff0d562_2        15.9 MB
    setuptools-68.2.2          |   py38haa95532_0         933 KB
    sqlite-3.45.3              |       h2bbff1b_0         973 KB
    wheel-0.41.2               |   py38haa95532_0         126 KB
    ------------------------------------------------------------
                                           Total:        20.8 MB

The following NEW packages will be INSTALLED:

  ca-certificates    pkgs/main/win-64::ca-certificates-2024.3.11-haa95532_0
```

```
C:\Users\Sneha\Documents\Projects\KidneyNet>conda activate kidney

(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>
```

Fig: Creating a virtual environment called ‘kidney’

Once the VE is activated in the project folder, we can go ahead and instead/run the requirements.py folder for all the necessary libraries installation.

Requirements.txt

❓ How to automate writing this file?

Using `pip freeze`

1. **Install your project's dependencies**: Ensure all your project's dependencies are installed in your virtual environment.

Generate the `requirements.txt` file:

```
pip freeze > requirements.txt
```

2. This command will list all installed packages and their versions, outputting them to the `requirements.txt` file.

Using `pipreqs`

`pipreqs` scans your project directory and generates a `requirements.txt` file based on the imports it finds in your code.

Install `pipreqs`:

```
pip install pipreqs
```

1.

Run `pipreqs` in your project directory:

```
pipreqs /path/to/your/project
```

2. This command will create a `requirements.txt` file in your project directory.

```
requirements.txt U ×
requirements.txt
1   tensorflow==2.12.0
2   pandas
3   dvc
4   mlflow==2.2.2
5   notebook
6   numpy
7   matplotlib
8   seaborn
9   python-box==6.0.2
10  pyYAML
11  tqdm
12  ensure==1.0.2
13  joblib
14  types-PyYAML
15  scipy
16  Flask
17  Flask-Cors
18  gdown
19  -e .
```

Fig. Created a requirements.txt file for all the necessary libraries download for easy importing later on

The -e option in a Python requirements file stands for "editable". When you see -e . in a requirements file, it tells pip, the Python package installer, to install the package in the current directory (represented by the dot) in “editable” mode.

In editable mode, changes to the source code of the package will be reflected immediately in your environment without needing to reinstall the package. This is particularly useful during development when you’re making frequent changes to the package.

Link Creation: When you run pip install -e ., pip creates a .egg-link file in the site-packages directory. This file contains the path to the source code of the package.

Path Addition: Pip also updates an easy-install.pth file in the site-packages directory, adding the path to your package. This file is used by Python to determine which directories to search when importing packages.

Development Mode: The package is now installed in "development mode". Any changes you make to the source code will be immediately available in your Python environment, without needing to reinstall the package

Logging Code Setup

src > cnnClassifier > __init__.py > ...

```python
import os
import sys
import logging

logging_str = "[%(asctime)s: %(levelname)s: %(module)s: %(message)s

log_dir = "logs"
log_filepath = os.path.join(log_dir,"running_logs.log")
os.makedirs(log_dir, exist_ok=True)


logging.basicConfig(
    level= logging.INFO,
    format= logging_str,

    handlers=[
        logging.FileHandler(log_filepath),
        logging.StreamHandler(sys.stdout)
    ]
)

logger = logging.getLogger("cnnClassifierLogger")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python

```
PS C:\Users\Sneha\Documents\Projects\KidneyNet> & C:/Users/Sneha/anaconda3/python.exe c:/Us
ers/Sneha/Documents/Projects/KidneyNet/src/cnnClassifier/__init__.py
PS C:\Users\Sneha\Documents\Projects\KidneyNet>
```

Fig: Setting up logging code

1. Importing Required Modules:

The os and sys modules provide numerous functions to interact with the system. The logging module is used to log messages for application events.

2. Setting Up the Logging String:

This string sets the format of the log messages. It includes the time the log message was created (%(asctime)s), the severity level of the log (%(levelname)s), the module where the log message originated (%(module)s), and the log message itself (%(message)s).

3. Setting Up the Log Directory and File Path:

The log_dir variable is set to the string “logs”, which is the name of the directory where the log file will be stored. The log_filepath variable is set to the path of the log file within the log directory. The os.makedirs() function is used to ensure that the log directory exists; if it doesn’t, the directory is created.

4. Configuring the Logger:

The logging.basicConfig() function is used to configure the logging system. The level parameter sets the root logger to the specified severity level (logging.INFO in this case). The format parameter sets the format of the log message. The handlers parameter is a list that specifies where the log messages will be sent.

5. Getting the Logger:

The logging.getLogger() function returns a logger with the specified name. This logger can then be used to write log messages.

This setup allows for detailed logging of events in an application, which can be very useful for debugging and understanding the flow of the program. The log messages will be written to the console and a file, making it easy to review the events later. The logging module also provides flexibility to log messages of different severity levels, ranging from debug messages to critical errors. This can help a lot in managing and maintaining large applications.

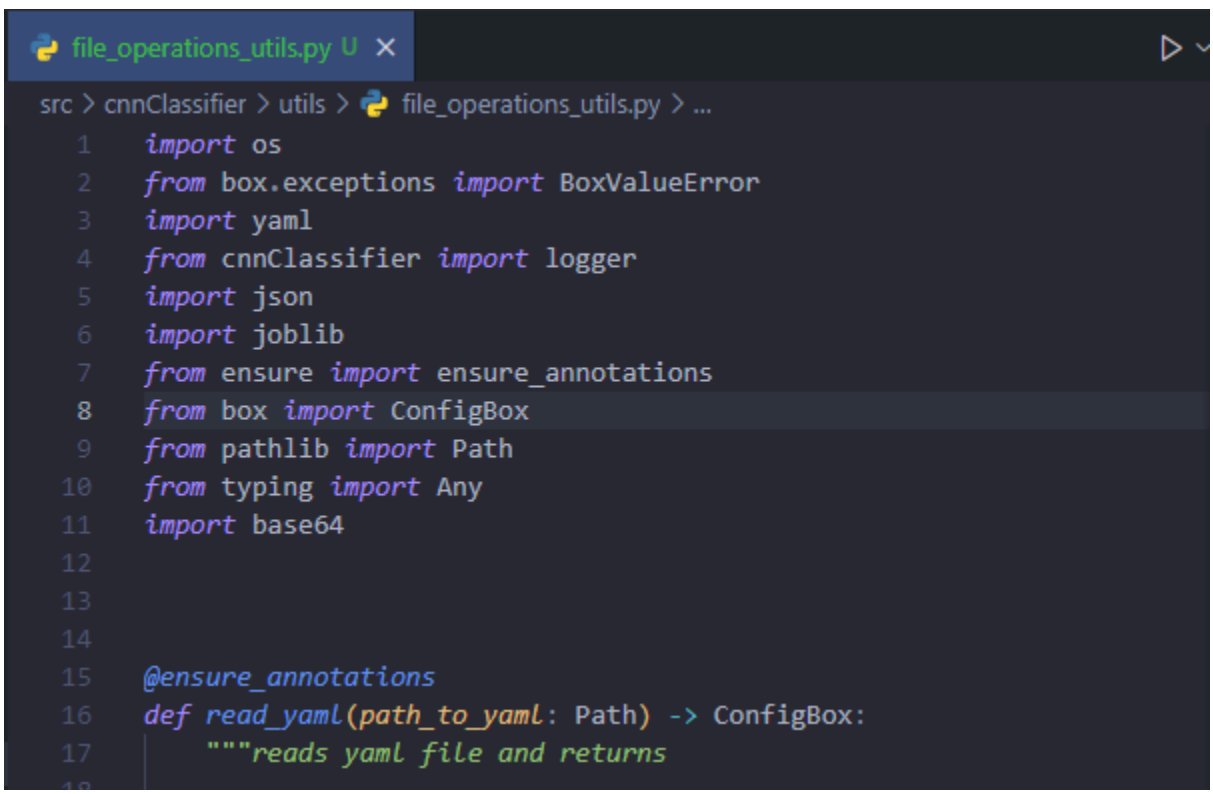


Fig: Script for some basic functions for creating files, saving, loading, etc

This Python script contains a collection of utility functions that perform various file operations, including reading and writing YAML, JSON, and binary files, creating directories, and encoding and decoding images. Here's a detailed explanation of each function:

- `read_yaml`: This function reads a YAML file and returns its contents as a ConfigBox object. If the YAML file is empty, it raises a ValueError.
- `create_directories`: This function creates a list of directories. If a directory already exists, it will not raise an error due to the exist_ok=True parameter in os.makedirs().
- `save_json`: This function saves a dictionary as a JSON file.
- `load_json`: This function loads a JSON file and returns its contents as a ConfigBox object.
- `save_bin`: This function saves data as a binary file using the `joblib.dump()` function.
- `load_bin`: This function loads data from a binary file using the `joblib.load()` function.
- `get_size`: This function returns the size of a file in kilobytes.
- decodeImage: This function decodes a base64 encoded image and saves it as a file.
- encodeImageIntoBase64: This function reads an image file, encodes it as base64, and returns the encoded data.

The ensure_annotations decorator is used to ensure that the function arguments match the types specified in the function's annotations.

e is a variable that is used to hold the instance of the exception that was raised. This is a common pattern in Python for handling exceptions. When an exception is raised in the try block, the except block catches it and the instance of the exception is assigned to the variable e. You can then use this variable to get more information about the exception or to re-raise the exception using raise e.

The code checks if the YAML file is empty by trying to load the file with yaml.safe_load(yaml_file). If the file is empty, this function will return None, and when None is passed to ConfigBox, it raises a BoxValueError. So, if a BoxValueError is caught, the code assumes that the YAML file is empty and raises a ValueError with the message "yaml file is empty".

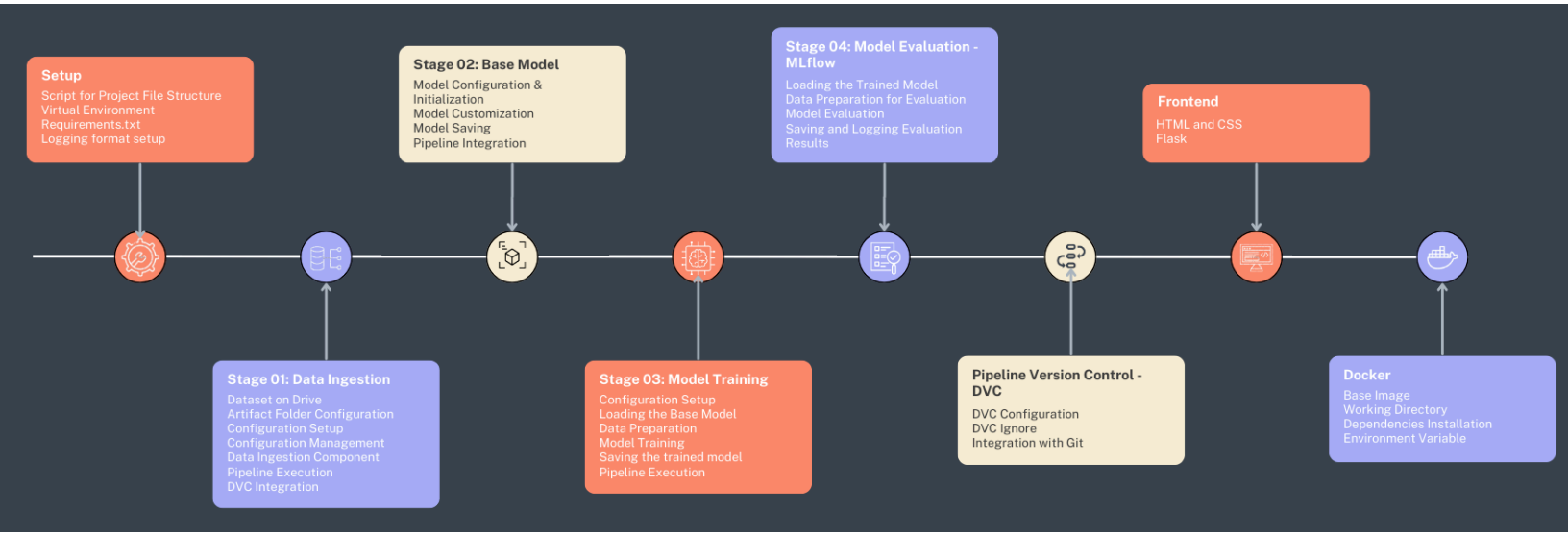
It's important to note that this is a somewhat indirect way of checking if the file is empty, and it might not always be accurate. For example, if the file contains only comments or only whitespace, yaml.safe_load() will also return None, even though the file is not technically empty. A more direct way to check if a file is empty would be to read the file and check if the result is an empty string. However, for the purpose of loading a YAML file, treating a file with only comments or whitespace as empty might be the desired behavior

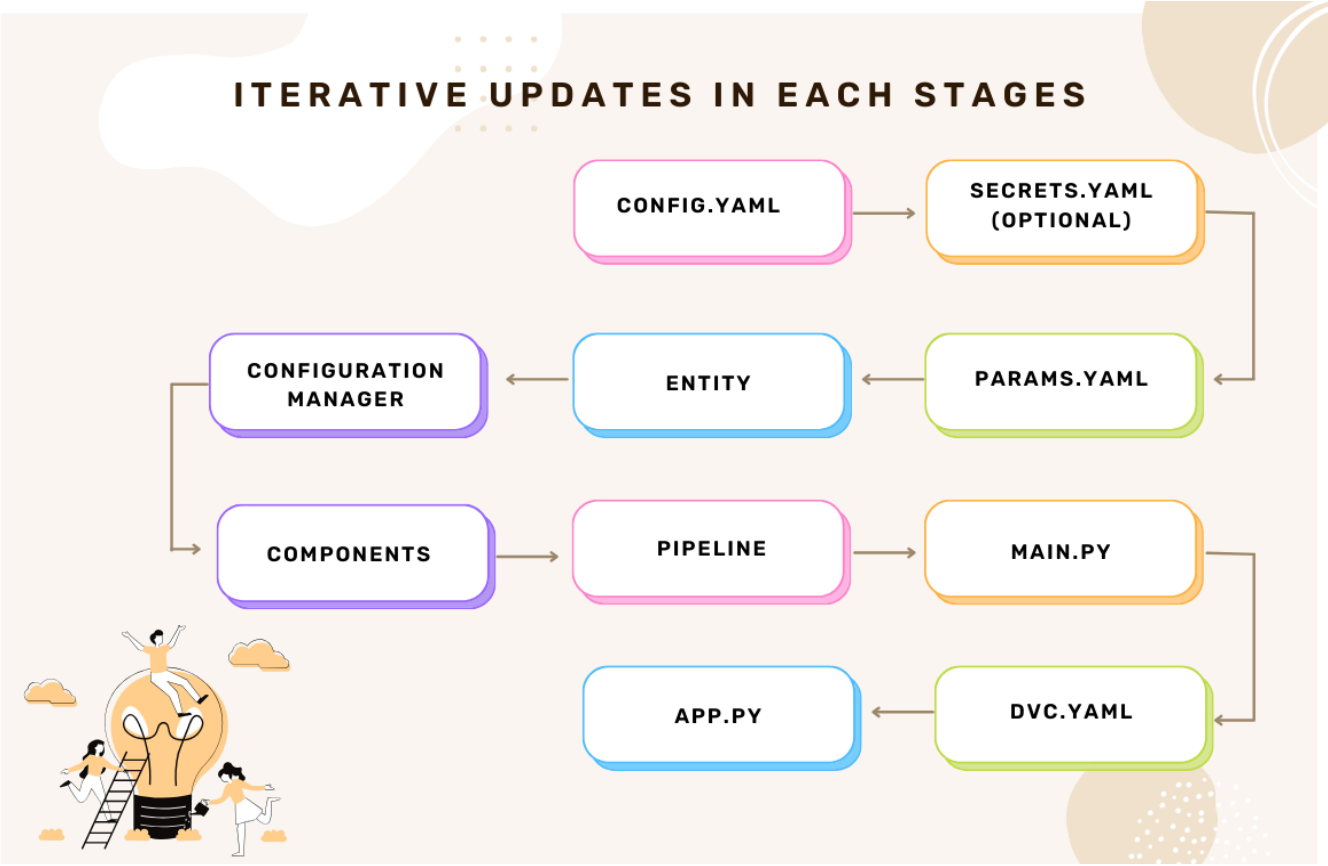
Data serialization is the process of converting data objects present in complex data structures into a byte stream for storage, transfer, and distribution purposes on physical devices

This process allows data to be stored (e.g., in files or databases), transmitted (e.g., across a network), and then reconstructed later, possibly in a different computer environment

In Python, a decorator is like the costume. It’s a special type of function that adds some new behavior to an existing function, without changing the underlying function itself.

Project Flow





Analysis of the KidneyNet Project Structure

The KidneyNet project is structured into multiple folders, each serving a specific purpose in the machine learning pipeline for kidney disease classification. Here's a breakdown of the key folders and their roles:

1. src
Purpose: Contains the source code for the project.
Contents:

cnnClassifier: This sub-folder contains the Python scripts for the convolutional neural network model's pipeline, including data ingestion, model preparation, training, and evaluation.
components: Reusable code components used across different stages of the model pipeline.
Connection: Interacts with the artifacts folder to save and retrieve model data, and with configuration files for parameters.

2. config
Purpose: Stores configuration files like config.yaml which are crucial for parameterizing the stages of the ML pipeline.
Connection: Used by the Python scripts in src to load configurations dynamically, ensuring flexibility and ease of adjustments.

3. artifacts
Purpose: Stores outputs from various stages of the ML pipeline, such as processed datasets, model files, and evaluation metrics.
Contents:

data_ingestion: Stores datasets processed during the data ingestion stage.
prepare_base_model: Contains base model files prepared for training.
training: Stores trained model files.

Connection: Generated and used by scripts in the src folder. Managed by DVC to track changes and versions of the data artifacts.

4. mlruns
Purpose: Used by MLflow to log and store experiment metrics, parameters, and model versions.
Connection: Interacts with the MLflow tracking server, providing a UI to compare and manage different runs and models.

5. notebooks
Purpose: Contains Jupyter notebooks for exploratory data analysis, model experimentation, and results visualization.
Connection: May use data from artifacts for analysis and can output figures or models back to artifacts.

6. logs
Purpose: Stores log files from the application, useful for debugging and tracking the application's operational history.
Connection: Generated by the application during runtime, helps in monitoring and troubleshooting.

7. Dockerfile
Purpose: Defines the environment and instructions for building a Docker container for the project, ensuring consistency across different environments.
Connection: Packages the application with all its dependencies into a container, which can then be run in any environment supporting Docker.

8. dvc.yaml
Purpose: Configuration file for DVC pipelines, defining stages of the ML pipeline and their dependencies.
Connection: Orchestrates the execution of different stages defined in the src folder, ensuring reproducibility and version control of data and models.

Why Each Folder is Needed:

- src: Core application logic.
- config: Centralized and flexible configuration management.
- artifacts: Version-controlled storage of data and models, crucial for reproducibility.
- mlruns: Tracking and managing experiments.
- notebooks: Interactive platform for analysis and visualization.
- logs: Operational monitoring and debugging.
- Dockerfile: Environment consistency and deployment.
- dvc.yaml: Pipeline definition and data version control.
- This structure supports a robust ML development workflow, integrating tools like MLflow for experiment tracking, Docker for environment management, and DVC for data and pipeline version control, crucial for scalable and reproducible ML projects.

Other Key Folders and Files in `cnnClassifier`:

1. components
Purpose: Contains reusable components that are used across different stages of the machine learning pipeline. For example, model training and evaluation components.
Need: Promotes code reusability and modularity, simplifying maintenance and scalability of the application.

2. config
Purpose: Houses configuration management setup, including the configuration.py file which loads and processes configuration files.

Need: Provides a centralized place for managing configurations, allowing easy adjustments and ensuring consistency across different parts of the application.

3. constants
Purpose: Defines constant values used throughout the application.
Need: Centralizes the definition of constants to ensure consistency and ease of updates.

4. entity
Purpose: This folder, as seen in the config_entity.py file, defines data classes for configuration entities. These entities represent the configuration for different stages of the machine learning pipeline, such as data ingestion, model preparation, training, and evaluation.
Need: Organizes and encapsulates related data structures, making the code cleaner and more manageable.

5. pipeline
Purpose: Contains scripts that define the stages of the machine learning pipeline, such as data ingestion, model preparation, training, and evaluation.
Need: Separates the pipeline into distinct stages, making the process clear, manageable, and modular.

6. utils
Purpose: Provides utility functions that are used across the application, such as file operations and data handling utilities.
Need: Encapsulates common functionality to avoid duplication and simplify maintenance.

Stage 01: Data Ingestion

Summary Data Ingestion

Data Ingestion in my Codebase

Data ingestion in this project is managed through a series of Python scripts and configurations that handle the downloading and extraction of data necessary for training the kidney disease classification model. Here's how it is structured:

1. Configuration Setup

The data ingestion configuration is defined in the `DataIngestionConfig` class within the `config_entity.py` file. This configuration includes paths and URLs necessary for the data ingestion process.

2. Configuration Management

The `ConfigurationManager` class in configuration.py is responsible for loading these configurations from YAML files and providing an instance of `DataIngestionConfig.`

3. Data Ingestion Component

The DataIngestion class in `data_ingestion.py` uses the configuration to perform the actual data downloading and extraction. It downloads the dataset from a specified URL and extracts it to a designated directory.

> Downloading: The dataset is downloaded using the gdown library, which is suitable for files hosted on Google Drive.
> Extraction: After downloading, the dataset is extracted using Python's zipfile module.

4. Pipeline Execution

The ingestion process is orchestrated by the `DataIngestionTrainingPipeline` class in `stage1_data_ingestion.py`. This class initializes the DataIngestion component and calls its methods to perform the download and extraction tasks.

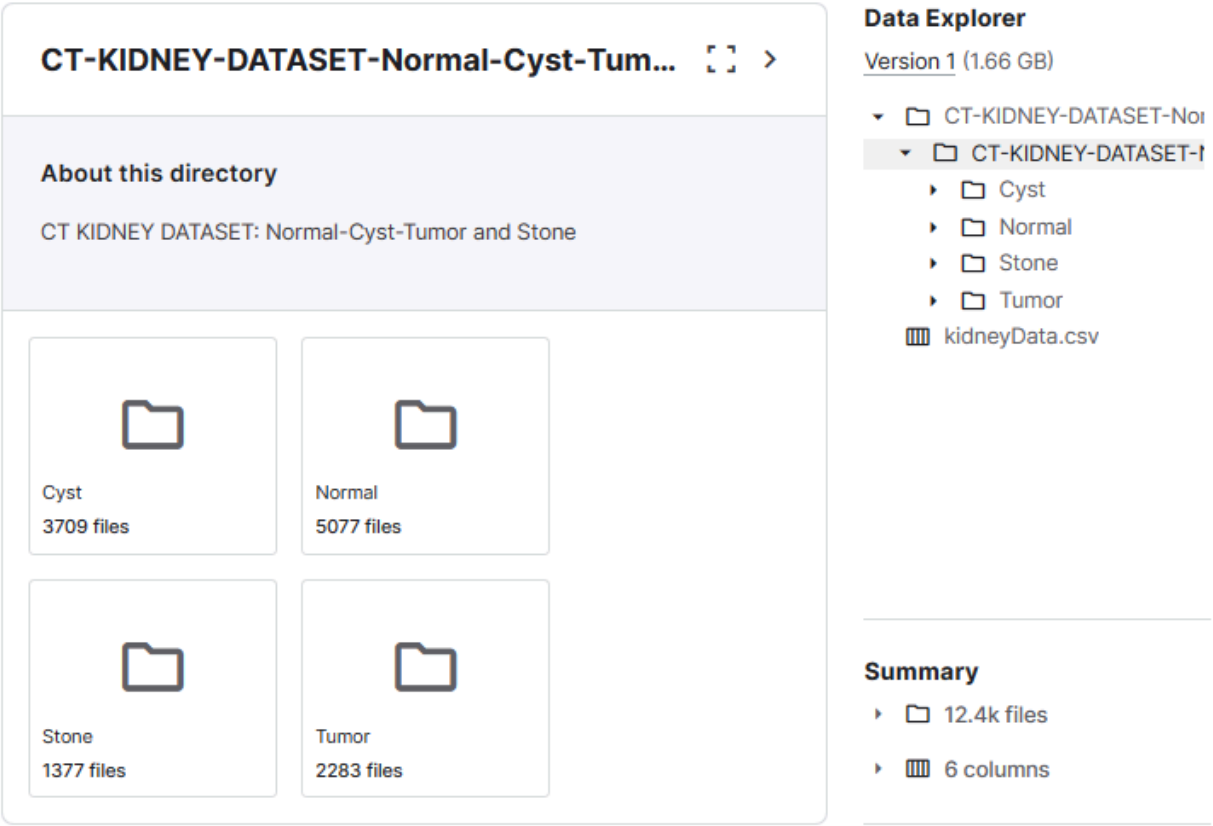
5. DVC Integration

The data ingestion stage is also integrated into a DVC pipeline, ensuring reproducibility and version control of the data. The `dvc.yaml` file specifies the command to run the data ingestion script and tracks the output directory.

This setup ensures that the data ingestion process is well-organized, configurable, and integrated with the version control system, making it robust and reproducible.

DataSet

🔗[CT KIDNEY DATASET: Normal-Cyst-Tumor and Stone (kaggle.com)]



As we can see the data size is almost 2GB, I will be using GDrive to read the data, you use any other cloud services as well like Amazon S3 cloud storage, etc.

Basic Details:

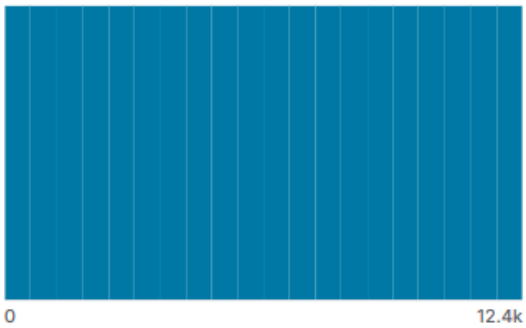
4 Classes/Target
- Cyst
- Normal
- Stone
- Tumor

kidneyData.csv (1.51 MB)

Detail Compact Column 6 of 6 columns

#

Serial No.



Valid	12.4k	100%
Mismatched	0	0%
Missing	0	0%
Mean	6.22k	
Std. Deviation	3.59k	
Quantiles	0	Min
	3111	25%
	6223	50%
	9334	75%
	12.4k	Max

image_id

Image ID

12446
unique values

Valid	12.4k	100%
Mismatched	0	0%
Missing	0	0%
Unique	12.4k	
Most Common	Tumor- (10...	0%

path

Image Path

12446
unique values

Valid	12.4k	100%
Mismatched	0	0%
Missing	0	0%
Unique	12.4k	
Most Common	/content/da...	0%

diag

Kidney Normal, Cyst, Tumor and Stone

Normal	41%
Cyst	30%
Other (3660)	29%

Valid	12.4k	100%
Mismatched	0	0%
Missing	0	0%
Unique	4	
Most Common	Normal	41%

target

Tumor-3; Normal-1; Cyst-0; Stone-2

Valid	12.4k	100%
Mismatched	0	0%
Missing	0	0%
Mean	1.18	
Std. Deviation	1.05	
Quantiles	0	Min
	0	25%
	1	50%
	2	75%
	3	Max

Class

Tumor-3; Normal-1; Cyst-0; Stone-2

Normal	41%
Cyst	30%
Other (3660)	29%

Valid	12.4k	100%
Mismatched	0	0%
Missing	0	0%
Unique	4	
Most Common	Normal	41%

Upload the data zip file to Google Drive
After uploading, copy the link address > access to anyone > copy link

config.yaml U | data_ingestion.ipynb U | trails.ipynb U

research > trails.ipynb > ...

+ Code + Markdown | Interrupt Restart ··· kidney (Python 3.8.0)

Downloading Dataset from GDrive

```python
import gdown

url = 'https://drive.google.com/file/d/18JJ5SvjkqQvRgcSwqAUCSygnDnM8
```

[1] ✓ 0.2s Python

```python
file_id = url.split("/")[-2]
file_id
```

[2] ✓ 0.0s Python

```
'18JJ5SvjkqQvRgcSwqAUCSygnDnM8wfwj'
```

```python
prefix = 'https://drive.google.com/uc?/export=download&id='
gdown.download(prefix+file_id, "Kidney-CT-Scan-data.zip")
```

[3] 39.3s Python

```
Downloading...
From (original): https://drive.google.com/uc?/export=download&id=18JJ5Sv
From (redirected): https://drive.google.com/uc?%2Fexport=download&id=18J
To: c:\Users\Sneha\Documents\Projects\KidneyNet\research\Kidney-CT-Scan-
 25%|       | 412M/1.63G [00:37<01:03, 19.0MB/s]
```

Fig. Downloading the dataset from gdrive

Artifacts Configuration or Config folder

Fig. configuration for data ingestion

An artifact is a machine learning term that describes the output (a fully trained model, a model checkpoint, or a file) created by the training process.

`artifacts_root: artifacts`: This sets the root directory for the artifacts (i.e., output data or results) as the artifacts folder.

Stage 01:

The `data_ingestion` section seems to be related to the process of importing data into the application:

`root_dir: artifacts/data_ingestion`: This sets the directory for data ingestion related artifacts.
source_URL: https://drive.google.com/file/d/1vlhZ5c7abUKF8xXERIw6m9Te8fW7ohw3/view?usp=sharing: This is the URL where the data file is located.

`local_data_file: artifacts/data_ingestion/data.zip`: This is the path where the data file will be saved locally after it's downloaded from the source_URL.

`unzip_dir: artifacts/data_ingestion`: This is the directory where the data file will be unzipped after it's downloaded.

Entity file

```python
from dataclasses import dataclass
from pathlib import Path


@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path

```

Fig. Linked to the config.yaml file

```yaml
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL: https://drive.google.com/file/d/18JJ5SvjkqQvRgc
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion

```

Entity.py and DataIngestionConfig class: The entity.py file typically contains classes that represent entities in your project. In this case, it contains a DataIngestionConfig class. This class is a Python dataclass, which is a simple way to create classes that primarily store data. The DataIngestionConfig class has four attributes: root_dir, source_URL, local_data_file, and unzip_dir. These attributes are likely used to configure the data ingestion process in a deep learning project.

`root_dir`: The root directory where the data ingestion process will take place.
`source_URL`: The URL from where the data file will be downloaded.
`local_data_file`: The local path where the downloaded data file will be saved.
`unzip_dir`: The directory where the data file will be unzipped after download.

In an end-to-end deep learning project, such a setup helps in maintaining a clean and organized project structure. It also allows for easy configuration of different stages of the project, like data ingestion in this case.

The DataIngestionConfig class can be used to create an object with all the necessary configuration for the data ingestion process, which can then be passed around to different functions or methods that need this information. This makes the code more modular, readable, and maintainable.

Stage 02: files or section below this point was updated after stage 02

```
config.yaml    config_entity.py    params.yaml
src > cnnClassifier > entity > config_entity.py > ...
from dataclasses import dataclass
from pathlib import Path


@dataclass(frozen=True)
class DataIngestionConfig:
    root_dir: Path
    source_URL: str
    local_data_file: Path
    unzip_dir: Path

@dataclass(frozen=True)
class PrepareBaseModelConfig:
    root_dir: Path
    base_model_path: Path
    updated_base_model_path: Path
    params_image_size: list
    params_learning_rate: float
    params_include_top: bool
    params_weights: str
    params_classes: int
```

PrepareBaseModelConfig using the @dataclass decorator. Let's break it down:

1. @dataclass(frozen=True):
    - The @dataclass decorator is used to automatically generate special methods (such as __init__, __repr__, and __eq__) for a class based on its attributes.

    - The frozen=True argument means that instances of this class will be immutable once created. In other words, their attributes cannot be modified after initialization.
    - When a data class is frozen, it ensures that the values of its attributes remain constant, which can be useful for certain scenarios (e.g., using instances as dictionary keys or set elements).

2. PrepareBaseModelConfig class:
    - This class represents configuration settings related to preparing a base machine learning model.
    - It has the following attributes:
        - root_dir: A Path object representing the root directory.
        - base_model_path: A Path object representing the path to the base model.
        - updated_base_model_path: A Path object representing the path to the updated base model.
        - params_image_size: A list containing image size parameters.
        - params_learning_rate: A float representing the learning rate.
        - params_include_top: A boolean indicating whether to include the top layers in the model.
        - params_weights: A string specifying the weights to use.
        - params_classes: An integer representing the number of classes.

3. ConfigurationManager class:
    - This class manages configuration settings for a machine learning application.
    - It reads configuration data from YAML files (config_filepath and params_filepath).
    - The get_prepare_base_model_config method returns an instance of PrepareBaseModelConfig with the specified configuration values.

In summary, the code defines a data class (PrepareBaseModelConfig) with frozen instances, ensuring that the configuration settings remain constant throughout the program execution. The ConfigurationManager class handles reading and managing these settings.

Constant file

Fig. path to some import files noted in this constant folder

This code is defining constants that represent the paths to two important configuration files in your project: config.yaml and params.yaml.

CONFIG_FILE_PATH = Path("config/config.yaml"): This line is creating a Path object that represents the path to the config.yaml file. This file typically contains various configuration options for your project.

PARAMS_FILE_PATH = Path("params.yaml"): Similarly, this line is creating a Path object for the params.yaml file. This file might contain parameters for your project, such as hyperparameters for a machine learning model.

The reason this code is placed in the constants directory is that these paths are likely used throughout your project, and it's good practice to define such commonly used values in a central location. This way, if the paths ever need to be changed, you can do so in one place, and the changes will propagate throughout your code. This helps keep your code DRY (Don't Repeat Yourself), making it more maintainable and less prone to errors.

Configuration Manager

configuration.py

src > cnnClassifier > config > configuration.py > ConfigurationManager > get_data_ingestion_config

```python
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.file_op_utils import read_yaml, create_dire
from cnnClassifier.entity.config_entity import DataIngestionConfig

class ConfigurationManager:
    def __init__(
        self,
        config_filepath = CONFIG_FILE_PATH,
        params_filepath = PARAMS_FILE_PATH):

        self.config = read_yaml(config_filepath)
        self.params = read_yaml(params_filepath)

        create_directories([self.config.artifacts_root])



    def get_data_ingestion_config(self) -> DataIngestionConfig:
        config = self.config.data_ingestion

        create_directories([config.root_dir])

        data_ingestion_config = DataIngestionConfig(
            root_dir=config.root_dir,
            source_URL=config.source_URL,
            local_data_file=config.local_data_file,
            unzip_dir=config.unzip_dir
        )

        return data_ingestion_config
```

Configuration Manager: The ConfigurationManager class is responsible for managing the configuration of the project. It reads the configuration and parameters from the config.yaml and params.yaml files respectively.

It also creates the directories necessary for the project. The get_data_ingestion_config method returns a DataIngestionConfig object that contains the configuration for the data ingestion process. The -> DataIngestionConfig part is a type hint that indicates the return type of the method. Type hints are optional in Python, but they can make your code easier to understand and debug.

Stage 02: Base Model

```python
def get_prepare_base_model_config(self) -> PrepareBaseModelConfig:
    config = self.config.prepare_base_model

    create_directories([config.root_dir])

    prepare_base_model_config = PrepareBaseModelConfig(
        root_dir=Path(config.root_dir),
        base_model_path=Path(config.base_model_path),
        updated_base_model_path=Path(config.updated_base_model_path),
        params_image_size=self.params.IMAGE_SIZE,
        params_learning_rate=self.params.LEARNING_RATE,
        params_include_top=self.params.INCLUDE_TOP,
        params_weights=self.params.WEIGHTS,
        params_classes=self.params.CLASSES
    )

    return prepare_base_model_config
```

Components folder

Stage 01: Data Ingestion

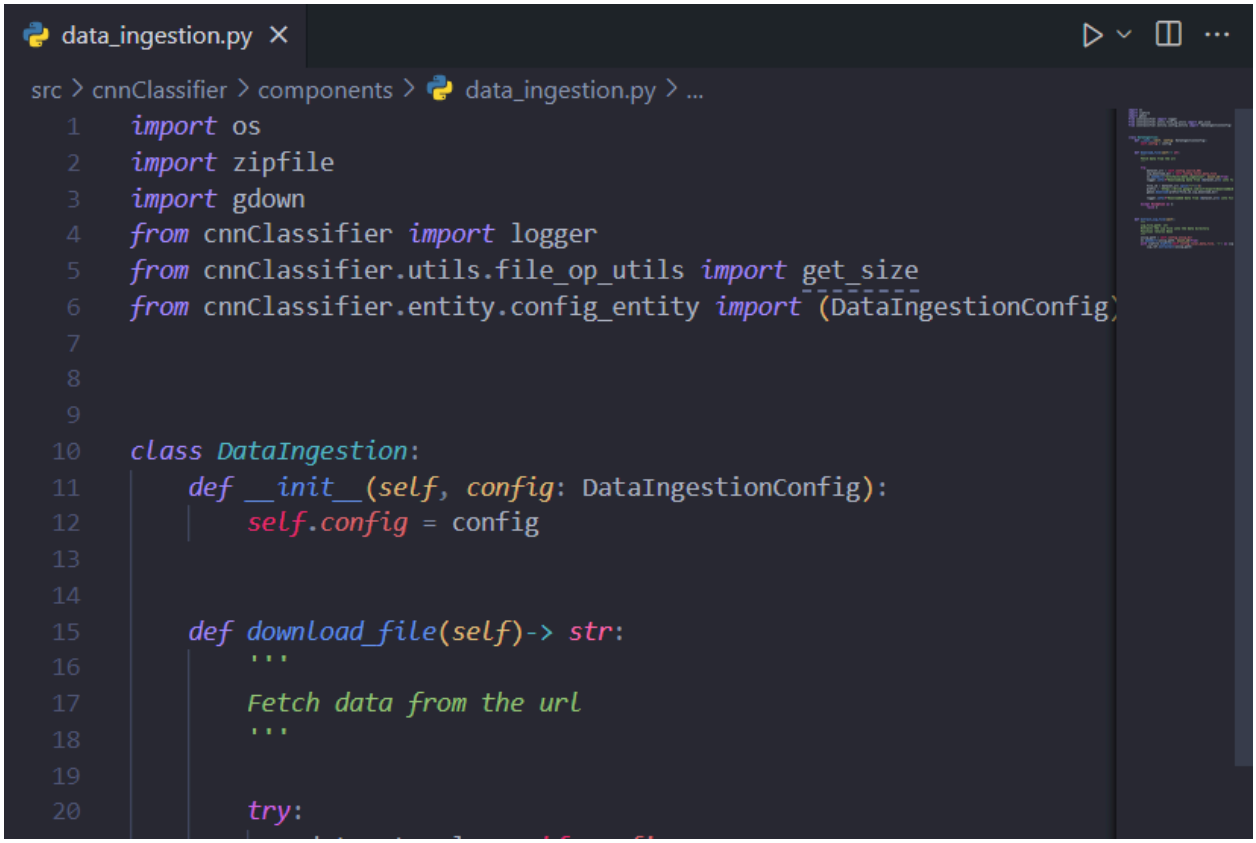


Fig. Deciding directory for artifacts folder

Stage 02: Base Model

configuration.py | prepare_base_model.py | data_ingestion.py | stage2_prepare_base_n

cnnClassifier > components > prepare_base_model.py > PrepareBaseModel > get_base_model

```python
import os
import urllib.request as request
from zipfile import ZipFile
import tensorflow as tf
from pathlib import Path
from cnnClassifier.entity.config_entity import PrepareBaseModelConfig


class PrepareBaseModel:
    def __init__(self, config: PrepareBaseModelConfig):
        self.config = config


    def get_base_model(self):
        self.model = tf.keras.applications.xception.Xception(
            input_shape=self.config.params_image_size,
            weights=self.config.params_weights,
            include_top=self.config.params_include_top
        )

        self.save_model(path=self.config.base_model_path, model=self.model)



    @staticmethod
    def prepare_full_model(model, classes, freeze_all, freeze_till, learning_rate):
```

Fig. Created a new folder prepare_base_model.py in components folder and base_model.h5 and base_model_updated.h5 as output in artifacts folder

Pipeline Folder

guration.py | prepare_base_model.py | stage2_prepare_base_model.py | main.py

src > cnnClassifier > pipeline > stage2_prepare_base_model.py > ...

```python
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.prepare_base_model import PrepareBaseModel
from cnnClassifier import logger


STAGE_NAME = "Prepare base model"


class PrepareBaseModelTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        prepare_base_model_config = config.get_prepare_base_model_config()
        prepare_base_model = PrepareBaseModel(config=prepare_base_model_config
        prepare_base_model.get_base_model()
        prepare_base_model.update_base_model()



if __name__ == '__main__':
    try:
        logger.info(f"*******************")
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = PrepareBaseModelTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx=========
    except Exception as e:
        logger.exception(e)
        raise e
```

Fig. stage2_prepare_base_model.py file in pipeline folder

Main.py & Final Run for Data Ingestion Stage

main.py

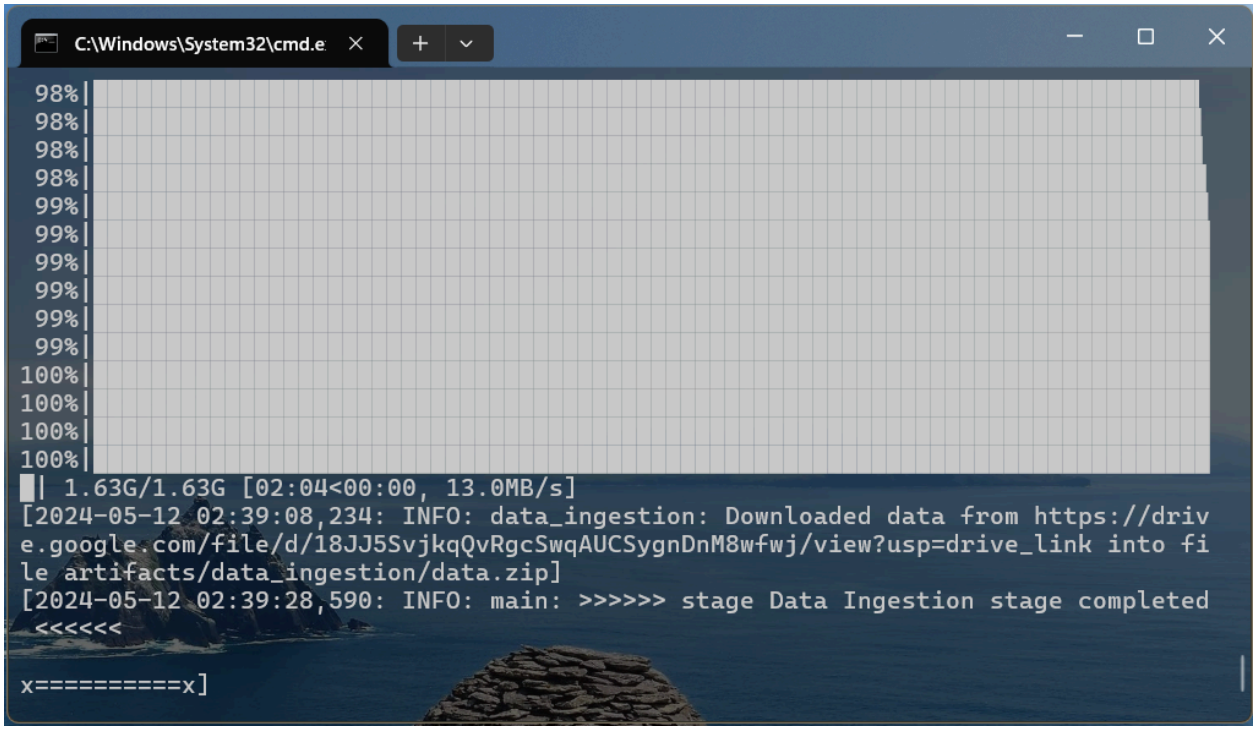
```python
from cnnClassifier import logger
from cnnClassifier.pipeline.stage1_data_ingestion import DataIngesti



STAGE_NAME = "Data Ingestion stage"
try:
    logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
    data_ingestion = DataIngestionTrainingPipeline()
    data_ingestion.main()
    logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx====
except Exception as e:
        logger.exception(e)
        raise e
```

C:\Windows\System32\cmd.e

```
(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>python main.py
[2024-05-12 02:37:01,313: INFO: main: >>>>>> stage Data Ingestion stage started <
<<<<<]
[2024-05-12 02:37:01,313: INFO: file_op_utils: yaml file: config\config.yaml load
ed successfully]
[2024-05-12 02:37:01,313: INFO: file_op_utils: yaml file: params.yaml loaded succ
essfully]
[2024-05-12 02:37:01,313: INFO: file_op_utils: created directory at: artifacts]
[2024-05-12 02:37:01,313: INFO: file_op_utils: created directory at: artifacts/da
ta_ingestion]
[2024-05-12 02:37:01,313: INFO: data_ingestion: Downloading data from https://dri
ve.google.com/file/d/18JJ5SvjkqQvRgcSwqAUCSygnDnM8wfwj/view?usp=drive_link into f
ile artifacts/data_ingestion/data.zip]
Downloading...
From (original): https://drive.google.com/uc?/export=download&id=18JJ5SvjkqQvRgcS
wqAUCSygnDnM8wfwj
From (redirected): https://drive.google.com/uc?%2Fexport=download&id=18JJ5SvjkqQv
RgcSwqAUCSygnDnM8wfwj&confirm=t&uuid=dea5a077-1d0c-4d1b-bbb6-25c28e333a1e
To: C:\Users\Sneha\Documents\Projects\KidneyNet\artifacts\data_ingestion\data.zip
  1%|
| 21.5M/  1%|
         | 23.1M/1.63G [00:04<04:00  2%|
```


C:\Windows\System32\cmd.e
98%|
98%|
98%|
98%|
99%|
99%|
99%|
99%|
99%|
99%|
100%|
100%|
100%|
100%|
| 1.63G/1.63G [02:04<00:00, 13.0MB/s]
[2024-05-12 02:39:08,234: INFO: data_ingestion: Downloaded data from https://drive.google.com/file/d/18JJ5SvjkqQvRgcSwqAUCSygnDnM8wfwj/view?usp=drive_link into file artifacts/data_ingestion/data.zip]
[2024-05-12 02:39:28,590: INFO: main: >>>>>> stage Data Ingestion stage completed <<<<<<
x==========x]

Stage 02: Base Model

Summary

Base Model Creation in my Codebase
After data ingestion, the base model creation in this codebase is handled by the `PrepareBaseModel` class, which is part of the machine learning pipeline for kidney disease classification. Here's how the base model is prepared:

1. Model Configuration

The configuration for the base model is managed by the `PrepareBaseModelConfig` class, which includes parameters such as image size, weights, and whether to include the top layer of the model.

2. Model Initialization and Configuration

The `PrepareBaseModel` class initializes the base model using the TensorFlow Keras library. It specifically uses the Xception model, a deep learning model pre-trained on the ImageNet dataset, suitable for image classification tasks.

3. Model Customization

The base model is then customized for the specific task by adding a new top layer to adapt it to the number of classes (4 classes: Cyst, Normal, Stone, Tumor). This is done in the `_prepare_full_model` method, where the top layers are configured based on the training requirements, such as whether to freeze all layers or just a portion.

4. Model Saving

After customization, the model is saved using the `save_model` method, which uses TensorFlow's save function to store the model at the specified path.

5. Pipeline Integration

The entire process of preparing the base model is orchestrated by the `PrepareBaseModelTrainingPipeline` class in the `stage2_prepare_base_model.py` script, which is part of the DVC pipeline.

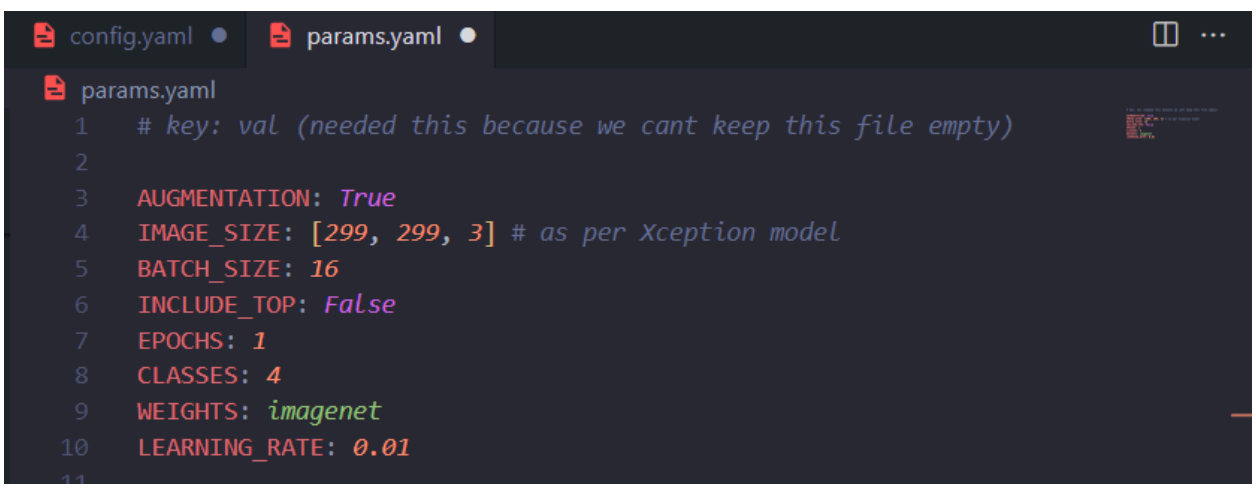
This setup ensures that the base model is properly configured, customized, and integrated into the machine learning pipeline, ready for further training and evaluation.

Params.yaml

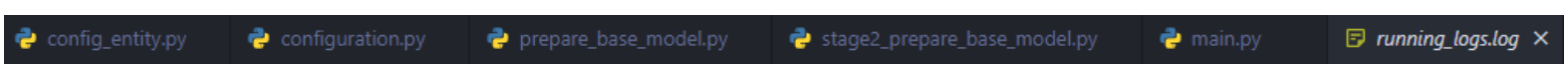
I will be using Xception model 🔗[Xception (keras.io)]

The input shape for a neural network model is determined by the requirements of the specific architecture and task. Let's break down the information you provided:

1. Include_top is True:
    - When include_top is True, it means that the fully connected layers (also known as the "top" layers) of the pre-trained model are included. These layers are typically used for classification tasks.
    - In this case, the input shape is expected to be `(299, 299, 3)` for models like Xception, etc.
2. include_top is False:
    - When include_top is False, it means that you are using the model for feature extraction or transfer learning. You'll add your own custom top layers (e.g., dense layers) for your specific task.
    - In this scenario, the input shape can vary, but it should have exactly 3 input channels (for RGB images), and the width and height should be no smaller than 71.
    - As mentioned on the Xception page, `(150, 150, 3)`, is indeed a valid value for the input shape when include_top is set to False.



All these files are updated after this



```python
STAGE_NAME = "Prepare base model"
try:
    logger.info(f"*******************")
    logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
    prepare_base_model = PrepareBaseModelTrainingPipeline()
    prepare_base_model.main()
    logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx=====
except Exception as e:
        logger.exception(e)
        raise e
```

Fig. final main.py run to implement all the changes and check if everything is running smoothly

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                Python

 block14_sepconv2_bn (BatchNorm  (None, 10, 10, 2048  8192      ['block14_sepconv2[0][0]']
 alization)                     )



 block14_sepconv2_act (Activati  (None, 10, 10, 2048  0         ['block14_sepconv2_bn[0][0]
']
 on)                            )

']

==================================================================================================
======
Total params: 21,680,684
Trainable params: 819,204
Non-trainable params: 20,861,480
__________________________________________________________________________________________________
______
[2024-06-02 18:48:14,016: INFO: main: >>>>>> stage Prepare base model completed <<<<<<

x==========x]
PS C:\Users\Sneha\Documents\Projects\KidneyNet>
```

Fig. main.py run after completion of Stage 2: Base Model creation

Stage 03: Model Training

Summary

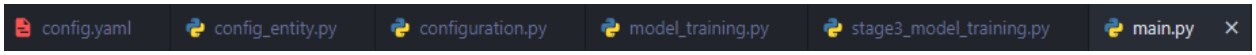


Fig. Flow of Stage 3

Model Training in my Codebase

After the base model preparation, the model training is managed by the Training class and orchestrated by the `ModelTrainingPipeline`. Here's how the model training is conducted:

1. Configuration Setup

The training configuration is defined in the `TrainingConfig` class, which includes parameters such as the path to the updated base model, training data directory, epochs, batch size, and augmentation settings.

2. Loading the Base Model

The Training class begins by loading the pre-trained and updated base model from the specified path. This model includes the custom top layers added during the base model preparation stage.

3. Data Preparation

The training and validation data generators are set up using `ImageDataGenerator` from TensorFlow. This handles data augmentation if it is enabled, and prepares data batches for training and validation.

4. Model Training

The model is trained using the fit method of the Keras API, where it processes the data in batches through the model, adjusting the model weights to minimize the loss.

5. Saving the Trained Model

After training, the model is saved to the specified path, making it available for later use in evaluation or deployment.

6. Pipeline Execution

The training process is executed within the `ModelTrainingPipeline` class, which sets up the configuration, initializes the Training class, and runs the training process.

This structured approach ensures that the model is trained effectively, leveraging the pre-trained base model and custom configurations, and is ready for evaluation and deployment.

Stage 04: Model Evaluation - MLflow

Summary

Model Evaluation in my Codebase

After model training, the evaluation of the trained model is conducted using the Evaluation class, orchestrated by the `EvaluationPipeline`. Here's how the model evaluation is performed:

1. Configuration Setup

The evaluation configuration is defined in the `EvaluationConfig` class, which includes parameters such as the path to the trained model, data for evaluation, and MLflow settings for logging the evaluation results.

2. Loading the Trained Model

The trained model is loaded from the specified path using the `load_model` method. This model is then used to perform the evaluation on the validation dataset.

3. Data Preparation for Evaluation

The validation data generator is set up to process the images from the validation dataset, preparing them in the format expected by the model.

4. Model Evaluation

The model's performance is evaluated using the evaluate method, which computes the loss and accuracy metrics on the validation data.

5. Saving and Logging Evaluation Results

The evaluation scores are saved to a JSON file and also logged to MLflow. MLflow tracks the parameters, metrics, and model, facilitating comparison across different runs.

6. Pipeline Execution

The evaluation process is executed within the `EvaluationPipeline` class, which sets up the configuration, initializes the `Evaluation` class, and runs the evaluation process.

This structured approach ensures that the model's performance is rigorously evaluated and the results are systematically logged and tracked using MLflow, providing a robust framework for model validation and experimentation tracking.

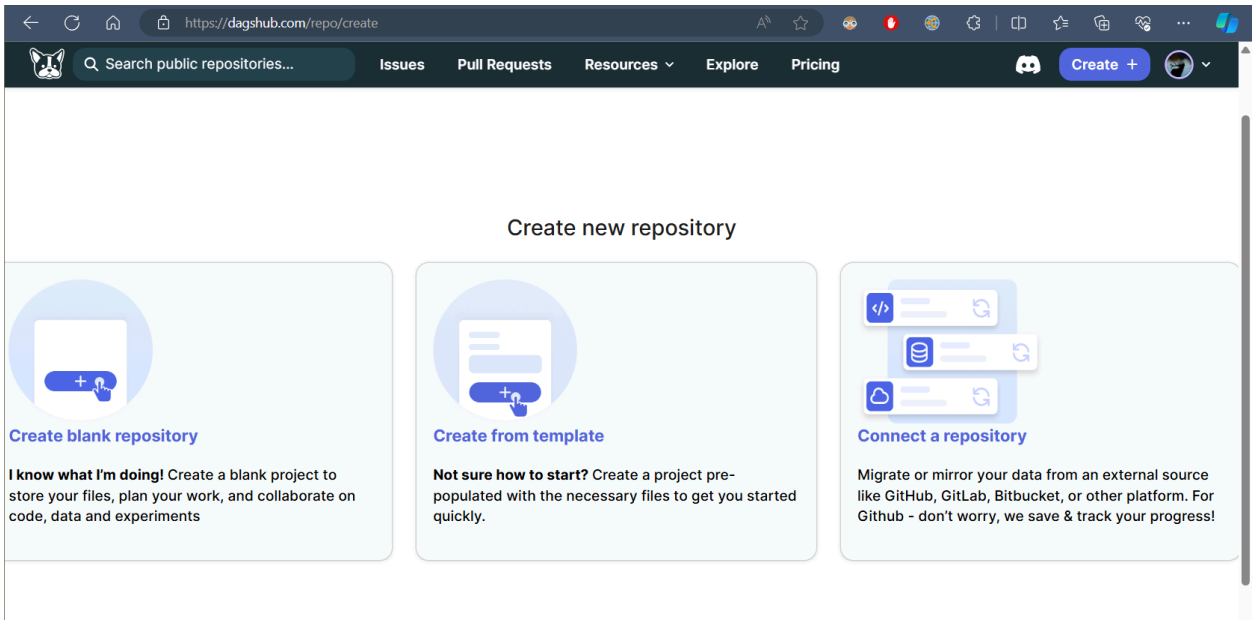


Fig. Creating of Dagshub account and connecting kidneyNet github repo

Fig. Stage 4 Model Evaluation File update flow

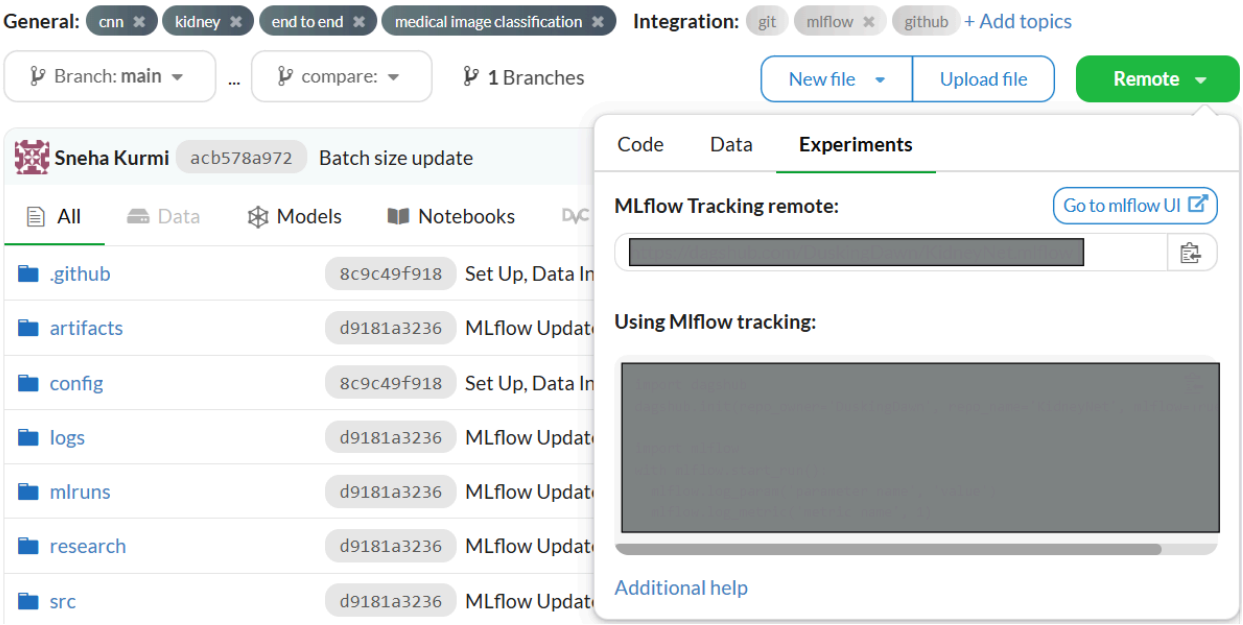


Fig. There MLflow tracking credentials are required for enabling tracking for your project model

```
Sneha@DESKTOP-U7KR60N MINGW64 ~/Documents/Projects/KidneyNet (main)
$ export export MLFLOW_TRACKING_USERNAME=9a0edab91919b156231f8a279dfdad
6e030
```

Fig. Dagshub (P.S. as you can see I made the mistake of writing export twice; dont that) need to run this again after anytime you close your environment

🔗 [DagsHub Integration with MLflow - Free remote MLflow server - DagsHub Docs]

❓ Why are we using Dagshub instead of just using MLflow?

```
src > cnnClassifier > entity > config_entity.py > ...

@dataclass(frozen=True)
class EvaluationConfig:
    path_of_model: Path
    training_data: Path
    all_params: dict
    mlflow_uri: str
    params_image_size: list
    params_batch_size: int
```

Fig. Updating config_entity file

```
config_entity.py M    configuration.py M    model_evaluation_

src > cnnClassifier > config > configuration.py > ConfigurationManager > get_training_conf

class ConfigurationManager:

    def get_evaluation_config(self) -> EvaluationConfig:
        eval_config = EvaluationConfig(
            path_of_model="artifacts/training/model.h5",
            training_data="artifacts/data_ingestion/kidney-ct
            mlflow_uri="https://dagshub.com/DuskingDawn/Kidne
            all_params=self.params,
            params_image_size=self.params.IMAGE_SIZE,
            params_batch_size=self.params.BATCH_SIZE
        )
        return eval_config
```

Fig. config → configuration.py update for evaluation

src > cnnClassifier > components > model_evaluation_mlflow.py > Evaluation > _valid_gener

```python
import tensorflow as tf
from pathlib import Path
import mlflow
import mlflow.keras
from urllib.parse import urlparse
from cnnClassifier.entity.config_entity import EvaluationConf
from cnnClassifier.utils.file_op_utils import read_yaml, crea


class Evaluation:
    def __init__(self, config: EvaluationConfig):
        self.config = config


    def _valid_generator(self):

        datagenerator_kwargs = dict(
            rescale = 1./255
```

Fig. model_evaluation_mlflow

src > cnnClassifier > pipeline > stage4_model_evaluation.py > EvaluationPipeline > main

```python
from cnnClassifier.config.configuration import ConfigurationM
from cnnClassifier.components.model_evaluation_mlflow import
from cnnClassifier import logger



STAGE_NAME = "Evaluation stage"


class EvaluationPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        eval_config = config.get_evaluation_config()
        evaluation = Evaluation(eval_config)
```

Fig. Pipeline Update

main.py > ...

```python


STAGE_NAME = "Evaluation stage"
try:
    logger.info(f"*******************")
    logger.info(f">>>>>> stage {STAGE_NAME} started <
    model_evalution = EvaluationPipeline()
    model_evalution.main()
    logger.info(f">>>>>> stage {STAGE_NAME} completed

except Exception as e:
        logger.exception(e)
        raise e
```

Fig. main.py update

```
Sneha@DESKTOP-U7KR60N MINGW64 ~/Documents/Projects/KidneyNet (main)
$ python main.py
[2024-06-03 22:16:03,568: INFO: main: ******************]
[2024-06-03 22:16:03,568: INFO: main: >>>>>> stage Training started <<<<<<]
[2024-06-03 22:16:03,568: INFO: file_op_utils: yaml file: config\config.yaml
 loaded successfully]
[2024-06-03 22:16:03,568: INFO: file_op_utils: yaml file: params.yaml loaded
 successfully]
[2024-06-03 22:16:03,568: INFO: file_op_utils: created directory at: artifac
ts]
[2024-06-03 22:16:03,568: INFO: file_op_utils: created directory at: artifac
ts\training]
Found 2489 images belonging to 1 classes.
Found 9957 images belonging to 1 classes.
622/622 [==============================] - 1698s 3s/step - loss: 103.2695 -
accuracy: 0.2487 - val_loss: 112.6569 - val_accuracy: 0.9851
[2024-06-03 22:44:23,973: INFO: main: >>>>>> stage Training completed <<<<<<

x=========x]
******]
[2024-06-03 22:44:23,973: INFO: main: >>>>>> stage Evaluation stage started
<<<<<<]
[2024-06-03 22:44:23,973: INFO: file_op_utils: yaml file: config\config.yaml
 loaded successfully]
[2024-06-03 22:44:23,985: INFO: file_op_utils: yaml file: params.yaml loaded
 successfully]
[2024-06-03 22:44:23,987: INFO: file_op_utils: created directory at: artifac
ts]
Found 3733 images belonging to 1 classes.
108/234 [============>.................] - ETA: 4:4
109/234 [============>.................] - ETA: 4:4
110/234 [=============>................] - ETA: 4:4
111/234 [=============>................] - ETA: 4:4
112/234 [=============>................] - ETA: 4:4113/234 [=============>..
...............] - ETA: 4:4114/234 [=============>................] - ETA: 4:
4115/234 [=============>................] - ETA: 4:4116/234 [=============>.
...............] - ETA: 4:3117/234 [==============>...............] - ETA: 4
:3118/234 [==============>...............] - ETA: 4:3119/234 [==============
```

Fig. Final run after all the updation

❗ Error

```
[2024-06-04 01:21:04,823: ERROR: main: The configured tracking uri scheme: 'file' is invalid for use with the proxy mlflow-artifact scheme. The allowed tracking schemes are: {'http', 'https'}]
Traceback (most recent call last):
  File "main.py", line 56, in <module>
    model_evalution.main()
  File "c:\users\sneha\documents\projects\kidneynet\src\cnnClassifier\pipeline\stage4_model_evaluation.py", line 20, in main
    evaluation.log_into_mlflow()
  File "c:\users\sneha\documents\projects\kidneynet\src\cnnClassifier\components\model_evaluation_mlflow.py", line 74, in log_into_mlflow
    mlflow.keras.log_model(self.model, "model")
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tensorflow\__init__.py", line 222, in log_model
    return Model.log(
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\models\model.py", line 552, in log
    mlflow.tracking.fluent.log_artifacts(local_path, mlflow_model.artifact_path)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\fluent.py", line 817, in log_artifacts
    MlflowClient().log_artifacts(run_id, local_dir, artifact_path)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\client.py", line 1069, in log_artifacts
    self._tracking_client.log_artifacts(run_id, local_dir, artifact_path)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\_tracking_service\client.py", line 448, in log_artifacts
    self._get_artifact_repo(run_id).log_artifacts(local_dir, artifact_path)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\_tracking_service\client.py", line 416, in _get_artifact_repo
    artifact_repo = get_artifact_repository(artifact_uri)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\artifact_repository_registry.py", line 106, in get_artifact_repository
    return _artifact_repository_registry.get_artifact_repository(artifact_uri)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\artifact_repository_registry.py", line 72, in get_artifact_repository
    return repository(artifact_uri)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\mlflow_artifacts_repo.py", line 45, in __init__
    super().__init__(self.resolve_uri(artifact_uri, get_tracking_uri()))
```

```
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\mlflow_artifact
s_repo.py", line 59, in resolve_uri
    _validate_uri_scheme(track_parse.scheme)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\mlflow_artifact
s_repo.py", line 35, in _validate_uri_scheme
    raise MlflowException(
mlflow.exceptions.MlflowException: The configured tracking uri scheme: 'file' is invalid for use
with the proxy mlflow-artifact scheme. The allowed tracking schemes are: {'http', 'https'}
Traceback (most recent call last):
  File "main.py", line 61, in <module>
    raise e
  File "main.py", line 56, in <module>
    model_evalution.main()
  File
"c:\users\sneha\documents\projects\kidneynet\src\cnnClassifier\pipeline\stage4_model_evalu
ation.py", line 20, in main
    evaluation.log_into_mlflow()
  File
"c:\users\sneha\documents\projects\kidneynet\src\cnnClassifier\components\model_evaluatio
n_mlflow.py", line 74, in log_into_mlflow
    mlflow.keras.log_model(self.model, "model")
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tensorflow\__init__.py", line
222, in log_model
    return Model.log(
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\models\model.py",
line 552, in log
    mlflow.tracking.fluent.log_artifacts(local_path, mlflow_model.artifact_path)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\fluent.py",
line 817, in log_artifacts
    MlflowClient().log_artifacts(run_id, local_dir, artifact_path)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\client.py",
line 1069, in log_artifacts
    self._tracking_client.log_artifacts(run_id, local_dir, artifact_path)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\_tracking_service\
client.py", line 448, in log_artifacts
    self._get_artifact_repo(run_id).log_artifacts(local_dir, artifact_path)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\tracking\_tracking_service\
client.py", line 416, in _get_artifact_repo
    artifact_repo = get_artifact_repository(artifact_uri)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\artifact_reposi
tory_registry.py", line 106, in get_artifact_repository
    return _artifact_repository_registry.get_artifact_repository(artifact_uri)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\artifact_reposi
```

```
tory_registry.py", line 72, in get_artifact_repository
    return repository(artifact_uri)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\mlflow_artifacts_repo.py", line 45, in __init__
    super().__init__(self.resolve_uri(artifact_uri, get_tracking_uri()))
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\mlflow_artifacts_repo.py", line 59, in resolve_uri
    _validate_uri_scheme(track_parse.scheme)
  File
"C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\mlflow\store\artifact\mlflow_artifacts_repo.py", line 35, in _validate_uri_scheme
    raise MlflowException(
mlflow.exceptions.MlflowException: The configured tracking uri scheme: 'file' is invalid for use with the proxy mlflow-artifact scheme. The allowed tracking schemes are: {'http', 'https'}
```

—----------

Created a mini dataset with same structure but just 16 samples in each class for testing propose

Honestly, I think the error problem was something else. But once I did the below the issuel got resolved.

```python
import dagshub
dagshub.init(repo_owner='DuskingDawn', repo_name='KidneyNet', mlflow=True)

import mlflow
with mlflow.start_run():
    mlflow.log_param('parameter name', 'value')
    mlflow.log_metric('metric name', 1)
```

Python

```
! ! ! AUTHORIZATION REQUIRED ! ! !

c:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\rich\live.py:229: UserWarning: ins
Jupyter support
  warnings.warn('install "ipywidgets" for Jupyter support')

Open the following link in your browser to authorize the client:
https://dagshub.com/login/oauth/authorize?state=77d68b50-b2a3-4bcf-94ee-48dbb07249bf&clien

Initialized MLflow to track repo "DuskingDawn/KidneyNet"

Repository DuskingDawn/KidneyNet initialized!
```

Fig. There was version conflict issue between httpx

```
(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>pip install httpx>=0.25.0
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
dagshub 0.3.27 requires httpx~=0.23.0, but you have httpx 0.27.0 which is incompatible.
```

```
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This
behaviour is the source of the following dependency conflicts.
jupyterlab 4.1.8 requires httpx>=0.25.0, but you have httpx 0.23.3 which is incompatible.
Successfully installed httpcore-0.16.3 httpx-0.23.3

(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>pip list | findstr httpx
httpx                     0.23.3
```

Fig. Dagshub required httpx version 0.23.0, had to uninstall the latest version and downloaded this compatible version

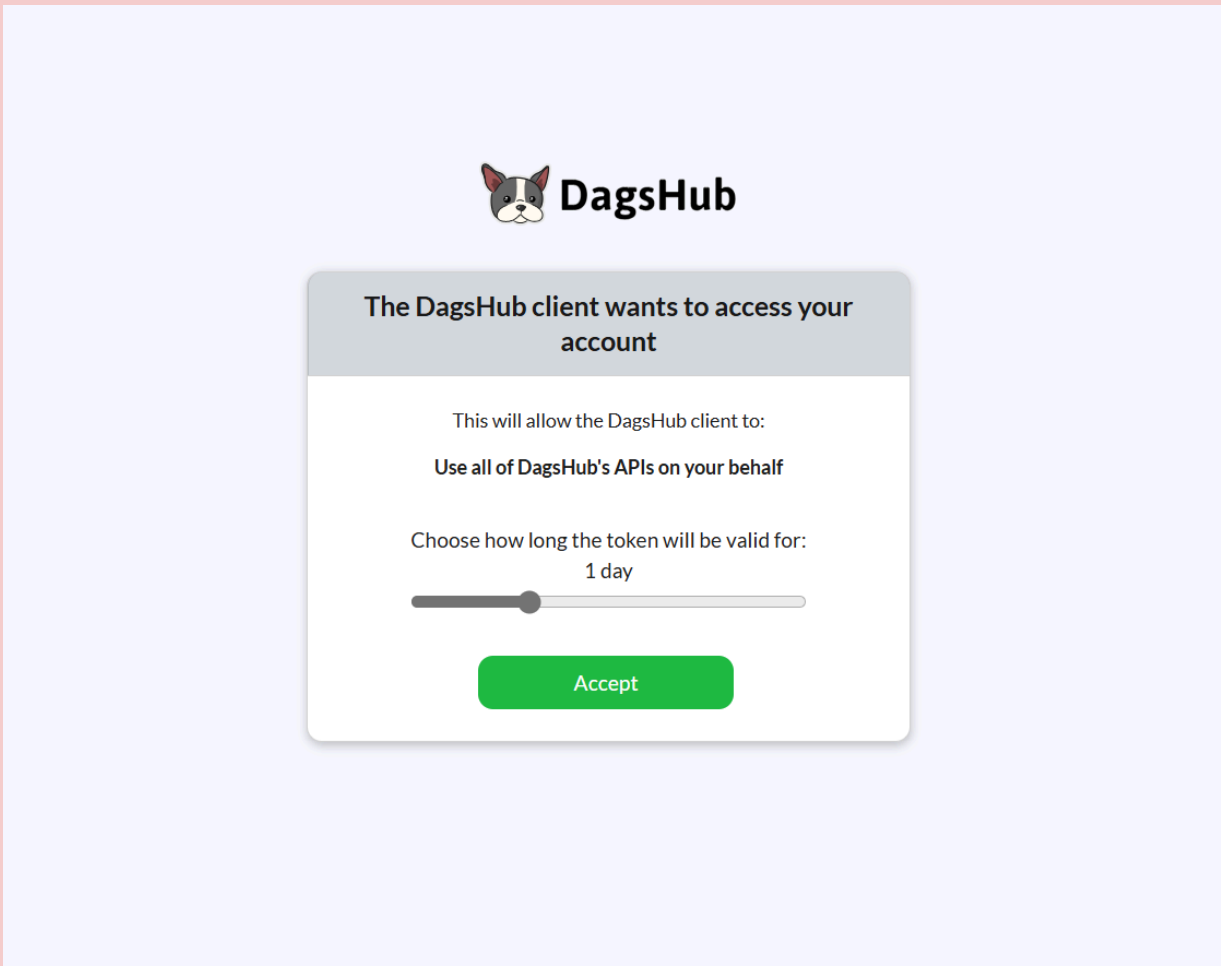


Fig. I gave the access for 1 day

```
ompiled_convolution_op while saving (showing 5 of 41). These functions will not be directly callable
 after loading.]
[2024-06-04 13:58:45,262: INFO: builder_impl: Assets written to: C:\Users\Sneha\AppData\Local\Temp\t
mpyuq49z0r\model\data\model\assets]
C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\_distutils_hack\__init__.py:33: UserWarning:
Setuptools is replacing distutils.
  warnings.warn("Setuptools is replacing distutils.")
[2024-06-04 13:58:56,505: INFO: dir_util: creating C:\Users\Sneha\Documents\Projects\KidneyNet\mlrun
s\0\9a0fb77a4b0242c29c778e1c5e656715\artifacts\model\data]
[2024-06-04 13:58:56,505: INFO: dir_util: creating C:\Users\Sneha\Documents\Projects\KidneyNet\mlrun
s\0\9a0fb77a4b0242c29c778e1c5e656715\artifacts\model\data\model]
[2024-06-04 13:58:56,505: INFO: dir_util: creating C:\Users\Sneha\Documents\Projects\KidneyNet\mlrun
s\0\9a0fb77a4b0242c29c778e1c5e656715\artifacts\model\data\model\assets]
[2024-06-04 13:58:56,523: INFO: dir_util: creating C:\Users\Sneha\Documents\Projects\KidneyNet\mlrun
s\0\9a0fb77a4b0242c29c778e1c5e656715\artifacts\model\data\model\variables]
[2024-06-04 13:58:56,652: INFO: main: >>>>>> stage Evaluation stage completed <<<<<<

x==========x]

Sneha@DESKTOP-U7KR60N MINGW64 ~/Documents/Projects/KidneyNet (main)
$ mlflow ui
INFO:waitress:Serving on http://127.0.0.1:5000
```

Fig. All the stages completed

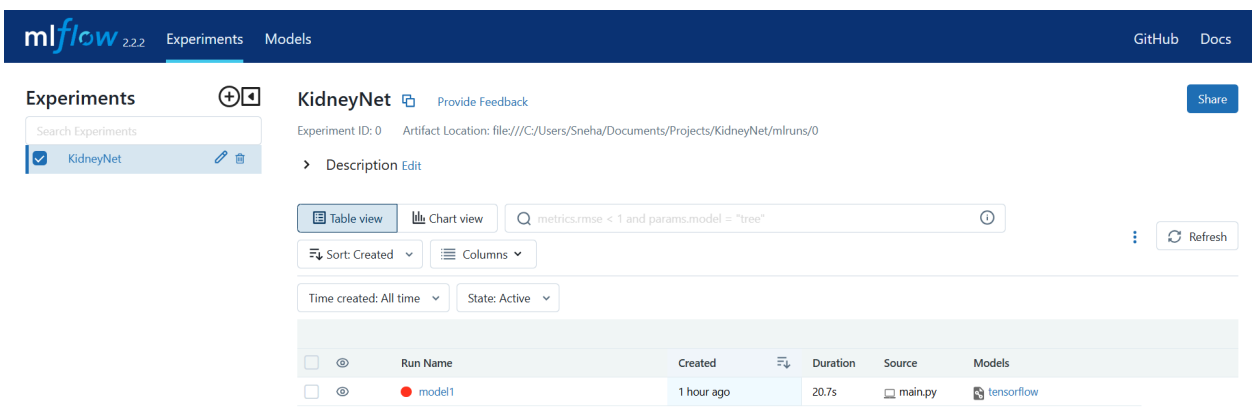


Fig. Model evaluations are successfully registered.

Pipeline Version Control - DVC

Summary

Pipeline Version Control with DVC in my Codebase

In your project, Data Version Control (DVC) is utilized to manage and version the machine learning pipelines. Here's how DVC is integrated and used:

1. DVC Configuration

The `.dvc/config` file sets up the core configuration for DVC, enabling features like autostage which automatically stages changes.

2. DVC Pipeline Definition

The `dvc.yaml` file defines the stages of your ML pipeline, specifying the commands to run, dependencies, and outputs for each stage. This includes stages for data ingestion, base model preparation, model training, and model evaluation.

3. DVC Ignore

The `.dvcignore` file is used to specify files or directories that DVC should ignore, which can improve performance and prevent unnecessary tracking.

4. Version Control and Reproducibility

DVC tracks the changes in data and code, allowing you to revert to any previous version of the pipeline or data. It ensures that every stage of your ML pipeline is reproducible by capturing the entire data transformation process.

5. Integration with Git

DVC works alongside Git. While Git tracks changes in the source code, DVC handles large data files, model weights, and pipeline stages. This integration ensures that both code and data are version controlled in a coordinated manner.

6. Running and Managing Pipelines

DVC commands like dvc repro are used to reproduce the pipeline from the `dvc.yaml` file. This command checks for changes in the dependencies and only re-runs the necessary stages, making the pipeline execution efficient.

This setup with DVC ensures that your machine learning pipeline is robust, version-controlled, and reproducible, aligning with best practices in MLOps.

Dvc library was install at the very start via requirement.txt

dvc.yaml M

dvc.yaml

```yaml
stages:
  data_ingestion:
    cmd: python src/cnnClassifier/pipeline/stage1_data_ingestion.py
    deps:
      - src/cnnClassifier/pipeline/stage1_data_ingestion.py
      - config/config.yaml
    outs:
      - artifacts/data_ingestion/CT-KIDNEY-DATASET-Normal-Cyst-Tumor-Stone


  prepare_base_model:
    cmd: python src/cnnClassifier/pipeline/stage2_prepare_base_model.py
    deps:
      - src/cnnClassifier/pipeline/stage2_prepare_base_model.py
      - config/config.yaml
    params:
      - IMAGE_SIZE
      - INCLUDE_TOP
      - CLASSES
      - WEIGHTS
      - LEARNING_RATE
    outs:
```

Fig. filling up dvc file for pipeline control

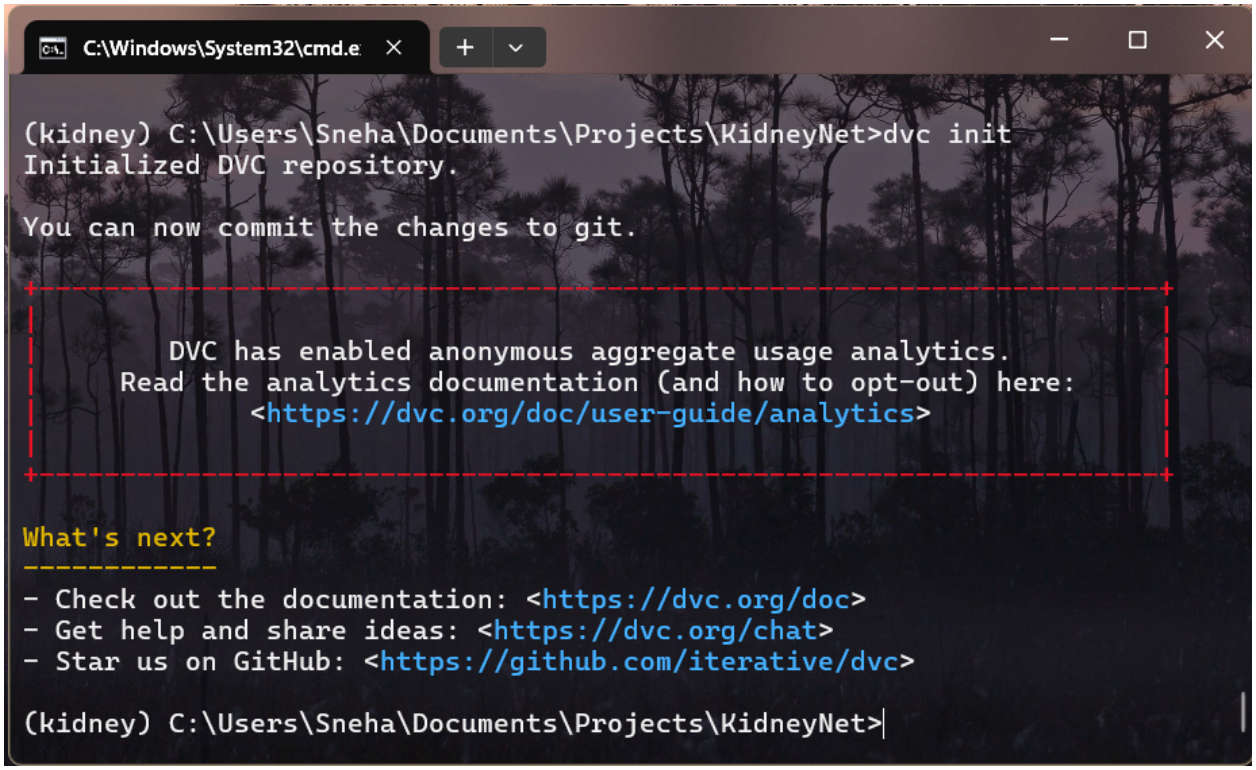


Fig. Initializing dvc using ‘dvc init’

Once you run ‘dvc init’ some new files would be created namely:

- .dvcignore
- .dvc → gitignore
- .dvc → config

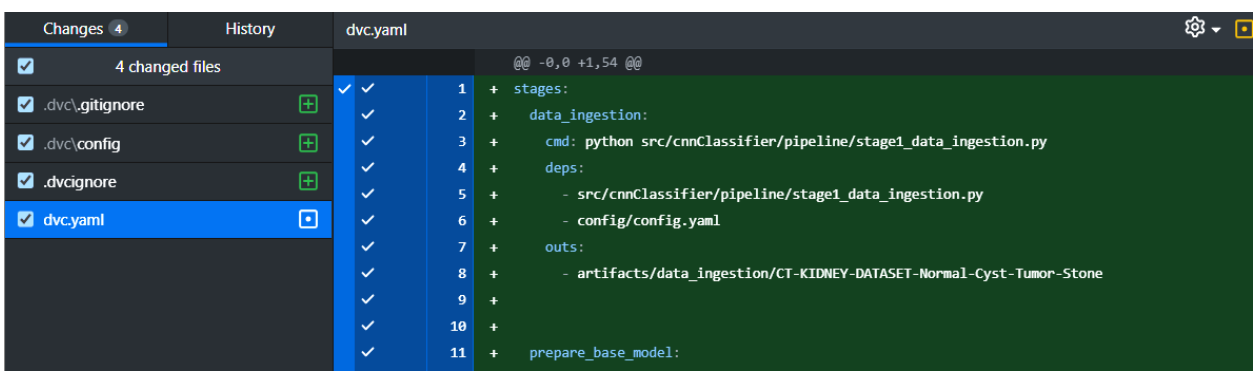


Fig. Committing these changes

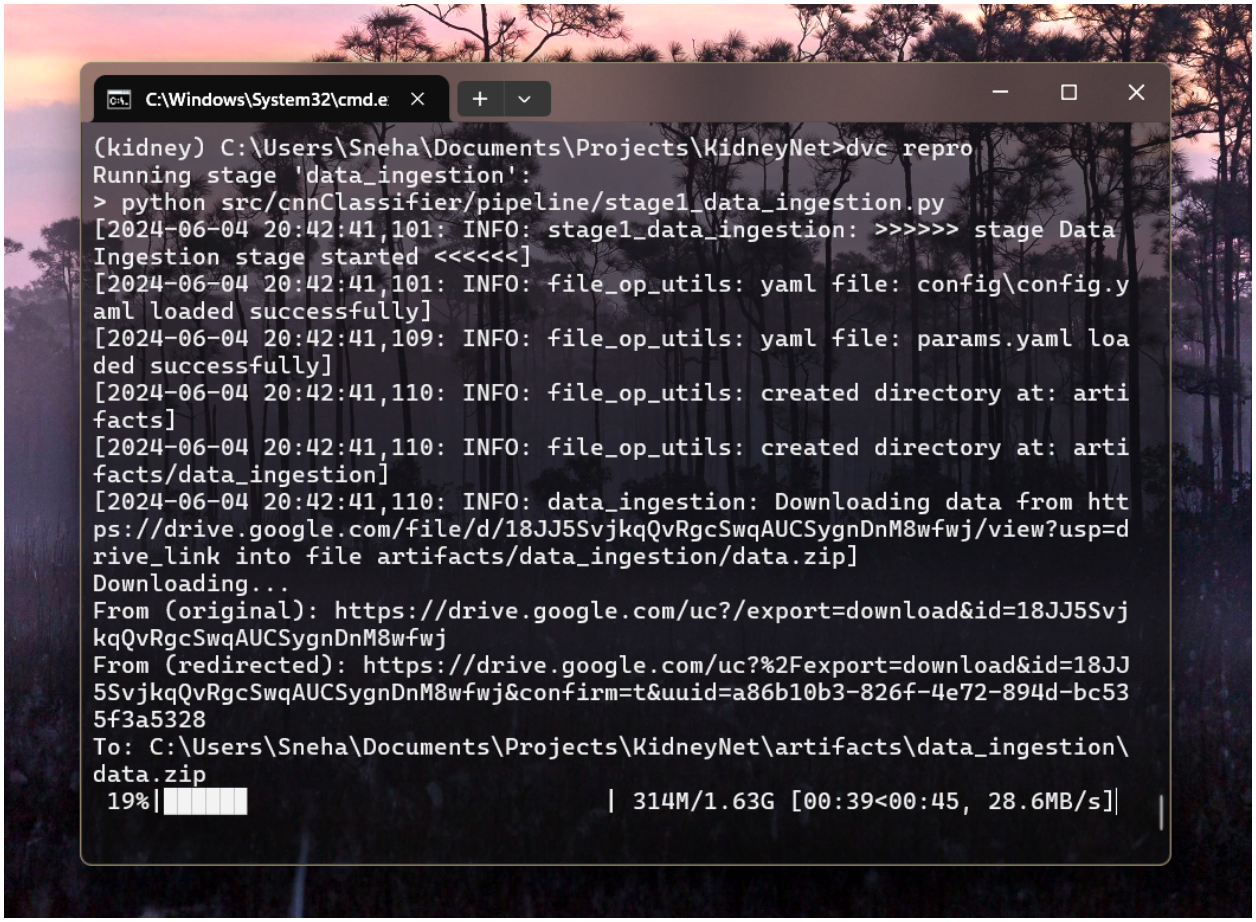


Fig. The first time you run after dvc initialization it will download data even tho data is present in data_ingestion folder

❗ ERROR

```
C:\Windows\System32\cmd.e

4_sepconv2_act[0][0]']


 dense (Dense)                (None, 4)                 819204      ['flatte
n[0][0]']


==================================================================================
==========================
Total params: 21,680,684
Trainable params: 819,204
Non-trainable params: 20,861,480
----------------------------------------------------------------------------------
--------------------------
[2024-06-04 21:03:32,807: INFO: stage2_prepare_base_model: >>>>>> stage P
repare base model completed <<<<<<

x==========x]
ERROR: failed to reproduce 'prepare_base_model':  output 'artifacts\prepa
re_base_model' is already tracked by SCM (e.g. Git).
    You can remove it from Git, then add to DVC.
        To stop tracking from Git:
            git rm -r --cached 'artifacts\prepare_base_model'
            git commit -m "stop tracking artifacts\prepare_base_model"

(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>
```

Solution:

```
Sneha@DESKTOP-U7KR60N MINGW64 ~/Documents/Projects/KidneyNet (main)
$ git rm -r --cached 'C:\Users\Sneha\Documents\Projects\KidneyNet\artifacts\prepare_
base_model'
rm 'artifacts/prepare_base_model/base_model.h5'
rm 'artifacts/prepare_base_model/base_model_updated.h5'

Sneha@DESKTOP-U7KR60N MINGW64 ~/Documents/Projects/KidneyNet (main)
$ git commit -m "stop tracking artifacts\prepare_base_model"
[main b4c23f6] stop tracking artifacts\prepare_base_model
 2 files changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 artifacts/prepare_base_model/base_model.h5
 delete mode 100644 artifacts/prepare_base_model/base_model_updated.h5
```

pwsh
bash

Fig. just do what is asked [add/run: improve "already tracked" message · Issue #3678 · iterative/dvc (github.com)]

```
C:\Windows\System32\cmd.e
[2024-06-04 22:38:21,215: INFO: dir_util: creating C:\Users\Sneha\Documen
ts\Projects\KidneyNet\mlruns\0\60b5542ceef141a7a4077f71eb74bcea\artifacts
\model\data\model]
[2024-06-04 22:38:21,216: INFO: dir_util: creating C:\Users\Sneha\Documen
ts\Projects\KidneyNet\mlruns\0\60b5542ceef141a7a4077f71eb74bcea\artifacts
\model\data\model\assets]
[2024-06-04 22:38:21,225: INFO: dir_util: creating C:\Users\Sneha\Documen
ts\Projects\KidneyNet\mlruns\0\60b5542ceef141a7a4077f71eb74bcea\artifacts
\model\data\model\variables]
[2024-06-04 22:38:21,507: INFO: stage4_model_evaluation: >>>>>> stage Eva
luation stage completed <<<<<<

x==========x]
Updating lock file 'dvc.lock'

To track the changes with git, run:

        git add dvc.lock 'artifacts\.gitignore' 'artifacts\training\.giti
gnore'

To enable auto staging, run:

        dvc config core.autostage true
Use `dvc push` to send your updates to remote storage.

(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>
```

autostage - if enabled, DVC will automatically stage (git add) DVC filesDVC files created or modified by DVC commands. The files will not be committed. Accepts values true and false (default). core. site_cache_dir - specify a custom location for misc temporary files.

```
C:\Windows\System32\cmd.e
  --show-origin  Show the source file containing each config value.

(kidney) C:\Users\Sneha\Documents\Projects\KidneyNet>dvc config core.auto
stage true
```

❓ Do I have to ‘dvc push’ while using DagsHub?

DAGsHub is already integrated with DVC, in the sense that whenever you create a project, it comes with a free, built-in, DVC remote. Since last week, you also get a free MLflow server, which means you can log experiments directly to DAGsHub and share it with your team or colleagues. 26 Apr 2021

Reddit
https://www.reddit.com › comments › myyctr › p_integ...

[P] Integrating Git, DVC, and MLflow into one - Reddit

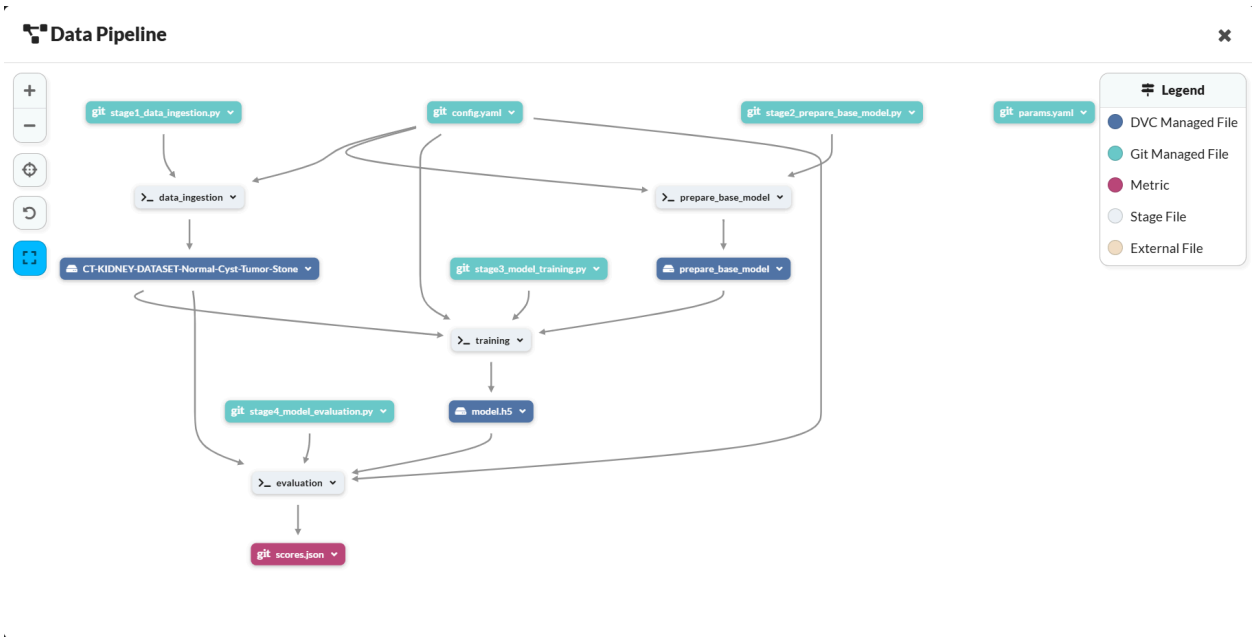


Fig. Dags Data Pipeline Visualization after DVC connection

Frontend

Summary

Frontend Code Handling in Your Codebase

In your codebase, the frontend code is primarily managed through HTML and JavaScript, with the index.html file located in the templates folder serving as the main entry point. Here's how the frontend is structured and handled:

1. HTML Structure

The `index.html` file provides the HTML structure for the user interface. It includes sections for uploading images, displaying prediction results, and other interactive elements.

2. Styling

CSS is used for styling the webpage, with styles defined directly within the `index.html` file under <style> tags, and external Bootstrap CSS is included for responsive design.

3. JavaScript and Interactivity

JavaScript is used to handle user interactions, such as file uploads and sending requests to the backend for image processing and prediction. The scripts are included at the bottom of the HTML file, and jQuery is used to simplify DOM manipulation and AJAX requests.

4. Dynamic Content Loading

The frontend dynamically updates content, such as displaying prediction results, without needing to reload the page. This is achieved using AJAX to fetch data from the server and then updating the DOM based on the response.

5. Deployment Considerations

The HTML file is designed to be served by a web server, and it interacts with the backend via API calls. Ensure that the server is properly configured to serve the index.html file as the default page and handle static assets and API requests correctly.

This setup allows for a user-friendly interface that interacts seamlessly with the backend, providing functionalities like image upload and displaying machine learning model predictions.

---

src > cnnClassifier > pipeline > prediction.py > PredictionPipeline > predict

```python
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import os

class PredictionPipeline:
    def __init__(self, filename):
        self.filename = filename

    def predict(self):
        # Load model
        model = load_model(os.path.join("model", "model.h5"))

        # Load and preprocess the image
        imagename = self.filename
        test_image = image.load_img(imagename, target_size=(224, 224))
        test_image = image.img_to_array(test_image)
        test_image = np.expand_dims(test_image, axis=0)

        # Make prediction
```

Fig. Prediction code

app.py > ...

```python

@app.route("/train", methods=['GET','POST'])
@cross_origin()
def trainRoute():
    os.system("python main.py")
    # os.system("dvc repro")
    return "Training done successfully!"
```

Fig. Keep this part in mind

```python
from flask import Flask, request, jsonify, render_template
import os
from flask_cors import CORS, cross_origin
from cnnClassifier.utils.file_op_utils import decodeImage
from cnnClassifier.pipeline.prediction import PredictionPipeline



os.putenv('LANG', 'en_US.UTF-8')
os.putenv('LC_ALL', 'en_US.UTF-8')

app = Flask(__name__)
CORS(app)


class ClientApp:
    def __init__(self):
        self.filename = "inputImage.jpg"
        self.classifier = PredictionPipeline(self.filename)


@app.route("/", methods=['GET'])
@cross_origin()
def home():
    return render_template('index.html')
```

Fig. created a new file called app.py

In your Flask application, the encoding and decoding of images are used to facilitate the transfer of image data between the client (e.g., a web browser or a mobile app) and the server. This process ensures that the image can be sent as a string in a JSON object, which is a common data format for web communication. Let's break down the process:

1. **Encoding the Image (Client-side)**:
    - The image is converted to a base64-encoded string. This is typically done in the client-side code (e.g., in JavaScript). Base64 encoding allows binary data (such as an image) to be represented as a string of ASCII characters.
    - The base64-encoded string can be easily included in a JSON object and sent to the server via an HTTP request.
2. **Decoding the Image (Server-side)**:
    - The server receives the JSON object containing the base64-encoded image string.
    - The `decodeImage` function is used to convert the base64-encoded string back into the original image format. This decoded image is then saved to the file system or processed further.

Here's how this fits into your application:

Encoding (Client-side)

When a user uploads an image, the client-side code (e.g., JavaScript in the browser) converts the image to a base64-encoded string and sends it to the server as part of a JSON object.

Decoding (Server-side)

On the server side, the `decodeImage` function handles the conversion of the base64 string back to an image file, which can then be processed by the `PredictionPipeline` for classification.

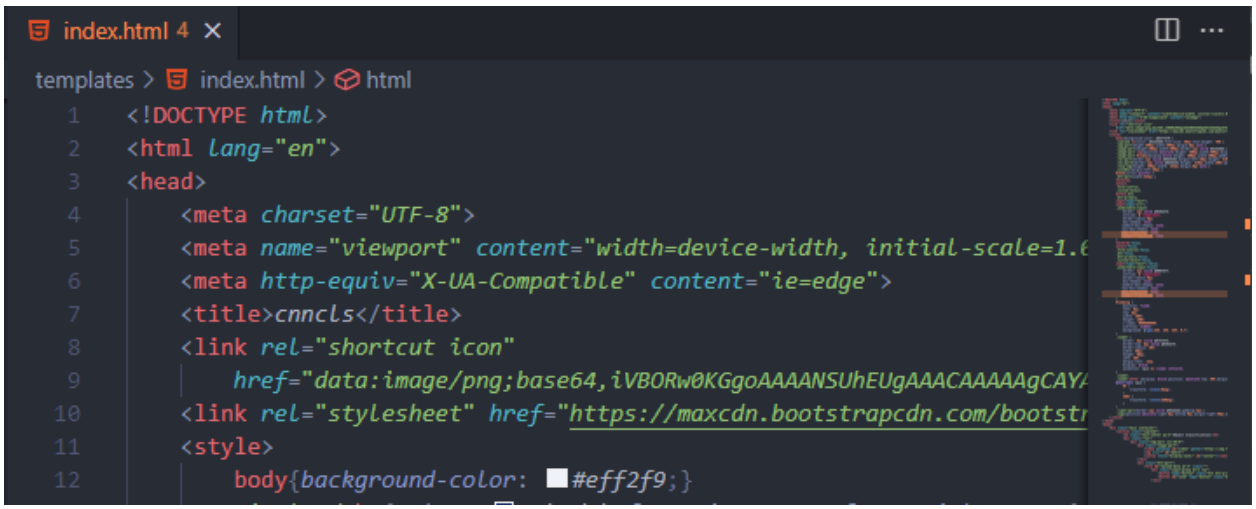


Fig. html code for frontend. (I hate this part, freaking bootstrap)

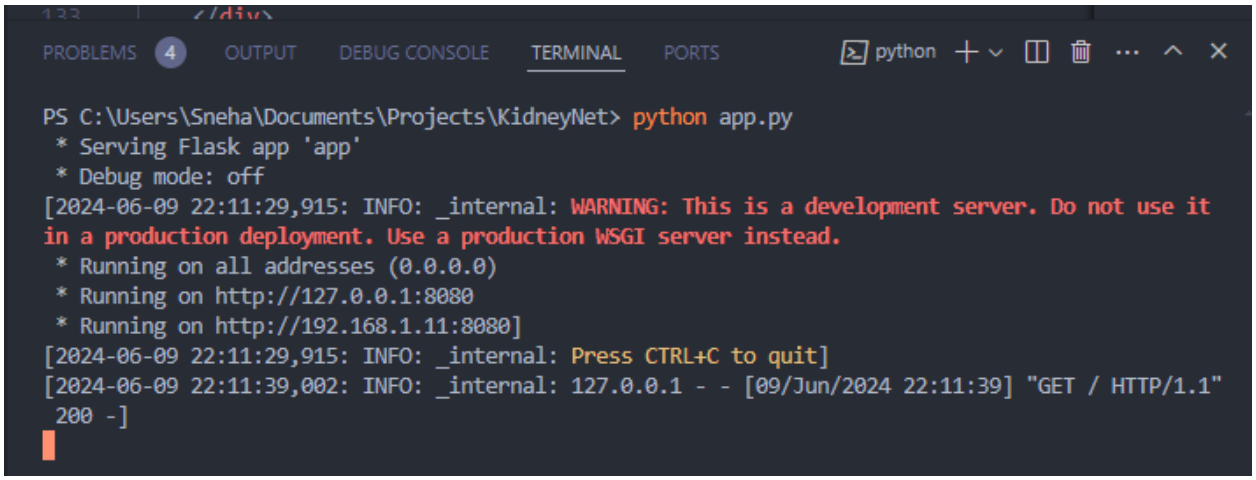


Fig. Run the code

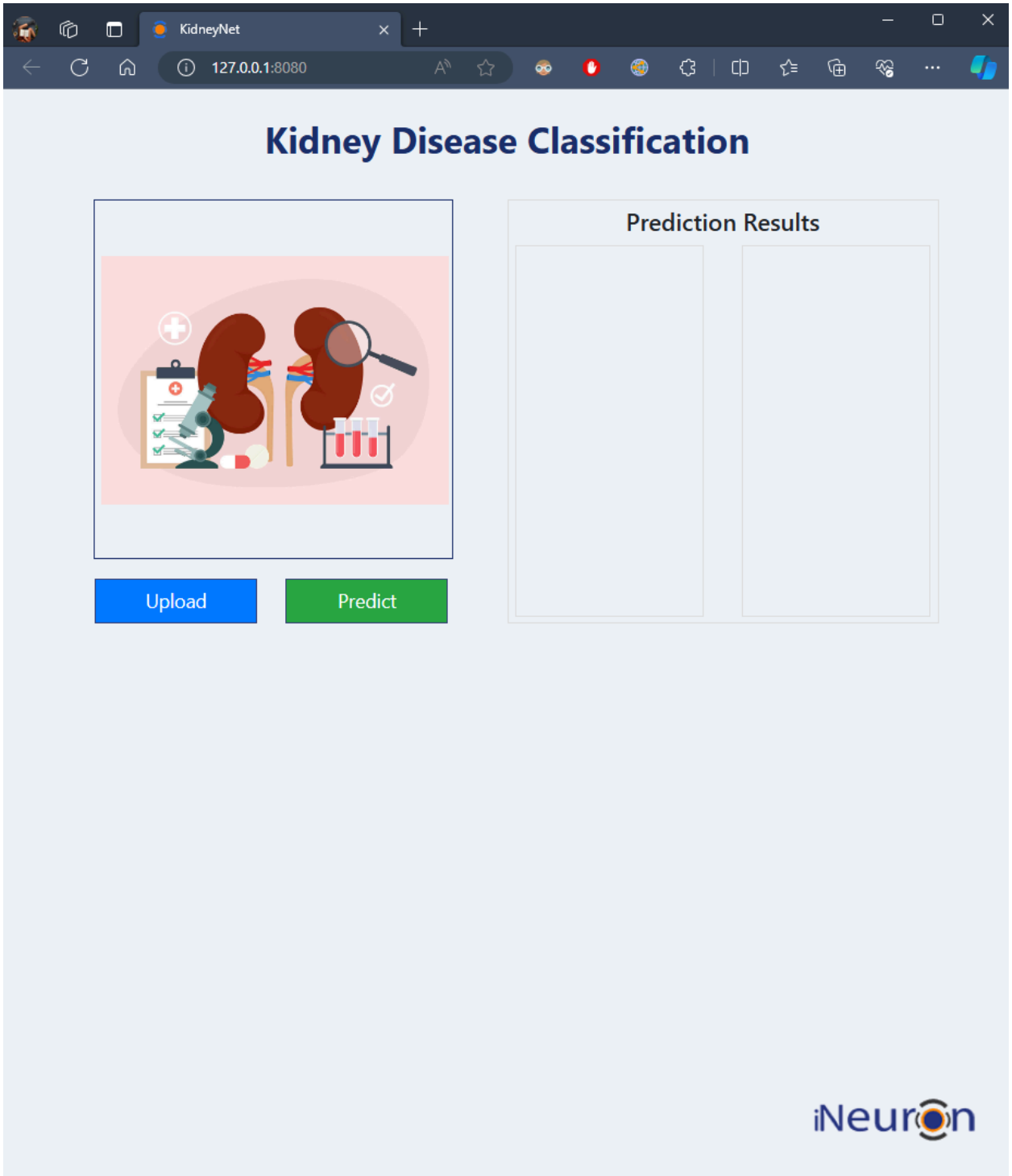


Fig. Voilà!

OK wait. Another error.

❗ ERROR

```
PS C:\Users\Sneha\Documents\Projects\KidneyNet> python app.py
 * Serving Flask app 'app'
 * Debug mode: off
[2024-06-09 22:15:51,082: INFO: _internal: WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://192.168.1.11:8080]
[2024-06-09 22:15:51,082: INFO: _internal: Press CTRL+C to quit]
[2024-06-09 22:15:54,653: INFO: _internal: 127.0.0.1 - - [09/Jun/2024 22:15:54] "GET / HTTP/1.1" 200 -]
[2024-06-09 22:16:36,866: ERROR: app: Exception on /predict [POST]]
Traceback (most recent call last):
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\flask\app.py", line 2190, in wsgi_app
    response = self.full_dispatch_request()
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\flask\app.py", line 1486, in full_dispatch_request
    rv = self.handle_user_exception(e)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\flask_cors\extension.py", line 176, in wrapped_function
    return cors_after_request(app.make_response(f(*args, **kwargs)))
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\flask\app.py", line 1484, in full_dispatch_request
    rv = self.dispatch_request()
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\flask\app.py", line 1469, in dispatch_request
    return self.ensure_sync(self.view_functions[rule.endpoint])(**view_args)
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\flask_cors\decorator.py", line 130, in wrapped_function
    resp = make_response(f(*args, **kwargs))
  File "app.py", line 44, in predictRoute
    result = clApp.classifier.predict()
  File "c:\users\sneha\documents\projects\kidneynet\src\cnnClassifier\pipeline\prediction.py", line 12, in predict
    model = load_model(os.path.join("model", "model.h5"))
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\keras\saving\saving_api.py", line 212, in load_model
    return legacy_sm_saving_lib.load_model(
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\keras\utils\traceback_utils.py", line 70, in error_handler
    raise e.with_traceback(filtered_tb) from None
  File "C:\Users\Sneha\anaconda3\envs\kidney\lib\site-packages\keras\saving\legacy\save.py", line 230, in load_model
    raise IOError(
OSError: No file or directory found at model\model.h5
```

[2024-06-09 22:16:36,878: INFO: _internal: 127.0.0.1 - - [09/Jun/2024 22:16:36] "POST /predict HTTP/1.1" 500 -]

SOLUTION:
Change the file location

```
def predict(self):
    # Load model
    model = load_model(os.path.join("artifacts", "training", "model.h5"))
```

Change the target image size according to my model

```
# Load and preprocess the image
imagename = self.filename
test_image = image.load_img(imagename, target_size=(299, 299))
test_image = image.img_to_array(test_image)
test_image = np.expand_dims(test_image, axis=0)
```

I almost broke a sweat thinking I need to train the while model again because of some mismatch.

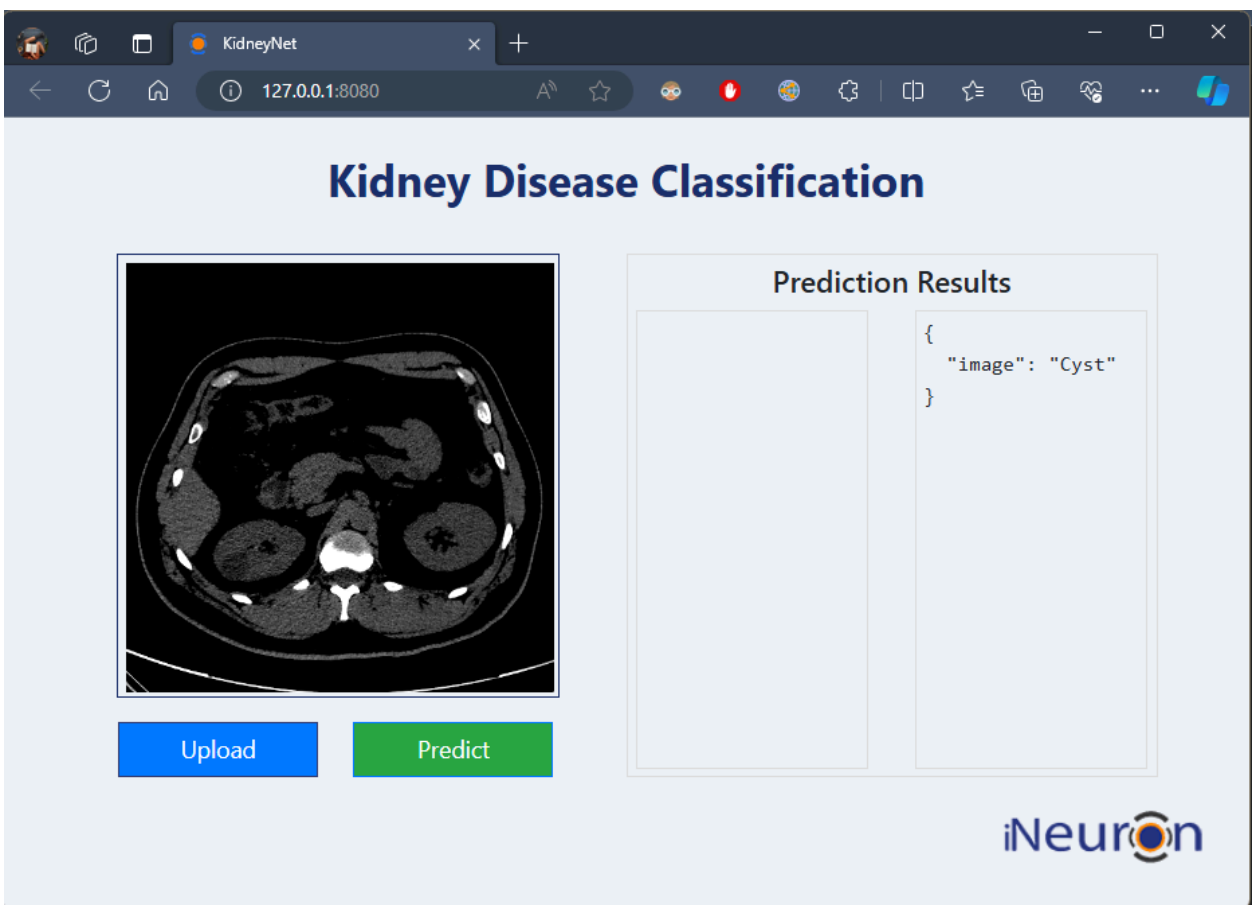


Fig. After two error corrections here is the result

Docker

Summary

Key Functions of the Dockerfile

1. Base Image: It starts from a Python 3.8 slim image, which is a lightweight version of the Python environment.

2. Working Directory: `Sets /usr/src/app` as the working directory where your application code will reside within the container.

3. Copying Application Code: Transfers your local application code into the Docker container.

4. Dependencies Installation: Installs the Python packages required by your application as specified in `requirements.txt`.

5. Port Exposure: Makes `port 80` available outside the container, allowing network interactions with your application.

6. Environment Variable: Sets an environment variable `NAME` with the value World, which can be used by your application for configuration.

7. Default Command: Specifies that the container should run `main.py` using Python when it starts.

Interaction with Other Components

Application Code: The Dockerfile directly interacts with your application code by copying it into the container and running it.

Dependencies: It ensures all dependencies listed in requirements.txt are installed in the container, isolating your application's environment from external changes.

Networking: By exposing port 80, it allows the application to communicate with external services or users, crucial for web applications.

Environment Configuration: The environment variable set in the Dockerfile can be accessed by your application code to modify behavior based on the deployment environment.

---

dockerfile U

dockerfile

```
# Use an official Python runtime as a parent image
FROM python:3.8-slim

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy the current directory contents into the container at /usr/src/ap
COPY . .

RUN pip install --no-cache-dir -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

ENV NAME World

CMD ["python", "main.py"]

Ctrl+L to chat, Ctrl+K to generate
```

❓ What does EXPOSE 80 do? and is it dangerous from a security point of view?

The EXPOSE 80 directive in a Dockerfile is used to inform Docker that the container listens on the specified network port(s) at runtime. In this case, EXPOSE 80 indicates that the container will listen on port 80.

What EXPOSE Does:

Documentation Purpose: It acts as a type of documentation between the person who builds the image and the person who runs the container, about which ports are intended to be published. Inter-container Communication: It helps facilitate communication between containers. When using Docker's default bridge network, other containers on the same network can communicate with this container on the exposed port without needing to publish the port on the host.

Security Implications:

Not Inherently Dangerous: By itself, EXPOSE does not make a container insecure. It does not open the port on the host machine; it merely indicates that the container will use this port.

Port Publishing: The security risk primarily arises when you publish the port to the outside world with -p or --publish in the docker run command, like docker run -p 80:80 yourimage. This maps port 80 of the container to port 80 on the host, making the application accessible via the host's IP address on port 80.
Potential Risks: If the application running inside the container has vulnerabilities (e.g., it serves unsecured or sensitive content, has bugs or exploits like SQL injection, buffer overflows, etc.), exposing it to the internet can indeed be dangerous.

Best Practices:

Minimize Exposure: Only expose and publish the ports that are necessary for your application.

Security Measures: Implement security best practices in your application code. Use firewalls, HTTPS, authentication, and other security measures to protect your application.

Regular Updates: Keep your application and its dependencies up-to-date with security patches.

In summary, EXPOSE 80 itself is not dangerous, but how you manage the exposure of your application to the network can introduce security risks. Always consider the necessity and security implications of exposing certain ports when designing and deploying your Docker containers.

'ENV NAME World' I understand that it is optional but I want to know why is it used in general scenario and what is industry practises?

The ENV instruction in a Dockerfile is used to set environment variables in the container. These variables are accessible to processes running inside the container. Here's why and how it's typically used:

Uses of ENV:

1. Configuration: Environment variables are a common way to configure settings for software applications. This can include database URLs, API keys, feature flags, and other operational configurations that might change between environments (development, staging, production).

2. Default Values: Setting default environment variables can provide defaults for your application that can be overridden with custom values at runtime if needed.

3. Reduce Hardcoding: Using environment variables helps avoid hardcoding sensitive information and configuration settings directly in the code or Dockerfile, making the application more secure and flexible.

4. Interoperability: They allow for easier interoperability between different services and components that might need common configuration items.

Industry Practices:

Security: Sensitive data like passwords, tokens, and secrets should ideally not be stored directly in Dockerfiles or image layers. Instead, they should be injected at runtime using secure mechanisms like Docker secrets, Kubernetes secrets, or environment variables set through CI/CD pipelines without exposing them in the codebase or version control.

Parameterization: Environment variables are used to parameterize containers so that the same Docker image can be used across different environments or configurations by simply changing the environment variables.

Documentation: Using ENV also serves as documentation, indicating what configurations or variables are expected and potentially what their default values are.
Immutability and Scalability: By using environment variables, the same container image can remain immutable while being scalable across different environments and instances, adhering to the principles of twelve-factor applications.

Example:

In a web application, you might see something like:

```
ENV DATABASE_URL postgres://user:password@hostname:port/dbname
```

This sets a default database URL, but it can be overridden when running the container:

```
ATABASE_URL=postgres://otheruser:otherpassword@otherhost:port/otherdb myapp
```

This flexibility is crucial for deploying the same application in different environments or with different configurations without needing to rebuild the image.

In summary, ENV is used to set up and configure the environment for applications running in Docker containers in a secure, flexible, and scalable manner. It adheres to best practices by avoiding hardcoding, allowing easy configuration changes, and supporting security through external configuration management.