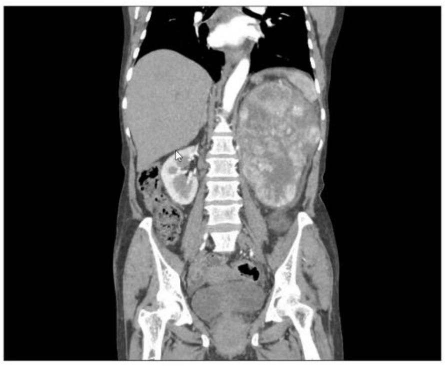
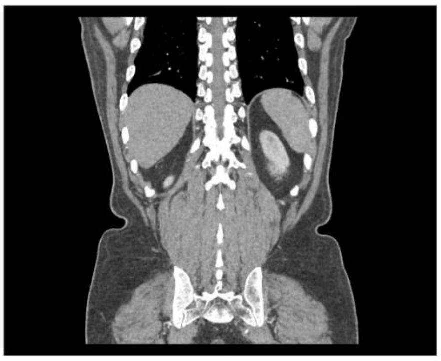
Complete End to End Deep Learning Project With MLFLOW, DVC And AWS Deployment- (***kidney disease classification***)

1. Introduction & GitHub Repository Setup
2. Project Template Creation
3. Project Setup & Requirements Installation
4. Logging, Utils & Exception Module
5. Project Workflows
6. All Components Notebook Experiment
7. All Components Modular Code Implementation
8. Training Pipeline
9. MLflow (MLOps Tool) - For Experiments tracking & Model Registration
10. DVC (MLOps Tool) - For Pipeline Tracking & implementation
11. Prediction Pipeline & User App Creation
12. Docker
13. Final CI/CD Deployment on AWS

Kidney-CT-Scan-Image


Tumor


Normal

What is Kidney tumors ?

Kidney tumors (also called renal tumors) are growths in the kidneys that can be benign or cancerous. Most do not cause symptoms and are discovered unexpectedly when you are being diagnosed and treated for another condition.

Step 1: Introduction & GitHub Repository Setup

- Create git repository Nikhil2893/Kidney_Disease_Classification_Deep_Learning_Project
- Add **README.md** file
- Select **.gitignore Python** template
- Git repository created.
- > git clone Nikhil2893/Kidney_Disease_Classification_Deep_Learning_Project
- Enter inside folder and open VS code

Problem Statement: Write an algorithm to classify whether an image contain either ***normal or tumor?***

Dataset: https://www.kaggle.com/c/dogs-vs-cats/data

Dataset I am using is mini version of above dataset.

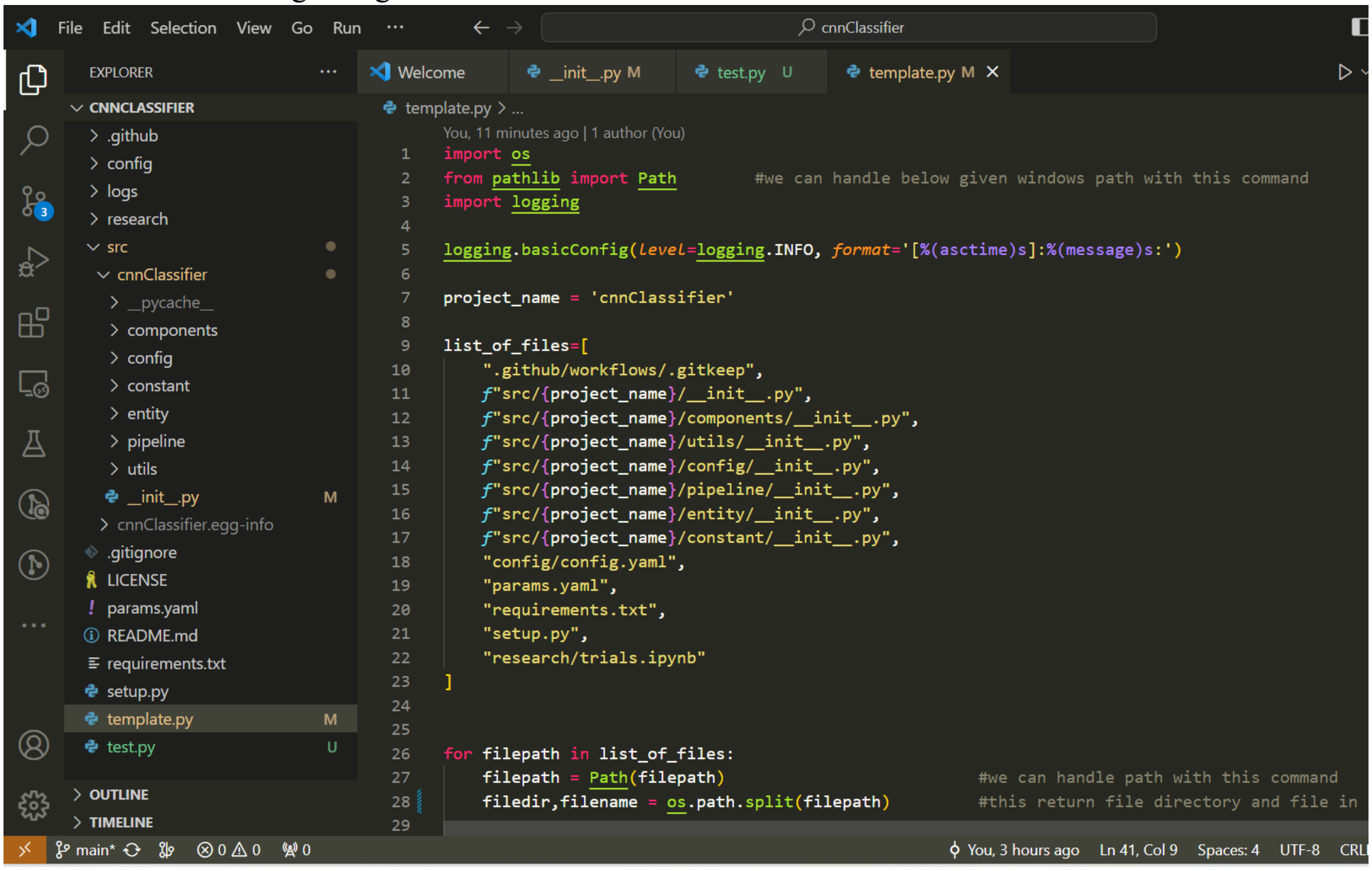
Step 2: Project Template creation

- Create new file "**template.py**" in 'cnnClassifier' folder.
- **template.py**

```
import os
from pathlib import Path   #we can handle below given windows path with this command

import logging

#logging string
logging.basicConfig(level=logging.INFO, format='[%(asctime)s]: %(message)s:')
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}/utils/__init__.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}/entity/__init__.py",
    f"src/{project_name}/constants/__init__.py",
    "config/config.yaml",
    "dvc.yaml",
    "params.yaml",
    "requirements.txt",
    "setup.py",
    "research/trials.ipynb",
    "templates/index.html"

]
```

```
for filepath in list_of_files:
    filepath = Path(filepath)
    filedir, filename = os.path.split(filepath)   #this return file directory 
and file in form of tuple-->>()
    if filedir !="":
        os.makedirs(filedir, exist_ok=True)
        logging.info(f"Creating directory; {filedir} for the file: {filename}")

    if (not os.path.exists(filepath)) or (os.path.getsize(filepath) == 0):
        with open(filepath, "w") as f:
            pass
            logging.info(f"Creating empty file: {filepath}")

    else:
        logging.info(f"{filename} is already exists")
```

- Run > **python template.py**
- Above project folder structure will be created.
- Commit changes in github



```
import os
from pathlib import Path        #we can handle below given windows path with this command
import logging

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

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}/utils/__init__.py",
    f"src/{project_name}/config/__init__.py",
    f"src/{project_name}/pipeline/__init__.py",
    f"src/{project_name}/entity/__init__.py",
    f"src/{project_name}/constant/__init__.py",
    "config/config.yaml",
    "params.yaml",
    "requirements.txt",
    "setup.py",
    "research/trials.ipynb"
]


for filepath in list_of_files:
    filepath = Path(filepath)                          #we can handle path with this command
    filedir,filename = os.path.split(filepath)         #this return file directory and file in

```

Step 3: Project Setup and Requirements installation

3.1 Install requirements.txt

```
tensorflow==2.12.0
pandas
dvc
mlflow==2.2.2
notebook
numpy
matplotlib
seaborn
python-box==6.0.2
pyYAML
tqdm
ensure==1.0.2
joblib
types-PyYAML
scipy
Flask
Flask-Cors
gdown
-e .
```

3.2 setup.py & requirements installations

Application will be created as a package

```python
import setuptools

with open("README.md", "r", encoding="utf-8") as f:
    long_description = f.read()


# Those things will be required when you publish project as PiPY package

__version__ = "0.0.0"
REPO_NAME = "Kidney_Disease_Classification_Deep_Learning_Project"
AUTHOR_USER_NAME = "Nikhil2893"
SRC_REPO = "cnnClassifier"
AUTHOR_EMAIL = "nikhilwakode2893@gmail.com"



setuptools.setup(
    name=SRC_REPO,
    version=__version__,
    author=AUTHOR_USER_NAME,
    author_email=AUTHOR_EMAIL,
    description="A small python package for CNN app",
    long_description=long_description,
    long_description_content="text/markdown",
    url=f"https://github.com/{AUTHOR_USER_NAME}/{REPO_NAME}",
    project_urls={
        "Bug Tracker": f"https://github.com/{AUTHOR_USER_NAME}/{REPO_NAME}/issues",
    },
    package_dir={"": "src"},
    packages=setuptools.find_packages(where="src")
)
```

- Create virtual environment : Create a conda environment after opening the repository

  > conda create -n kidney python=3.8 -y
  > conda activate kidney
  > pip install -r requirements.txt
  >commit changes in github

Step 4: Logging Module

- It is important to track the flow of code, when trying to implement code.
- Create logging module in: **src/cnnClassifier/__init__**
- I can create another folder also **src/logger.py BUT,** I will use above.

```
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)  #it will prints all log in terminal as well
    ]
)

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

- Create a file **test.py**
- run > **python test.py** to check above code running

```
from src.cnnClassifier import logger

logger.info("Welcome to our custom Log")
```

Step 5: Utils Module

- Those functions, we are using frequently in a code or project, we call this as a utility function.
- So, I can write such function/method in utils, and whenever I require, I can call that function from **src/cnnClassifier/utils**.
- Create "**common.py**" inside above folder.
- from **box.exceptions** import **BoxValueError:** it is used to handle exceptions

```
import os
from box.exceptions import BoxValueError
import yaml
from cnnClassifier import logger
import json
import joblib
from ensure import ensure_annotations
from box import ConfigBox
from pathlib import Path
from typing import Any
import base64

@ensure_annotations
def read_yaml(path_to_yaml: Path) -> ConfigBox:
    """reads yaml file and returns

    Args:
        path_to_yaml (str): path like input
    Raises:
        ValueError: if yaml file is empty
        e: empty file

    Returns:
        ConfigBox: ConfigBox type
    """
    try:
        with open(path_to_yaml) as yaml_file:
            content = yaml.safe_load(yaml_file)
            logger.info(f"yaml file: {path_to_yaml} loaded successfully")
            return ConfigBox(content)
    except BoxValueError:
```

```python
        raise ValueError("yaml file is empty")
    except Exception as e:
        raise e
@ensure_annotations
def create_directories(path_to_directories: list, verbose=True):
    """create list of directories

    Args:
        path_to_directories (list): list of path of directories
        ignore_log (bool, optional): ignore if multiple dirs is to be created.
Defaults to False.
    """
    for path in path_to_directories:
        os.makedirs(path, exist_ok=True)
        if verbose:
            logger.info(f"created directory at: {path}")


@ensure_annotations
def save_json(path: Path, data: dict):
    """save json data

    Args:
        path (Path): path to json file
        data (dict): data to be saved in json file
    """
    with open(path, "w") as f:
        json.dump(data, f, indent=4)

    logger.info(f"json file saved at: {path}")




@ensure_annotations
def load_json(path: Path) -> ConfigBox:
    """load json files data

    Args:
        path (Path): path to json file

    Returns:
        ConfigBox: data as class attributes instead of dict
    """
    with open(path) as f:
        content = json.load(f)

    logger.info(f"json file loaded succesfully from: {path}")
    return ConfigBox(content)
```

```python
@ensure_annotations
def save_bin(data: Any, path: Path):
    """save binary file

    Args:
        data (Any): data to be saved as binary
        path (Path): path to binary file
    """
    joblib.dump(value=data, filename=path)
    logger.info(f"binary file saved at: {path}")


@ensure_annotations
def load_bin(path: Path) -> Any:
    """load binary data

    Args:
        path (Path): path to binary file

    Returns:
        Any: object stored in the file
    """
    data = joblib.load(path)
    logger.info(f"binary file loaded from: {path}")
    return data

@ensure_annotations
def get_size(path: Path) -> str:
    """get size in KB

    Args:
        path (Path): path of the file

    Returns:
        str: size in KB
    """
    size_in_kb = round(os.path.getsize(path)/1024)
    return f"~ {size_in_kb} KB"


def decodeImage(imgstring, fileName):
    imgdata = base64.b64decode(imgstring)
    with open(fileName, 'wb') as f:
        f.write(imgdata)
        f.close()
def encodeImageIntoBase64(croppedImagePath):
    with open(croppedImagePath, "rb") as f:
        return base64.b64encode(f.read())
```

- Use of **@ensure_annotations** and **box.exceptions** is described in **research/trials.ipynb** file.
- Commit changes and push to github.
- Open **README.md**

Step 6: Project Workflow

Workflow:

1. **Update config.yaml**
2. **Update secrets.yaml [Optional]**
3. **Update params.yaml**
4. **Update the entity**
5. **Update the configuration manager in src config**
6. **Update the components**
7. **Update the pipeline**
8. **Update the main.py**
9. **Update the dvc.yaml**
10. **app.py**

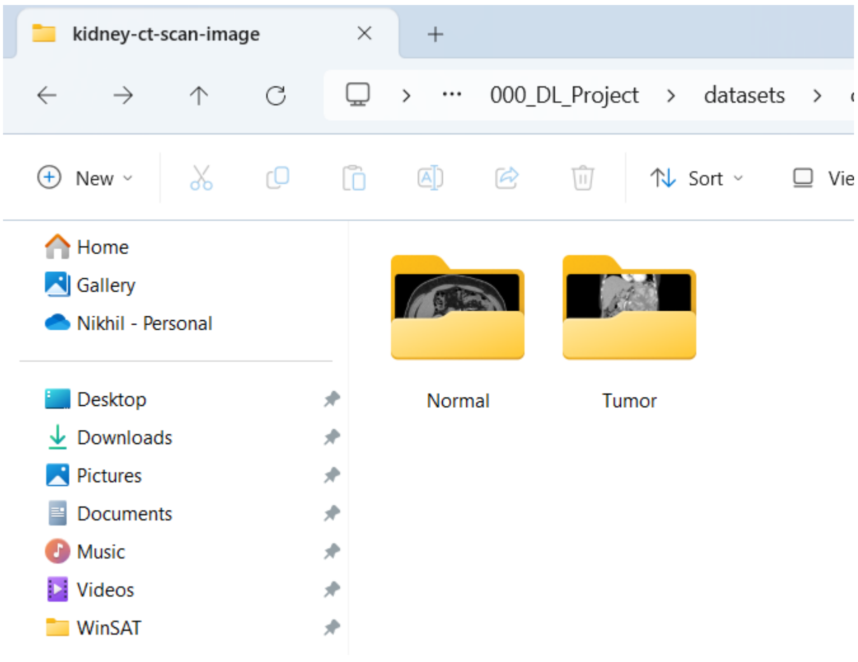
- **Workflows added →>> commit github**

Step 7: Data Ingestion workflow

*Data is store as **kidney-city-scan-image.zip***

Can be downloaded from Kaggle and store in google drive.

Link:
https://drive.google.com/file/d/1v2_34fuf3nM2AE6Nrv6k8FIv5TZCS4Fv/view?usp=sharing



Follow research/trials.ipynb to download from google drive.

```
## Donwload data from gdrive
```

- Before that follow the above workflow
- Open **config/config.yaml** and **update** it.
- Here get data from any url.

```
artifacts_root: artifacts

data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL:https://drive.google.com/file/d/1v2_34fuf3nM2AE6Nrv6k8FIv5TZCS4Fv/view?usp=sharing
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion
```

- Before implementing actual components, will do experiment on notebook side. Once run fine, we will copy paste same code in modular coding format.
- Create stage 1 file in research folder: >> **research/01_data_ingestion.ipynb (connect kernel to run file) and run the mentioned file.**

```
#Unzipping file
```

- **1.** Update config.yaml **completed**
- **2.** Update secrets.yaml **[Optional]**>>> I don't have secret>>> so it is also completed
- **3. Params.yaml** >>> we can't keep it empty>>> so write **"key:val"** in it**,** when I will create model, I will update it.
- Go to >> **research/01_data_ingestion.ipynb >>** write code

```
from collections import namedtuple
import os
```

- Whenever we create any methods or class, what would be its return type, is called **entity**.

```
DataIngestionConfig = namedtuple("DataIngestionConfig", [
    "root_dir",
    "source_URL",
    "local_data_file",
    "unzip_dir"
])
#This is a return type of configuration manager class
#This data, we are taking from config/config.yaml
# Above both are same things
```

```
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
```

<u>Both are same thing to read data:</u>

- We have created **4. entity successfully**
- Now I need to write **5.** Configuration manager in **“research/01_data_ingestion.ipynb” itself.**
- Go to **src/cnnClassifier/constants/__init__.py**

```
• from pathlib import Path
•
• CONFIG_FILE_PATH = Path("config/config.yaml") #this will take care your path,
  whether you are in windows/linux machine
• PARAMS_FILE_PATH = Path("params.yaml")
```

- I will come back to my notebook “**research/01_data_ingestion.ipynb**”.

- Go to **src/cnnClassifier/utils/__init__.py** and paste below thing

```
from cnnClassifier.utils.common import *
```

- I will come back to my notebook “**research/01_data_ingestion.ipynb**”

```
from cnnClassifier.constants import *
from cnnClassifier.utils import read_yaml, create_directories
```

Create configuration manager in **src/config/configuration.py**

```
# Create configuration manager class
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])  #it will create artifacts
folder

    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
```

- 5.update the configuration manager in src config >>> **DONE**
- 6. Update the **components**>>> This is next task

```python
import os
import zipfile
import gdown
from cnnClassifier import logger
from cnnClassifier.utils.common import get_size

class DataIngestion:
    def __init__(self, config: DataIngestionConfig):
        self.config = config


    def download_file(self)-> str:
        '''
        Fetch data from the url
        '''

        try:
            dataset_url = self.config.source_URL
            zip_download_dir = self.config.local_data_file
            os.makedirs("artifacts/data_ingestion", exist_ok=True)
            logger.info(f"Downloading data from {dataset_url} into file {zip_download_dir}")

            file_id = dataset_url.split("/")[-2]
            prefix = 'https://drive.google.com/uc?/export=download&id='
            gdown.download(prefix+file_id,zip_download_dir)

            logger.info(f"Downloaded data from {dataset_url} into file {zip_download_dir}")

        except Exception as e:
            raise e

def extract_zip_file(self):
        """
        zip_file_path: str
        Extracts the zip file into the data directory
        Function returns None
        """
        unzip_path = self.config.unzip_dir
        os.makedirs(unzip_path, exist_ok=True)
        with zipfile.ZipFile(self.config.local_data_file, 'r') as zip_ref:
            zip_ref.extractall(unzip_path)
```

7.Update the pipeline

```
try:
    config = ConfigurationManager()
    data_ingestion_config = config.get_data_ingestion_config()
    data_ingestion = DataIngestion(config=data_ingestion_config)
    data_ingestion.download_file()
    data_ingestion.extract_zip_file()
except Exception as e:
    raise e
```

Refer 01_data_ingestion.ipynb

NOW IT IS IMPORTANT TO CONVERT ABOVE CODE AS MODULAR CODING

001***Step by step- Update the following files for *DATA INGESTION component****

- **Update config.yaml ------------ Already updated**
- **Update secrets.yaml [Optional] ----------------------------- Already updated**
- **Update params.yaml----------------------------- Already updated**
- **4. Update 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
```

- **5. Update src/cnnClassifier/config/configuration.py**

```
from cnnClassifier.constants import *
from cnnClassifier.utils.common import read_yaml, create_directories
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
```

➢ **6. Update src/cnnClassifier/components/data_ingestion.py**

```
import os
import zipfile
import gdown
from cnnClassifier import logger
from cnnClassifier.utils.common import get_size
from cnnClassifier.entity.config_entity import (DataIngestionConfig)


class DataIngestion:
    def __init__(self, config: DataIngestionConfig):
        self.config = config

    def download_file(self)-> str:
        '''
        Fetch data from the url
        '''

        try:
            dataset_url = self.config.source_URL
            zip_download_dir = self.config.local_data_file
            os.makedirs("artifacts/data_ingestion", exist_ok=True)
            logger.info(f"Downloading data from {dataset_url} into file
{zip_download_dir}")

            file_id = dataset_url.split("/")[-2]
            prefix = 'https://drive.google.com/uc?/export=download&id='
            gdown.download(prefix+file_id,zip_download_dir)
            logger.info(f"Downloaded data from {dataset_url} into file
{zip_download_dir}")

        except Exception as e:
            raise e
    def extract_zip_file(self):
        """
        zip_file_path: str
        Extracts the zip file into the data directory
        Function returns None
        """
        unzip_path = self.config.unzip_dir
        os.makedirs(unzip_path, exist_ok=True)
        with zipfile.ZipFile(self.config.local_data_file, 'r') as zip_ref:
            zip_ref.extractall(unzip_path)
```

➢ **7. Update src/cnnClassifier/pipeline/stage_01_data_ingestion.py**

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.data_ingestion import DataIngestion
from cnnClassifier import logger

STAGE_NAME = "Data Ingestion stage"


class DataIngestionTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        data_ingestion_config = config.get_data_ingestion_config()
        data_ingestion = DataIngestion(config=data_ingestion_config)
        data_ingestion.download_file()
        data_ingestion.extract_zip_file()



if __name__ == '__main__':        #python will start reading from here
    try:
        logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
        obj = DataIngestionTrainingPipeline()
        obj.main()
        logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

➢ **8. Update src/cnnClassifier/main.py**

```
from src.cnnClassifier import logger
from cnnClassifier.pipeline.stage_01_data_ingestion import
DataIngestionTrainingPipeline

# logger.info("Welcome to our custom log")

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

Go to project folder

Remove artifacts folder

Open terminal

>>python main.py

Add artifacts/* in .gitignore file

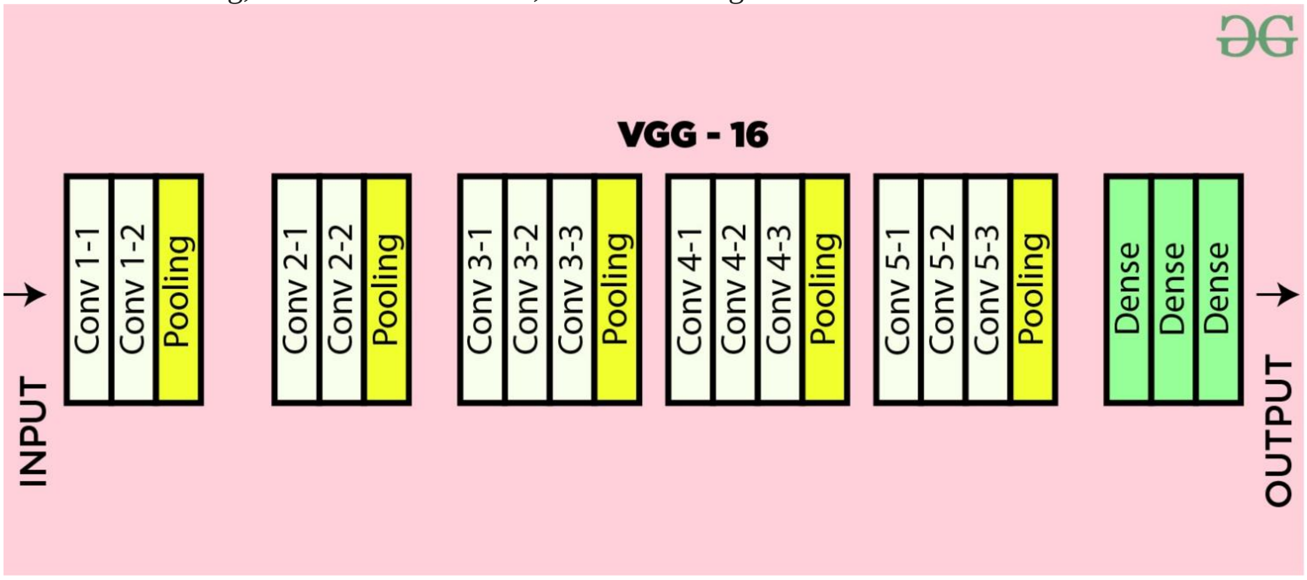
Remove .zip file for research folder and commit the changes in github.

002 **Step by step- Update the following files for *PREPARE BASE MODEL component****

Go to keras application for pretrained models

https://keras.io/api/applications/

I downloaded one model from here and SAVE it with my custom layer. So in next stage whenever I will start training, I will load this model, & start training.



Trained on Imagenet dataset.

I will use Convulation layer of this model and take my own (ANN) layer with last dense layer with 2 classes with SOFTMAX activation function.

- **1. Update config.yaml**

```yaml
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL:
https://drive.google.com/file/d/1v2_34fuf3nM2AE6Nrv6k8FIv5TZCS4Fv/view?usp=sharing
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion


prepare_base_model:
  root_dir: artifacts/prepare_base_model
  base_model_path: artifacts/prepare_base_model/base_model.h5
  updated_base_model_path: artifacts/prepare_base_model/base_model_updated.h5
```

Follow the notebook **research/02_prepare_base_model.ipynb** for complete code:

- **2. Update secrets.yaml [Optional]**
- **3. Update params.yaml**

```
➢ AUGMENTATION: False
➢ IMAGE_SIZE: [224, 224, 3] # as per VGG 16 model
➢ BATCH_SIZE: 16
➢ INCLUDE_TOP: False   ##because I don’t want to include last ANN layers
➢ EPOCHS: 4
➢ CLASSES: 2
➢ WEIGHTS: imagenet
➢ LEARNING_RATE: 0.01
```

- **4. Update 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
```

➢ 5. Update src/cnnClassifier/config/configuration.py

```python
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories
from cnnClassifier.entity.config_entity import (DataIngestionConfig,
                                                PrepareBaseModelConfig)


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


    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
```

6. Update src/cnnClassifier/components/prepare_base_model.py

```
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.vgg16.VGG16(
            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):
        if freeze_all:
            for layer in model.layers:
                model.trainable = False
        elif (freeze_till is not None) and (freeze_till > 0):
            for layer in model.layers[:-freeze_till]:
                model.trainable = False

        flatten_in = tf.keras.layers.Flatten()(model.output)
        prediction = tf.keras.layers.Dense(
            units=classes,
            activation="softmax"
        )(flatten_in)

        full_model = tf.keras.models.Model(
            inputs=model.input,
            outputs=prediction
        )
```

```python

        full_model.compile(
            optimizer=tf.keras.optimizers.SGD(learning_rate=learning_rate),
            loss=tf.keras.losses.CategoricalCrossentropy(),
            metrics=["accuracy"]
        )

        full_model.summary()
        return full_model


    def update_base_model(self):
        self.full_model = self._prepare_full_model(
            model=self.model,
            classes=self.config.params_classes,
            freeze_all=True,
            freeze_till=None,
            learning_rate=self.config.params_learning_rate
        )

        self.save_model(path=self.config.updated_base_model_path, model=self.full_model)



    @staticmethod
    def save_model(path: Path, model: tf.keras.Model):
        model.save(path)
```

- **7. Update src/cnnClassifier/pipeline/stage_02_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()
```

```python
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==========x")
    except Exception as e:
        logger.exception(e)
        raise e
```

➢ **8. Update src/cnnClassifier/main.py**

```python
from src.cnnClassifier import logger
from cnnClassifier.pipeline.stage_01_data_ingestion import
DataIngestionTrainingPipeline
from cnnClassifier.pipeline.stage_02_prepare_base_model import
PrepareBaseModelTrainingPipeline

# logger.info("Welcome to our custom log")

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==========x")
except Exception as e:
        logger.exception(e)
        raise e


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

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py >>>>Remove .zip file for research folder and commit the changes in github.

003 **Step by step- Update the following files for *MODEL TRAINING component*****

Refer file **research/03_model_training.ipynb**

- **1. Update config.yaml**

```
artifacts_root: artifacts


data_ingestion:
  root_dir: artifacts/data_ingestion
  source_URL:
https://drive.google.com/file/d/1v2_34fuf3nM2AE6Nrv6k8FIv5TZCS4Fv/view?usp=sharing
  local_data_file: artifacts/data_ingestion/data.zip
  unzip_dir: artifacts/data_ingestion


prepare_base_model:
  root_dir: artifacts/prepare_base_model
  base_model_path: artifacts/prepare_base_model/base_model.h5
  updated_base_model_path: artifacts/prepare_base_model/base_model_updated.h5


training:
  root_dir: artifacts/training
  trained_model_path: artifacts/training/model.h5    ###This model I will use for
prediction
```

- **2.Update secrets.yaml [Optional] ------------------------------ Already updated**
- **3.Update params.yaml------------------------------ Already updated**

- **4. Update 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


@dataclass(frozen=True)
class TrainingConfig:
    root_dir: Path
    trained_model_path: Path
    updated_base_model_path: Path
    training_data: Path
    params_epochs: int
    params_batch_size: int
    params_is_augmentation: bool
    params_image_size: list
```

➢ **5. Update src/cnnClassifier/config/configuration.py**

```
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import (DataIngestionConfig,
                                                PrepareBaseModelConfig,
                                                TrainingConfig)


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


    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


    def get_training_config(self) -> TrainingConfig:
        training = self.config.training
        prepare_base_model = self.config.prepare_base_model
        params = self.params
        training_data = os.path.join(self.config.data_ingestion.unzip_dir, "kidney-
ct-scan-image")
        create_directories([
            Path(training.root_dir)
        ])
        training_config = TrainingConfig(
            root_dir=Path(training.root_dir),
            trained_model_path=Path(training.trained_model_path),
            updated_base_model_path=Path(prepare_base_model.updated_base_model_path),
            training_data=Path(training_data),
            params_epochs=params.EPOCHS,
            params_batch_size=params.BATCH_SIZE,
            params_is_augmentation=params.AUGMENTATION,
            params_image_size=params.IMAGE_SIZE
        )

        return training_config
```

➢ **6. Update src/cnnClassifier/components/model_training.py**

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


class Training:
    def __init__(self, config: TrainingConfig):
        self.config = config


    def get_base_model(self):
        self.model = tf.keras.models.load_model(
            self.config.updated_base_model_path
        )

    def train_valid_generator(self):

        datagenerator_kwargs = dict(
            rescale = 1./255,
            validation_split=0.20
        )

        dataflow_kwargs = dict(
            target_size=self.config.params_image_size[:-1],
            batch_size=self.config.params_batch_size,
            interpolation="bilinear"
        )

        valid_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
            **datagenerator_kwargs
        )

        self.valid_generator = valid_datagenerator.flow_from_directory(
            directory=self.config.training_data,
            subset="validation",
            shuffle=False,
            **dataflow_kwargs
        )

        if self.config.params_is_augmentation:
            train_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
                rotation_range=40,
                horizontal_flip=True,
                width_shift_range=0.2,
                height_shift_range=0.2,
                shear_range=0.2,
                zoom_range=0.2,
                **datagenerator_kwargs
            )
        else:
            train_datagenerator = valid_datagenerator

        self.train_generator = train_datagenerator.flow_from_directory(
            directory=self.config.training_data,
            subset="training",
            shuffle=True,
            **dataflow_kwargs
        )


    @staticmethod
    def save_model(path: Path, model: tf.keras.Model):
        model.save(path)

    def train(self):
        self.steps_per_epoch = self.train_generator.samples //
self.train_generator.batch_size
        self.validation_steps = self.valid_generator.samples //
self.valid_generator.batch_size

        self.model.fit(
            self.train_generator,
            epochs=self.config.params_epochs,
            steps_per_epoch=self.steps_per_epoch,
            validation_steps=self.validation_steps,
            validation_data=self.valid_generator
        )

        self.save_model(
            path=self.config.trained_model_path,
            model=self.model
        )
```

➢ **7. Update src/cnnClassifier/pipeline/stage_03_model_training.py**

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.model_training import Training
from cnnClassifier import logger


STAGE_NAME = "Training"



class ModelTrainingPipeline:
    def __init__(self):
        pass

    def main(self):
        config = ConfigurationManager()
        training_config = config.get_training_config()
        training = Training(config=training_config)
        training.get_base_model()
        training.train_valid_generator()
        training.train()



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

➢ **8. Update src/cnnClassifier/main.py**

```
from src.cnnClassifier import logger
from cnnClassifier.pipeline.stage_01_data_ingestion import
DataIngestionTrainingPipeline
```

```python
from cnnClassifier.pipeline.stage_02_prepare_base_model import
PrepareBaseModelTrainingPipeline
from cnnClassifier.pipeline.stage_03_model_training import ModelTrainingPipeline

# logger.info("Welcome to our custom log")

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==========x")
except Exception as e:
        logger.exception(e)
        raise e


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==========x")
except Exception as e:
        logger.exception(e)
        raise e


STAGE_NAME = "Training"
try:
   logger.info(f"*******************")
   logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
   model_trainer = ModelTrainingPipeline()
   model_trainer.main()
   logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e
```

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py

>>>>Remove .zip file for research folder and commit the changes in github.

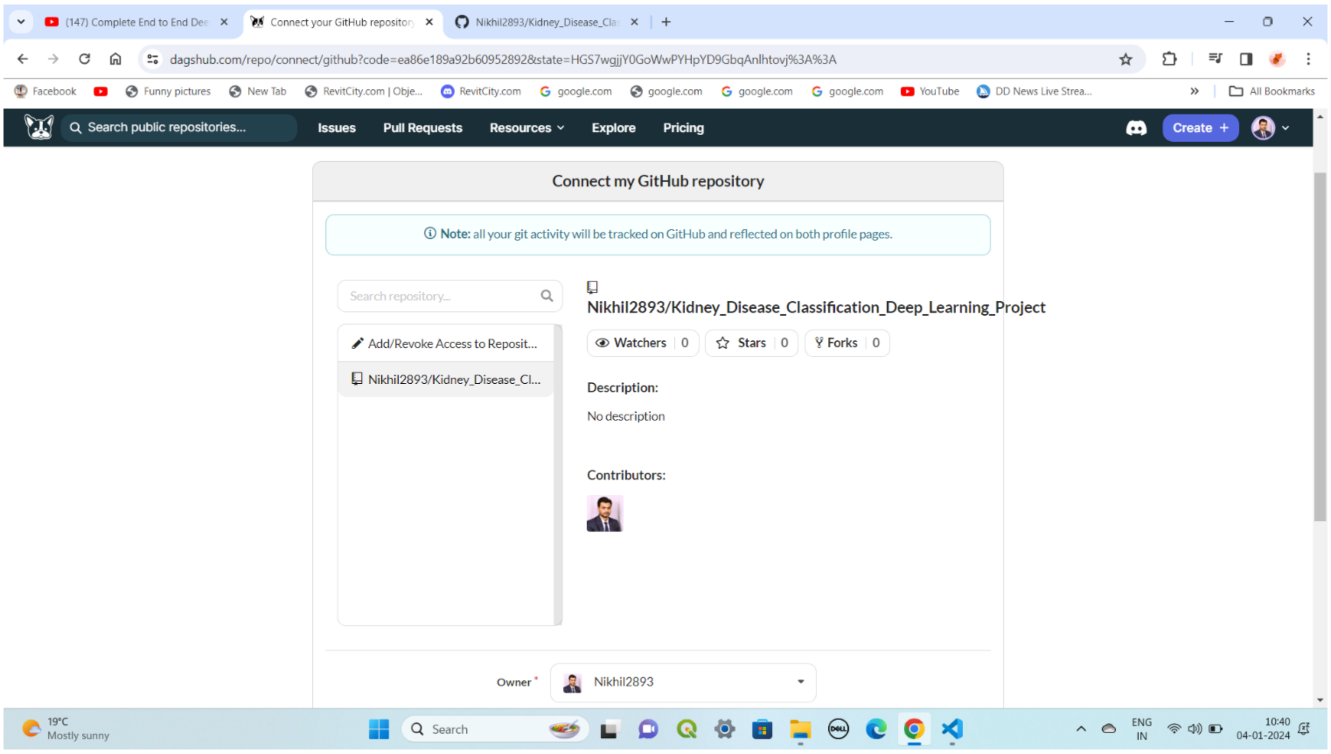
004 **Step by step- Update the following files for *MODEL EVALUATION & MLflow INTEGRATION component****

Agenda : How we can integrate MLflow with this project and will be tracking experiment and how will register model?

Refer : research/04_model_evaluation_with_MLflow.ipynb

~~~
## MLflow

- [Documentation](https://mlflow.org/docs/latest/index.html)

- [MLflow tutorial](https://youtu.be/qdcHHrsXA48?si=bD5vDS60akNphkem)

##### cmd
- mlflow ui

### dagshub
[dagshub](https://dagshub.com/)

MLFLOW_TRACKING_URI=https://dagshub.com/Nikhil2893/Kidney_Disease_Classification_Deep
_Learning_Project.mlflow \
MLFLOW_TRACKING_USERNAME=Nikhil2893 \
MLFLOW_TRACKING_PASSWORD=ff53be44bc1a7eb176f12bbfb8cee04567631847 \
python script.py

Run this to export as env variables:

### ***Execute below commands in bash terminal***

```bash

export
MLFLOW_TRACKING_URI=https://dagshub.com/Nikhil2893/Kidney_Disease_Classification_Deep
_Learning_Project.mlflow

export MLFLOW_TRACKING_USERNAME=Nikhil2893

export MLFLOW_TRACKING_PASSWORD=ff53be44bc1a7eb176f12bbfb8cee04567631847
~~~

- Connect dagshub with github repository
- Using dagshub , we can launch MLFlow server
- Can track different experiments with different parameters
- Model registry also possible in MLflow, without doing it manually.



dagshub.com/repo/connect/github?code=ea86e189a92b60952892&state=HGS7wgjjY0GoWwPYHpYD9GbqAnlhtovj%3A%3A

Connect my GitHub repository

Note: all your git activity will be tracked on GitHub and reflected on both profile pages.

Search repository...

Add/Revoke Access to Reposit...

Nikhil2893/Kidney_Disease_Cl...

Nikhil2893/Kidney_Disease_Classification_Deep_Learning_Project

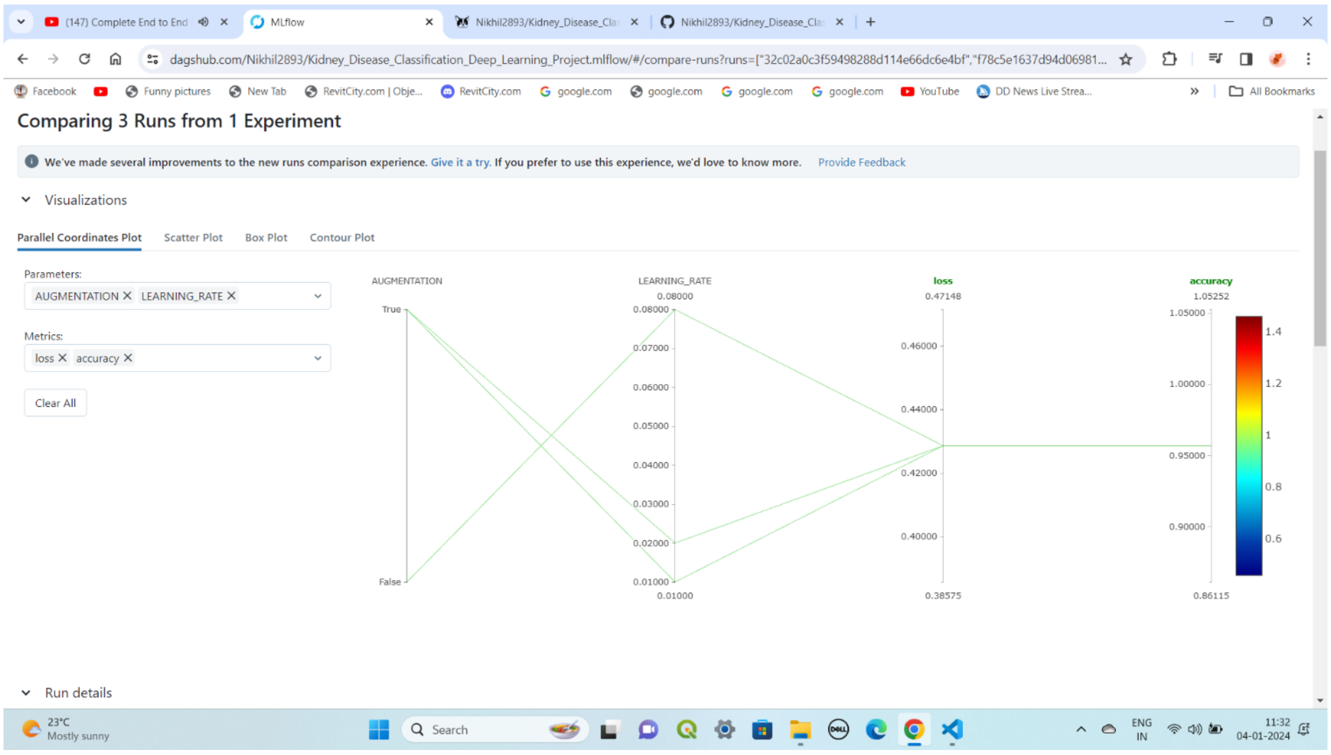
Watchers 0 | Stars 0 | Forks 0

Description:

No description

Contributors:

Owner * Nikhil2893



dagshub.com/Nikhil2893/Kidney_Disease_Classification_Deep_Learning_Project.mlflow/#/compare-runs?runs=["32c02a0c3f59498288d114e66dc6e4bf","f78c5e1637d94d06981...

Comparing 3 Runs from 1 Experiment

We've made several improvements to the new runs comparison experience. Give it a try. If you prefer to use this experience, we'd love to know more. Provide Feedback

Visualizations

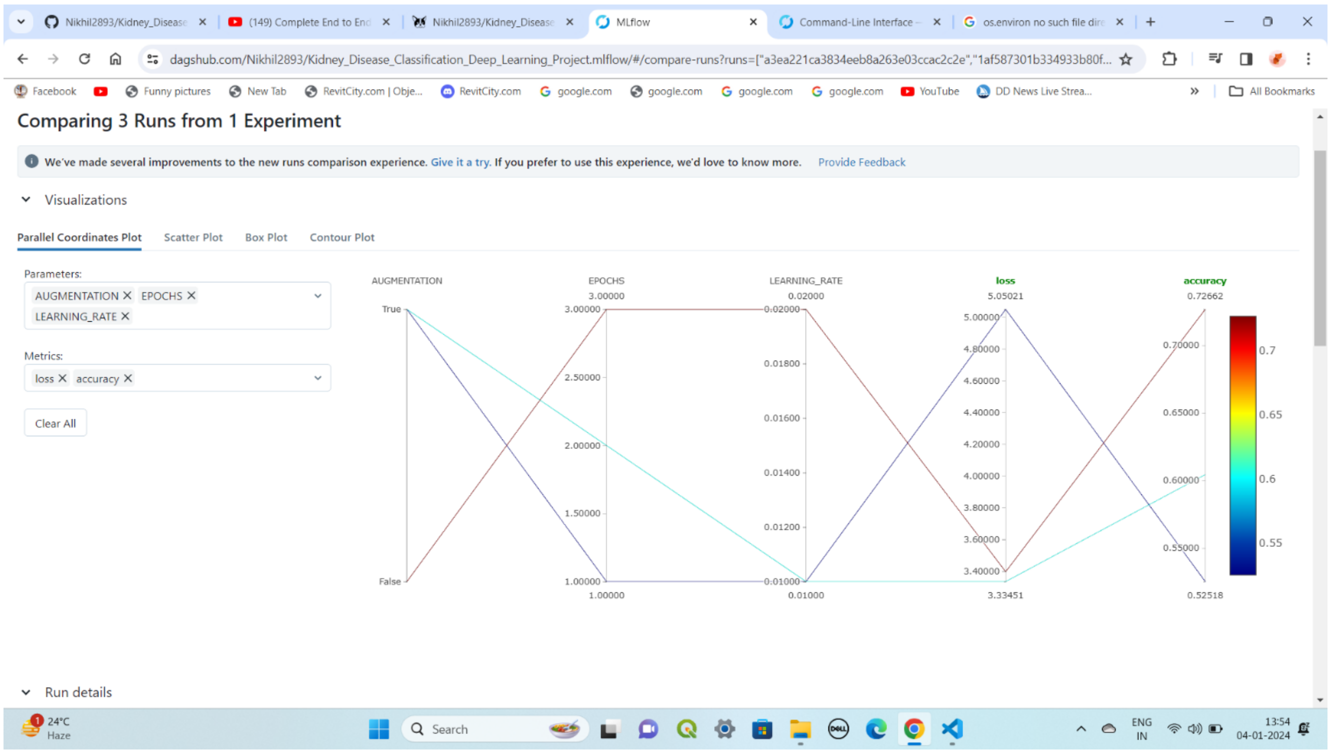
Parallel Coordinates Plot | Scatter Plot | Box Plot | Contour Plot

Parameters: AUGMENTATION ✕ LEARNING_RATE ✕

Metrics: loss ✕ accuracy ✕

Clear All

Run details


Comparing 3 Runs from 1 Experiment
We've made several improvements to the new runs comparison experience. Give it a try. If you prefer to use this experience, we'd love to know more. Provide Feedback
Visualizations
Parallel Coordinates Plot
Scatter Plot
Box Plot
Contour Plot
Parameters:
AUGMENTATION
EPOCHS
LEARNING_RATE
Metrics:
loss
accuracy
Clear All
AUGMENTATION
EPOCHS
LEARNING_RATE
loss
accuracy
Run details

1. **Update config.yaml-----ALREADY DONE**
2. **Update secrets.yaml [Optional]------------ ALREADY DONE**
3. **Update params.yaml-------------- ALREADY DONE**

- **4. Update 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

@dataclass(frozen=True)
class TrainingConfig:
    root_dir: Path
    trained_model_path: Path
    updated_base_model_path: Path
    training_data: Path
    params_epochs: int
    params_batch_size: int
    params_is_augmentation: bool
    params_image_size: list

@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
```

- **5. Update src/cnnClassifier/config/configuration.py**

```
from cnnClassifier.constants import *
import os
from cnnClassifier.utils.common import read_yaml, create_directories,save_json
from cnnClassifier.entity.config_entity import (DataIngestionConfig,
                                                PrepareBaseModelConfig,
                                                TrainingConfig,
                                                EvaluationConfig)
```

```python

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


    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


    def get_training_config(self) -> TrainingConfig:
        training = self.config.training
```

```
➢         prepare_base_model = self.config.prepare_base_model
➢         params = self.params
➢         training_data = os.path.join(self.config.data_ingestion.unzip_dir, "kidney-ct-scan-image")
➢         create_directories([
➢             Path(training.root_dir)
➢         ])
➢
➢         training_config = TrainingConfig(
➢             root_dir=Path(training.root_dir),
➢             trained_model_path=Path(training.trained_model_path),
➢             updated_base_model_path=Path(prepare_base_model.updated_base_model_path),
➢             training_data=Path(training_data),
➢             params_epochs=params.EPOCHS,
➢             params_batch_size=params.BATCH_SIZE,
➢             params_is_augmentation=params.AUGMENTATION,
➢             params_image_size=params.IMAGE_SIZE
➢         )
➢
➢         return training_config
➢
➢
➢     def get_evaluation_config(self) -> EvaluationConfig:
➢         eval_config = EvaluationConfig(
➢             path_of_model="artifacts/training/model.h5",
➢             training_data="artifacts/data_ingestion/kidney-ct-scan-image",
➢             mlflow_uri="https://dagshub.com/Nikhil2893/Kidney_Disease_Classification_Deep_Learning_Project.mlflow",
➢             all_params=self.params,
➢             params_image_size=self.params.IMAGE_SIZE,
➢             params_batch_size=self.params.BATCH_SIZE
➢         )
➢         return eval_config
```

➢ **6. Update src/cnnClassifier/components/model_evaluation_with_MLflow.py**

```
➢ import tensorflow as tf
➢ from pathlib import Path
➢ import mlflow
➢ import mlflow.keras
➢ from urllib.parse import urlparse
➢ from cnnClassifier.entity.config_entity import EvaluationConfig
➢ from cnnClassifier.utils.common import create_directories,read_yaml,save_json
➢
➢ class Evaluation:
➢     def __init__(self, config: EvaluationConfig):
➢         self.config = config
➢
```

```python

    def _valid_generator(self):

        datagenerator_kwargs = dict(
            rescale = 1./255,
            validation_split=0.30
        )

        dataflow_kwargs = dict(
            target_size=self.config.params_image_size[:-1],
            batch_size=self.config.params_batch_size,
            interpolation="bilinear"
        )

        valid_datagenerator = tf.keras.preprocessing.image.ImageDataGenerator(
            **datagenerator_kwargs
        )

        self.valid_generator = valid_datagenerator.flow_from_directory(
            directory=self.config.training_data,
            subset="validation",
            shuffle=False,
            **dataflow_kwargs
        )

    @staticmethod
    def load_model(path: Path) -> tf.keras.Model:
        return tf.keras.models.load_model(path)


    def evaluation(self):
        self.model = self.load_model(self.config.path_of_model)
        self._valid_generator()
        self.score = self.model.evaluate(self.valid_generator)
        self.save_score()

    def save_score(self):
        scores = {"loss": self.score[0], "accuracy": self.score[1]}
        save_json(path=Path("scores.json"), data=scores)


    def log_into_mlflow(self):
        mlflow.set_registry_uri(self.config.mlflow_uri)
        tracking_url_type_store = urlparse(mlflow.get_tracking_uri()).scheme

        with mlflow.start_run():
            mlflow.log_params(self.config.all_params)
            mlflow.log_metrics(
                {"loss": self.score[0], "accuracy": self.score[1]}
```

```
➢                 )
➢                 # Model registry does not work with file store
➢                 if tracking_url_type_store != "file":
➢
➢                     # Register the model
➢                     # There are other ways to use the Model Registry, which depends
   on the use case,
➢                     # please refer to the doc for more information:
➢                     # https://mlflow.org/docs/latest/model-registry.html#api-
   workflow
➢                     mlflow.keras.log_model(self.model, "model",
   registered_model_name="VGG16Model")
➢                 else:
➢                     mlflow.keras.log_model(self.model, "model")
```

➢ **7. Update src/cnnClassifier/pipeline/stage_04_model_evaluation.py**

```
from cnnClassifier.config.configuration import ConfigurationManager
from cnnClassifier.components.model_evaluation_mlflow import Evaluation
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)
        evaluation.evaluation()
        evaluation.save_score()
        # evaluation.log_into_mlflow()

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

➢ **8. Update src/cnnClassifier/main.py**

```python
from src.cnnClassifier import logger
from cnnClassifier.pipeline.stage_01_data_ingestion import DataIngestionTrainingPipeline
from cnnClassifier.pipeline.stage_02_prepare_base_model import PrepareBaseModelTrainingPipeline
from cnnClassifier.pipeline.stage_03_model_training import ModelTrainingPipeline
from cnnClassifier.pipeline.stage_04_model_evaluation import EvaluationPipeline

# logger.info("Welcome to our custom log")

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==========x")
except Exception as e:
        logger.exception(e)
        raise e

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==========x")
except Exception as e:
        logger.exception(e)
        raise e

STAGE_NAME = "Training"
try:
    logger.info(f"*******************")
    logger.info(f">>>>>> stage {STAGE_NAME} started <<<<<<")
    model_trainer = ModelTrainingPipeline()
    model_trainer.main()
    logger.info(f">>>>>> stage {STAGE_NAME} completed <<<<<<\n\nx==========x")
except Exception as e:
        logger.exception(e)
        raise e

```

```
➢ 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
  <<<<<<\n\nx==========x")
➢
➢ except Exception as e:
➢         logger.exception(e)
➢         raise e
```

Go to project folder >> Remove artifacts folder

Open terminal >>python main.py

>>>>Remove .zip file for research folder and commit the changes in github.

>>>> Save best model in *model/model.h5*

➢ 9. Update dvc.yaml

Use for PIPELINE TRACKING:

Whenever I am executing main.py

1. Main.py is connected with pipeline (**src/cnnClassifier/pipeline**)

First it run and execute one by one

- **DataIngestionTrainingPipeline()**
- **PrepareBaseModelTrainingPipeline()**
- **ModelTrainingPipeline()**
- **EvaluationPipeline()**

Means all components are running one by one.

It will track pipeline. Will check which pipeline is available or not? It will update it.

If any stage is already executed, it will skip that stage and move to next stage.

(data version control can be used to track both ***data*** and ***pipeline***

Go to project folder >> Remove artifacts folder

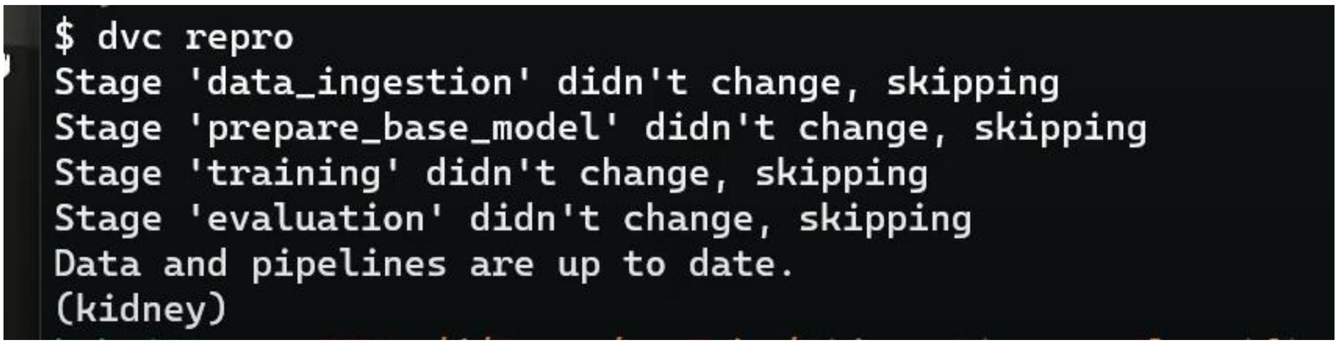
Open terminal >>python main.py

Now Instead executing python main.py, Just >>> **dvc repro**

Dvc.yaml>>> update it

```yaml
stages:
  data_ingestion:
    cmd: python src/cnnClassifier/pipeline/stage_01_data_ingestion.py
    deps:                                                     #dependencies
      - src/cnnClassifier/pipeline/stage_01_data_ingestion.py
      - config/config.yaml
    outs:                                                     #output
      - artifacts/data_ingestion/kidney-ct-scan-image


  prepare_base_model:
    cmd: python src/cnnClassifier/pipeline/stage_02_prepare_base_model.py
    deps:
      - src/cnnClassifier/pipeline/stage_02_prepare_base_model.py
      - config/config.yaml
    params:
      - IMAGE_SIZE
      - INCLUDE_TOP
      - CLASSES
      - WEIGHTS
      - LEARNING_RATE
    outs:
      - artifacts/prepare_base_model


  training:
    cmd: python src/cnnClassifier/pipeline/stage_03_model_training.py
    deps:
      - src/cnnClassifier/pipeline/stage_03_model_training.py
      - config/config.yaml
      - artifacts/data_ingestion/kidney-ct-scan-image
      - artifacts/prepare_base_model

    params:
      - IMAGE_SIZE
      - EPOCHS
      - BATCH_SIZE
      - AUGMENTATION
    outs:
      - artifacts/training/model.h5
  evaluation:
    cmd: python src/cnnClassifier/pipeline/stage_04_model_evaluation.py
    deps:
      - src/cnnClassifier/pipeline/stage_04_model_evaluation.py
      - config/config.yaml
      - artifacts/data_ingestion/kidney-ct-scan-image
      - artifacts/training/model.h5
    params:
      - IMAGE_SIZE
      - BATCH_SIZE
    metrics:
    - scores.json:
        cache: false
```

```
### DVC cmd

1. dvc init
2. dvc repro
3. dvc dag
```



```
$ dvc repro
Stage 'data_ingestion' didn't change, skipping
Stage 'prepare_base_model' didn't change, skipping
Stage 'training' didn't change, skipping
Stage 'evaluation' didn't change, skipping
Data and pipelines are up to date.
(kidney)
```

>>>> **commit the changes in github.**

➢ 10. Prediction pipeline and app.py

Create pipeline.py in src/cnnClassifier/pipeline/prediction.py and update it

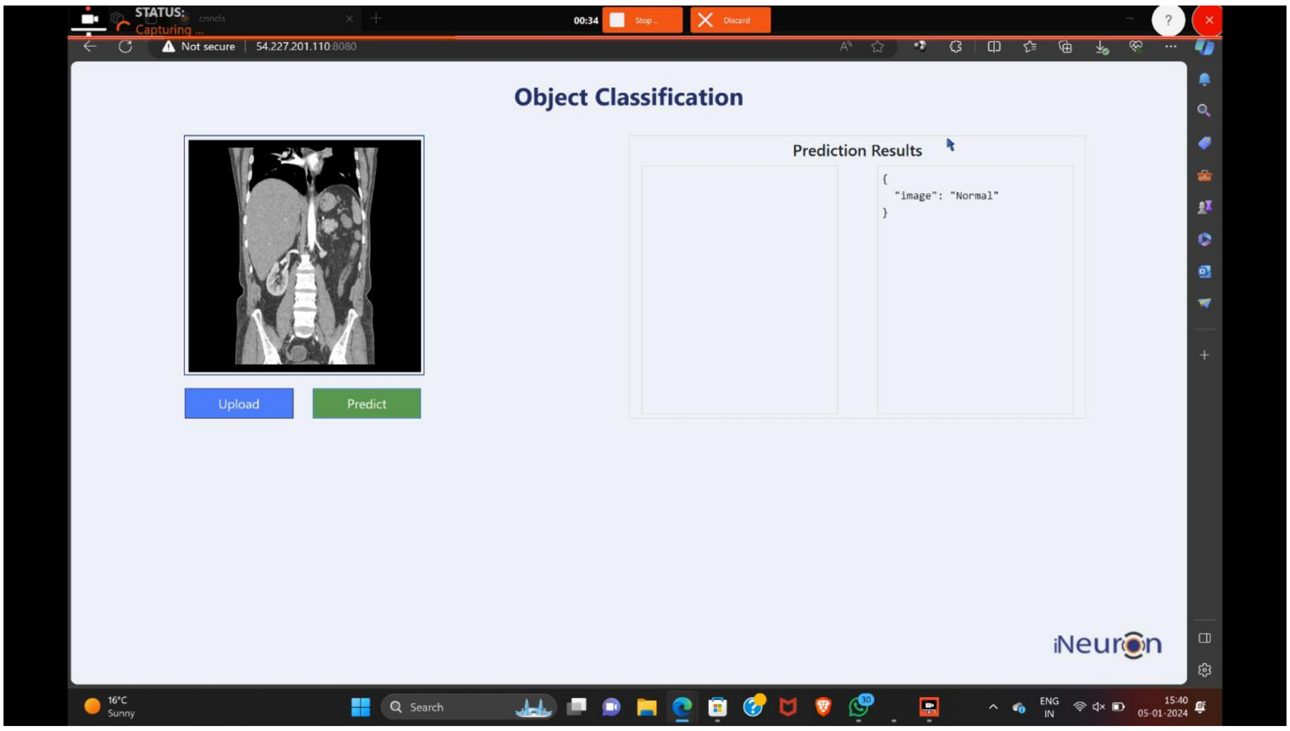
```python
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import os

class PredictionPipeline:
    def __init__(self,filename):
        self.filename =filename
    def predict(self):
        # load model
        model = load_model(os.path.join("model","model.h5"))

        imagename = self.filename          #user input image
        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)
        result = np.argmax(model.predict(test_image), axis=1)
        print(result)

        if result[0] == 1:
            prediction = 'Tumor'
            return [{ "image" : prediction}]
        else:
            prediction = 'Normal'
            return [{ "image" : prediction}]
```

app.py

Refer templates/index.html for html code

```
from flask import Flask, request, jsonify, render_template
import os
from flask_cors import CORS, cross_origin
from cnnClassifier.utils.common 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')
```

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

@app.route("/predict", methods=['POST'])
@cross_origin()
def predictRoute():
    image = request.json['image']
    decodeImage(image, clApp.filename)
    result = clApp.classifier.predict()
    return jsonify(result)


if __name__ == "__main__":
    clApp = ClientApp()
    app.run(host='0.0.0.0', port=8080) #for AWS deployment
```



Object Classification

Prediction Results

```
{
  "image": "Normal"
}
```

## About MLflow & DVC

MLflow

- Its Production Grade
- Trace all of your expriements
- Logging & taging your model

DVC

- Its very lite weight for POC only
- lite weight expriements tracker
- It can perform Orchestration (Creating Pipelines)

# AWS-CICD-Deployment-with-Github-Actions

## 1. Login to AWS console.

## 2. Create IAM user for deployment

#with specific access

1. EC2 access : It is virtual machine
2. ECR: Elastic Container registry to save your docker image in aws

#Description: About the deployment

1. **Build docker image of the source code**

```
FROM python:3.8-slim-buster

RUN apt update -y && apt install awscli -y
WORKDIR /app

COPY . /app
RUN pip install -r requirements.txt

CMD ["python3", "app.py"]
```

2. Push your docker image to ECR
3. Launch Your EC2
4. Pull Your image from ECR in EC2
5. Lauch your docker image in EC2

#Policy:

1. AmazonEC2ContainerRegistryFullAccess
2. AmazonEC2FullAccess

## 3. Create ECR repo to store/save docker image
- Save the URI: 025027871758.dkr.ecr.us-east-1.amazonaws.com/kidney

## 4. Create EC2 machine (Ubuntu)
## 5. Open EC2 and Install docker in EC2 Machine:

```
#optinal

sudo apt-get update -y

sudo apt-get upgrade

#required

curl -fsSL https://get.docker.com -o get-docker.sh

sudo sh get-docker.sh

sudo usermod -aG docker ubuntu

newgrp docker
```

# 6. Configure EC2 as self-hosted runner:
setting>actions>runner>new self hosted runner> choose os> then run command one by one

# 7. Setup github secrets:

AWS_ACCESS_KEY_ID=

AWS_SECRET_ACCESS_KEY=

AWS_REGION = us-east-1

AWS_ECR_LOGIN_URI = demo>> 566373416292.dkr.ecr.ap-south-1.amazonaws.com

ECR_REPOSITORY_NAME = simple-app