Chest Cancer Classification using MLflow-DVC-AWS

Index

11)Future Scope of Enhancement

12)References

ABSTRACT

This MLDevOps project uses the VGG16 CNN model for object detection and classification algorithm which is able to classify upto 1000 images of 1000 different categories with 92.7% accuracy. TensorFlow with Keras was chosen to be used in the model to provide an approachable, highly-productive interface for detecting Adenocarcinoma (Adenocarcinoma is a type of cancerous tumor that can occur in several parts of the body. It is defined as neoplasia of epithelial tissue that has glandular origin, glandular characteristics, or both.) cancer in the chest through chest CT scan datasets with a focus on modern deep learning.

We first train the model with the required basic parameters and log the experiments into MLflow.Then we modify the parameters and rerun the experiment each time and push the logs into MLflow where we compare the results of all the experiments and choose the best parameters based on Accuracy & Loss metrics that have been defined in the source code.

Once we have automated our pipelines, we integrate our source code into Dagshub which acts as a remote server/repository for our project.We then implement DVC (Data Version Control) through a .yaml file to orchestrate our entire pipeline.

We then create an IAM role, an EC2 server and a Elastic Container Registry to host our devops project on AWS Cloud. We create a local self-hosted runner in our github/gitlab Project which stores the AWS Secrets to connect to the AWS Resources.

After setup is complete, when we make a source code update and push the changes to github/gitlab, the pipelines get built and output can be seen on the public IPV4 address of our Amazon EC2 Instance configured on port 8080. In this web page, when we upload a CT scanned image of a chest, the website will tell whether Adenocarcinoma cancer is present or not.

Objective of the Project

The objective of this project is to build an end to end implementation of a chest cancer(Adenocarcinoma) detection system with CICD Deployment through Dagshub remote server for experiment tracking(through the MLflow UI), DVC for Orchestration of the CI/CD pipelines and AWS Cloud for final deployment.

Tools and Requirements

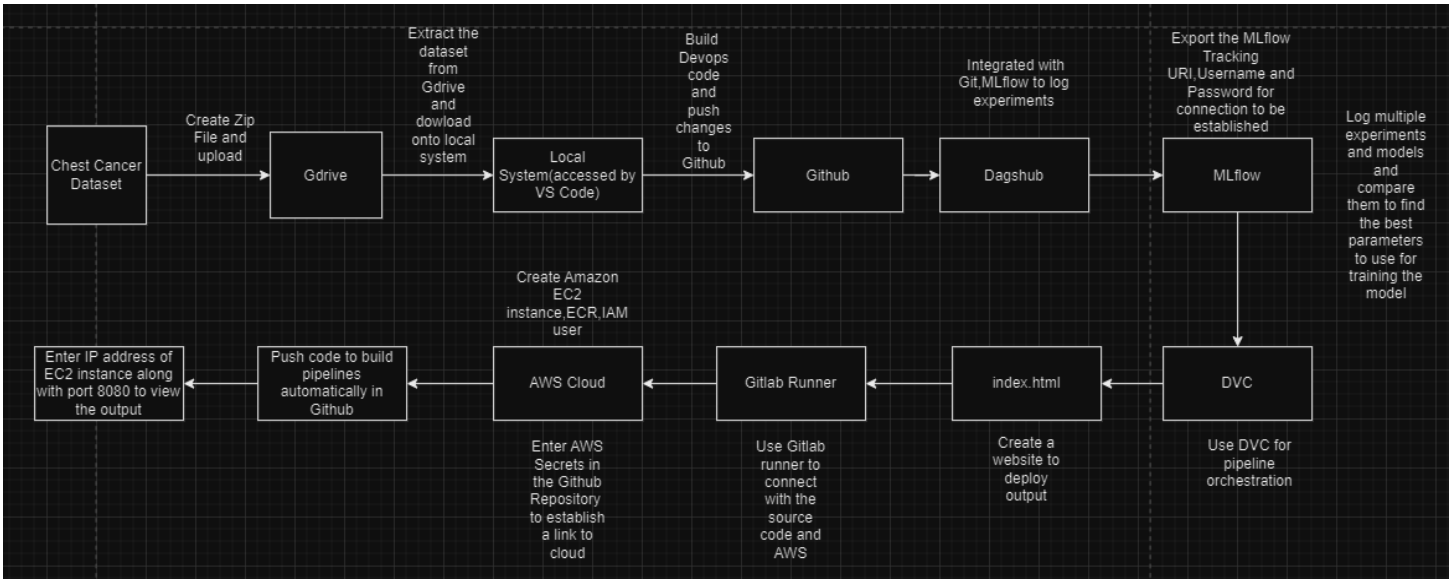
Requirements-

- tensorflow==2.12.0
- Pandas
- gdown
- 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
- -e .

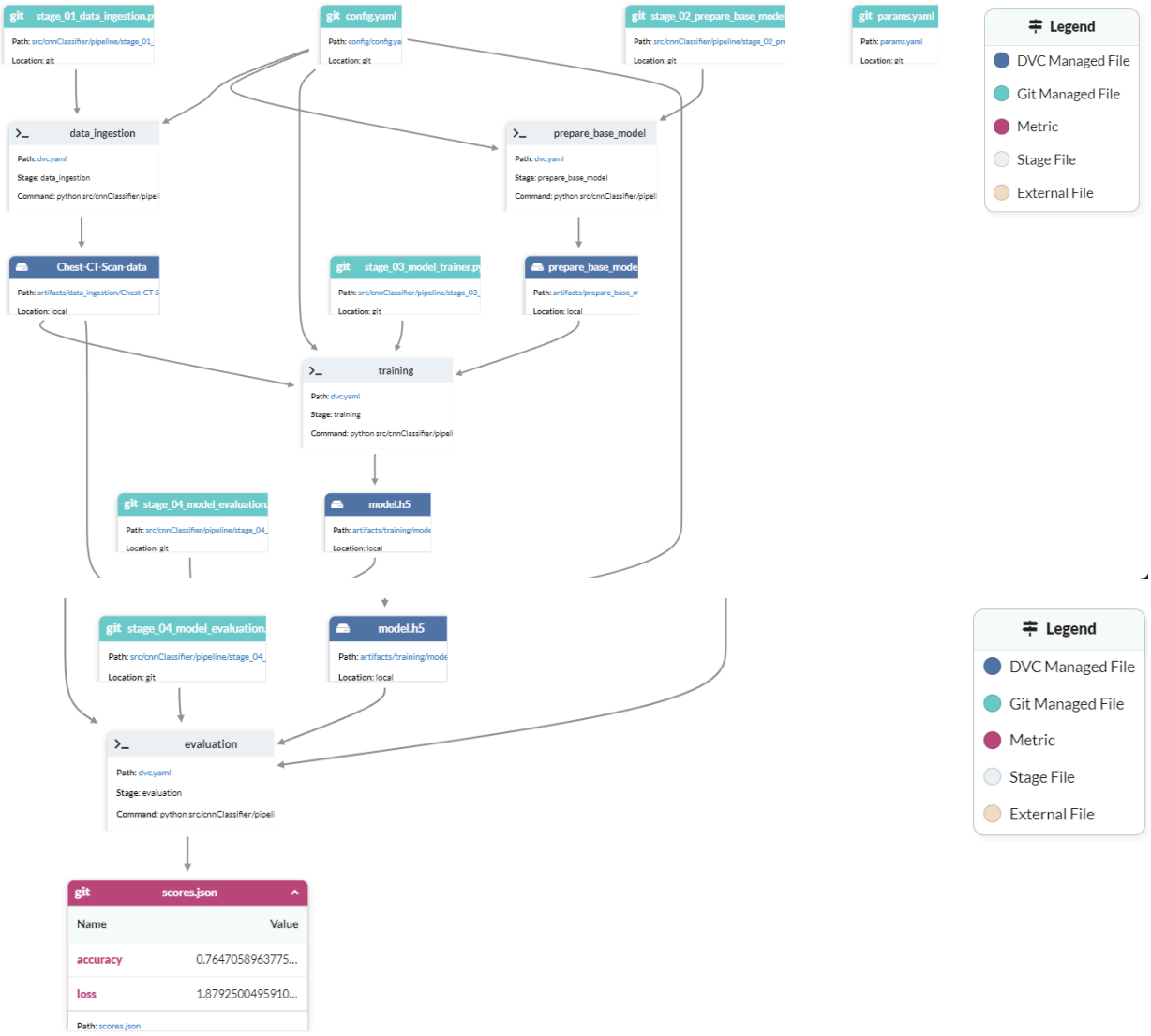
Tools-

- VSCode
- Github
- MLFlow
- DVC
- Gitlab
- Dagshub
- Docker
- AWS
- HTML,CSS,Bootstrap

System Architecture Diagram



Data Flow Diagram



System Configuration for MLflow

We use DagsHub which can integrate with MLflow to provide an easy way to track experiment parameters and metrics, and provides a built-in integration with Git and dataset management meaning that all our experiments become fully reproducible.

Environment Variables -

- os.environ["MLFLOW_TRACKING_URI"]="https://dagshub.com/SuhasMeda/End-to-End-Chest-Cancer-Classification-using-MLflow-DVC.mlflow"
- os.environ["MLFLOW_TRACKING_USERNAME"]="*****"
- os.environ["MLFLOW_TRACKING_PASSWORD"]="*****"

System Configuration Source Code -

```
import tensorflow as tf
model = tf.keras.models.load_model("artifacts/training/model.h5")

from dataclasses import dataclass
from pathlib import Path


@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

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

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_evaluation_config(self) -> EvaluationConfig:
```

```
        eval_config = EvaluationConfig(
            path_of_model="artifacts/training/model.h5",
            training_data="artifacts/data_ingestion/Chest-CT-Scan-data",

mlflow_uri="https://dagshub.com/SuhasMeda/End-to-End-Chest-Cancer-Classification-using-MLflow-DVC.mlfl
ow",
            all_params=self.params,
            params_image_size=self.params.IMAGE_SIZE,
            params_batch_size=self.params.BATCH_SIZE
        )
        return eval_config

import tensorflow as tf
from pathlib import Path
import mlflow
import mlflow.keras
from urllib.parse import urlparse


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


    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 = 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":
            mlflow.keras.log_model(self.model, "model", registered_model_name="VGG16Model")
        else:
            mlflow.keras.log_model(self.model, "model")
```

```
try:
    config = ConfigurationManager()
    eval_config = config.get_evaluation_config()
    evaluation = Evaluation(eval_config)
    evaluation.evaluation()
    evaluation.log_into_mlflow()
except Exception as e:
   raise e
```

System Configuration for DVC

We use DVC for orchestrating our pipelines with the help of our “dvc.yaml” file.

DVC commands used in our project -

dvc init - a new . dvc/ directory is created for configuration, default cache location, and other internal files and directories, that are hidden from the user.

dvc repro - It provides an interface to rerun the commands in the pipeline defined by the stage files in the current workspace.

dvc dag - It is used to display the dependency graph of the stages in our pipeline, as defined in the dvc.yaml file found in the project.

dvc.yaml file -

```
stages:

  data_ingestion:
    cmd: python src/cnnClassifier/pipeline/stage_01_data_ingestion.py
    deps:
      - src/cnnClassifier/pipeline/stage_01_data_ingestion.py
      - config/config.yaml
```

```
  outs:
    - artifacts/data_ingestion/Chest-CT-Scan-data


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_trainer.py
  deps:
    - src/cnnClassifier/pipeline/stage_03_model_trainer.py
    - config/config.yaml
    - artifacts/data_ingestion/Chest-CT-Scan-data
    - 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/Chest-CT-Scan-data
    - artifacts/training/model.h5
  params:
    - IMAGE_SIZE
    - BATCH_SIZE
  metrics:
  - scores.json:
      cache: false
```

Pre-requirements for Auto Training for AI/ML

We use code for converting image to Base64 format and vice versa for getting the final output.

Workflow -

1. Update config.yaml
2. Update params.yaml
3. Update the entity
4. Update the configuration manager in src config
5. Update the components
6. Update the pipeline
7. Update the main.py
8. Update the dvc.yaml

```
from dataclasses import dataclass
from pathlib import Path


@dataclass(frozen=True)
class TrainingConfig:
    root_dir: Path
    trained_model_path: Path
    updated_base_model_path: Path
```

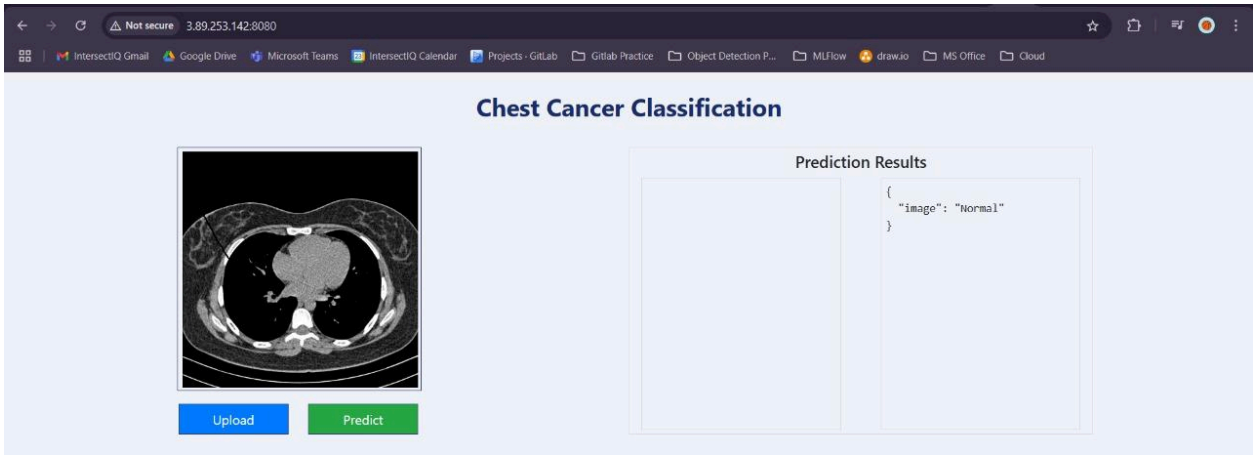
```
    training_data: Path
    params_epochs: int
    params_batch_size: int
    params_is_augmentation: bool
    params_image_size: list


from cnnClassifier.constants import *
from cnnClassifier.utils.common import read_yaml, create_directories
import tensorflow as tf

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_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, "Chest-CT-Scan-data")
        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

import os
import urllib.request as request
from zipfile import ZipFile
import tensorflow as tf
import time

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(
```

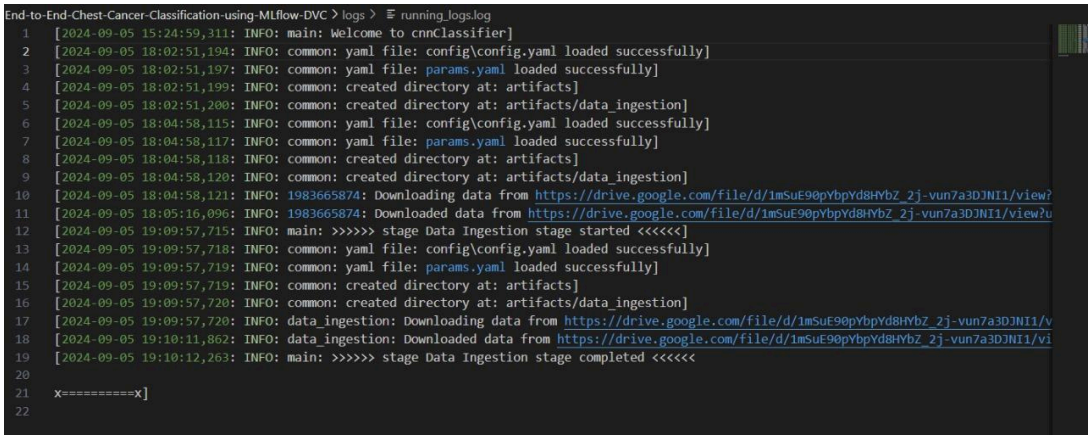
```
        directory=self.config.training_data,
        subset="validation",
        shuffle=False,
        **dataflow_kwargs
    )

    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,
```

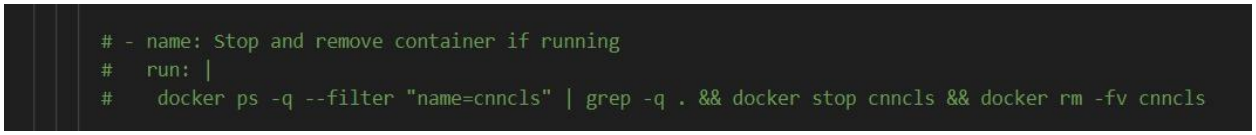
```
            epochs=self.config.params_epochs,
            steps_per_epoch=self.steps_per_epoch,
            validation_steps=self.validation_steps,
            validation_data=self.valid_generator
        )

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


try:
    config = ConfigurationManager()
    training_config = config.get_training_config()
    training = Training(config=training_config)
    training.get_base_model()
    training.train_valid_generator()
    training.train()

except Exception as e:
    raise e
```

Output

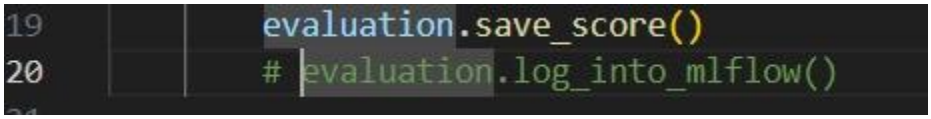


Screenshots

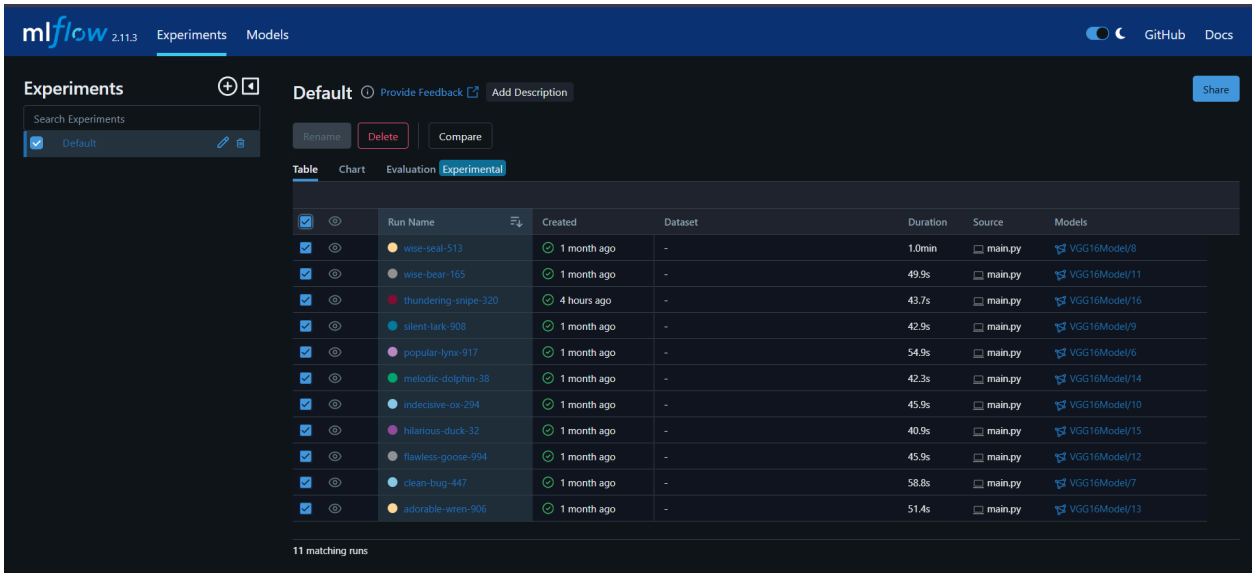


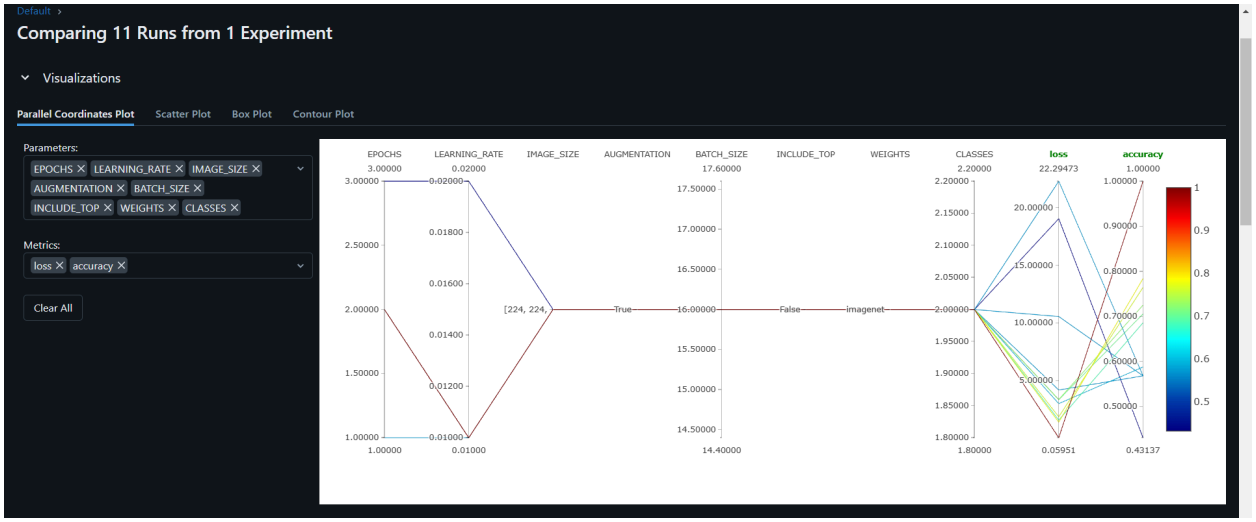


uncomment 2nd time onwards in the main.yaml file when executing cancer project

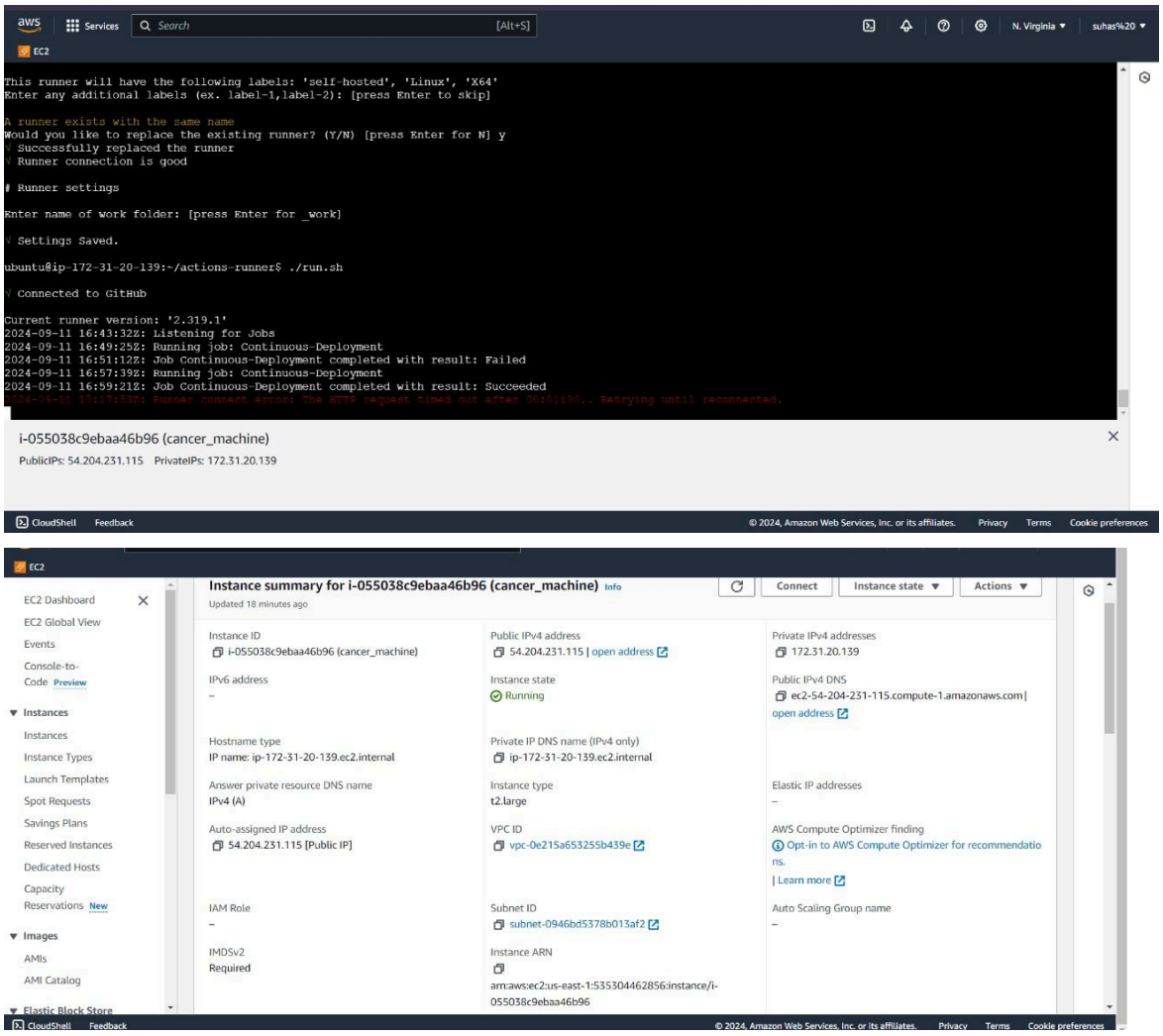


uncomment in stage_04_model_evaluation.py file to get logs in MLflow

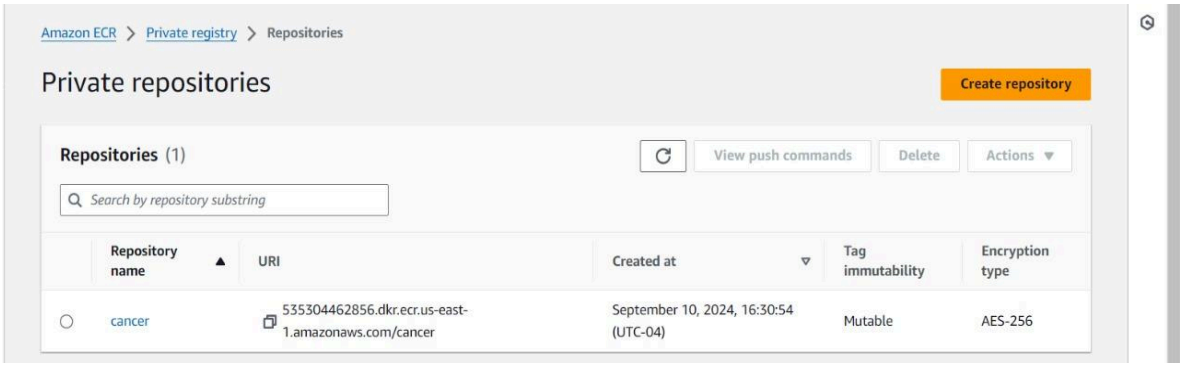


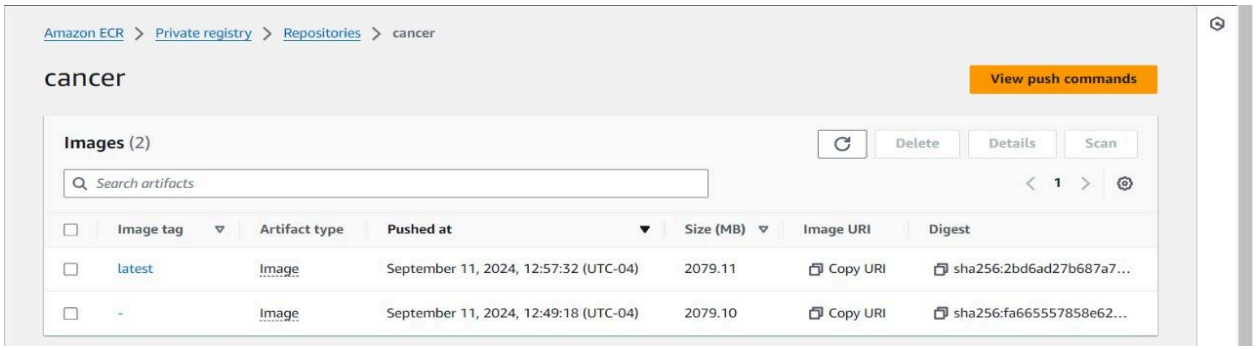


MLflow comparison of experiments

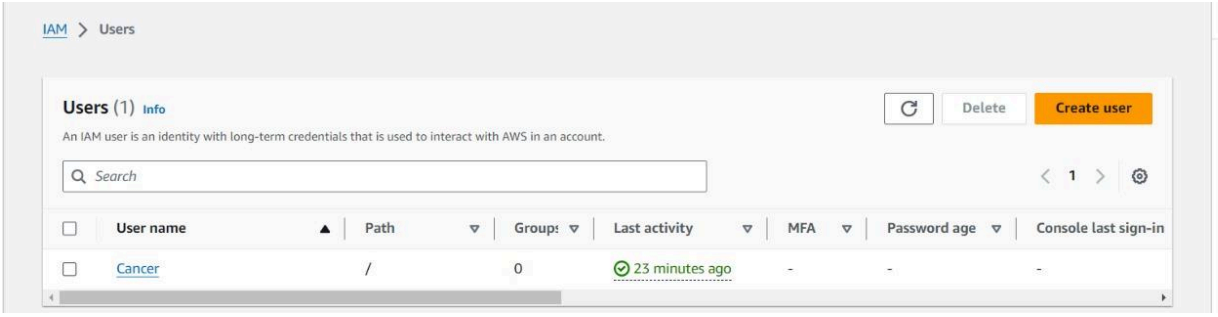


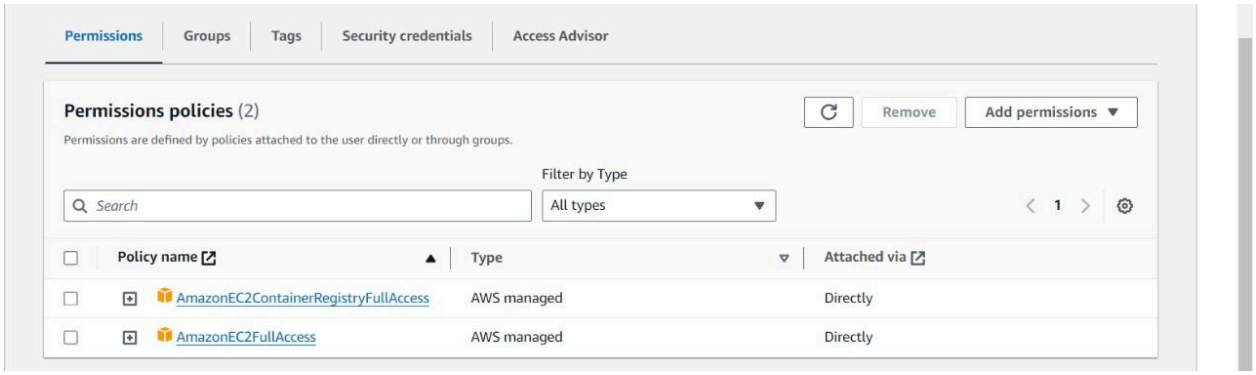
Ec2 Instance in AWS





ECR Registry in AWS to store the docker images





IAM User who has the policies required to access the EC2 instance in AWS.

Future Scope of Enhancement

- Use the Data version control (DVC) to enable rollbacking of models in MLFlow
- Replace Dagshub with a MLFlow Server inside nova_dev server(on-premise remote server)
- Enable dataset traceability through model weight files
- Pipeline Deployment capability to a server or edge machine (done)
- Parameterize the entire code through a .yaml file or a .env file

References

- https://dvc.org/doc
- https://mlflow.org/docs/latest/index.html
- https://dagshub.com/docs/
- https://www.tensorflow.org/guide/keras
- https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918
- https://www.youtube.com/@dswithbappy
- https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/adding-self-hosted-runners
- https://base64.guru/converter/decode/image