

Chest Cancer Classification using MLflow-DVC-AWS

Index

1)Abstract

2)Objective of the Project

3)Tools and Requirements

4)System Architecture Diagram

5)Data Flow Diagram

6)System Configuration for MLFlow

7)System Configuration for DVC

8)Pre-requirements for Auto Training for AI/ML

9)Output

10)Screenshots

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

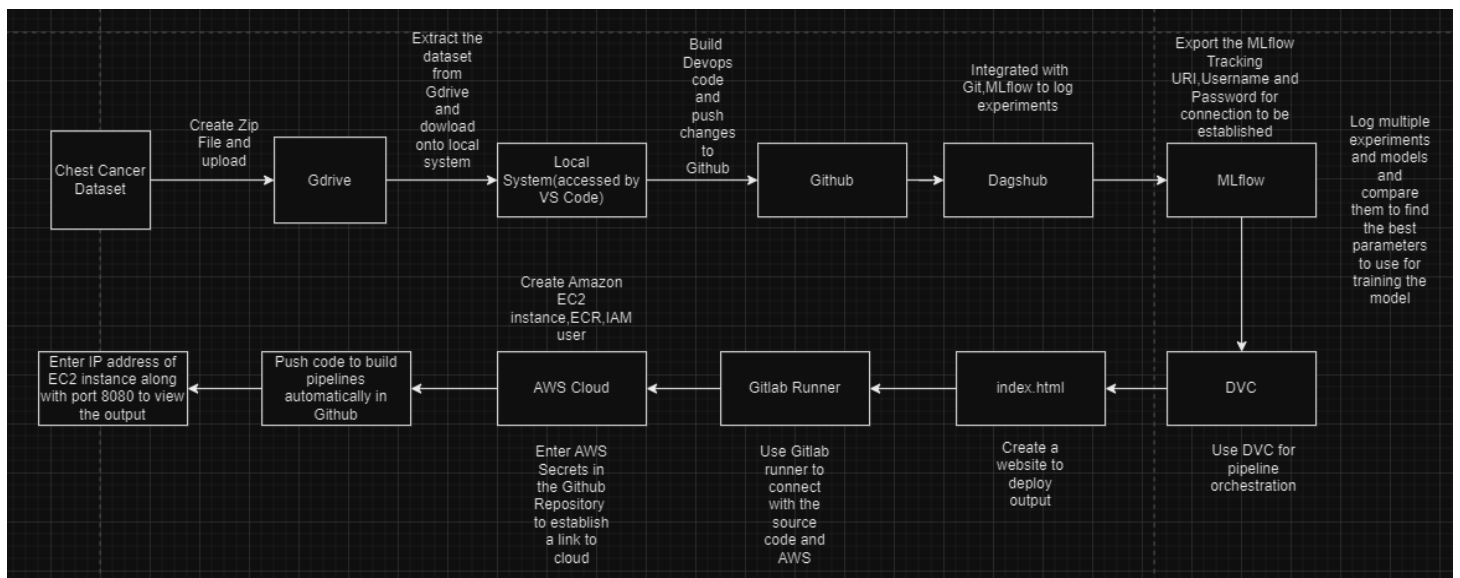
Requirements-

- 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 .

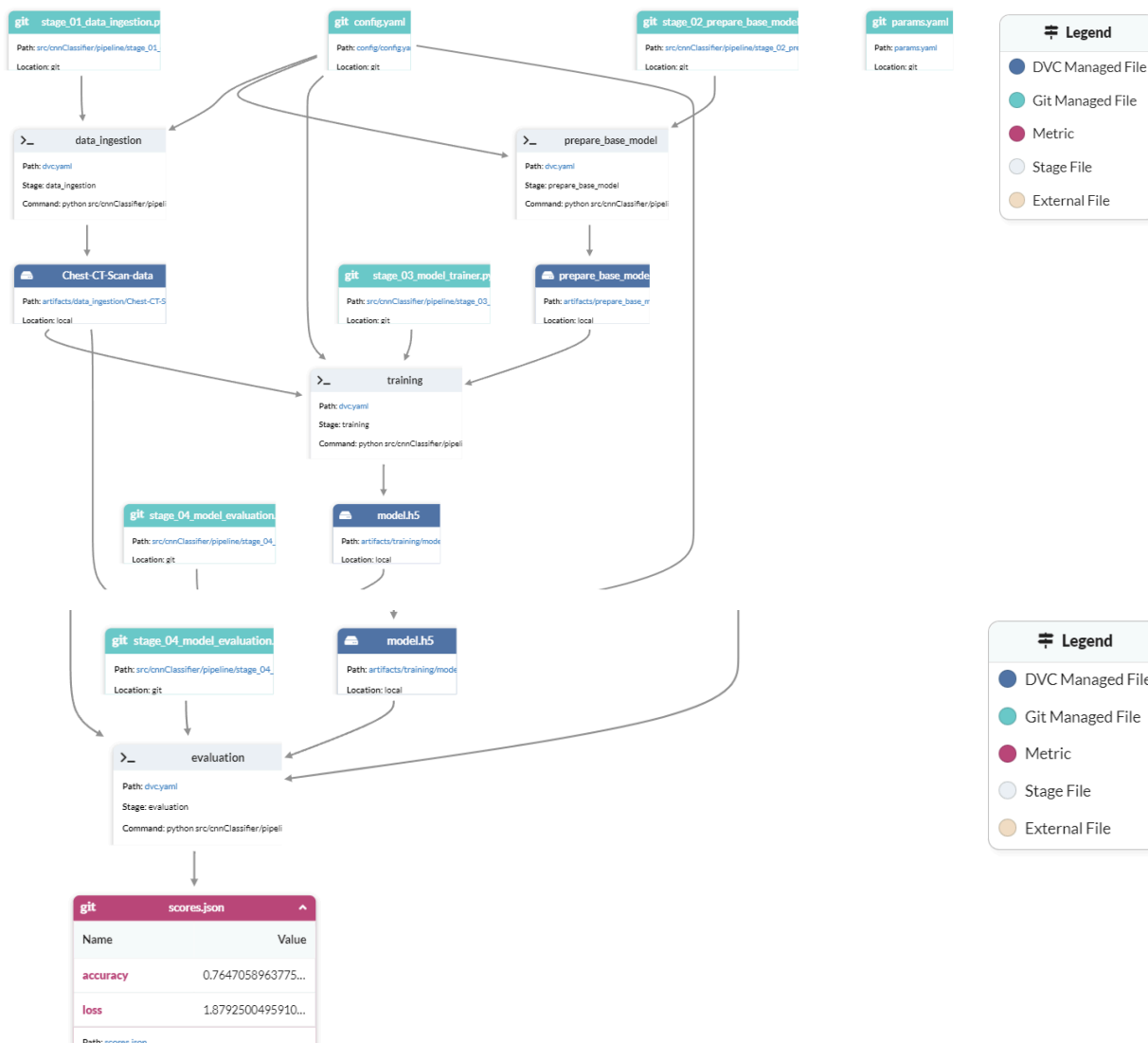
Tools-

- 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.mlflow",
    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  
)
```

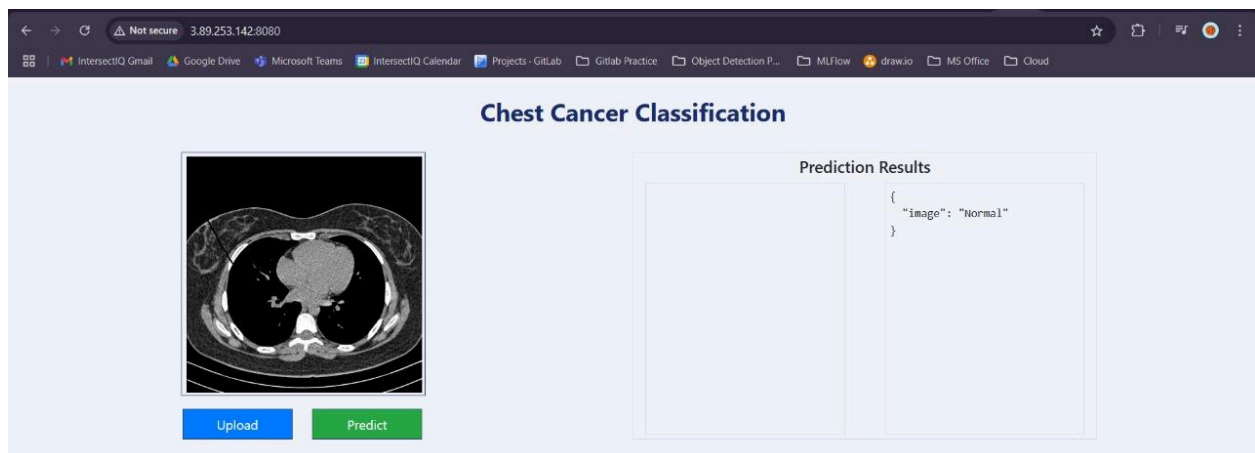
```
self.save_model(  
    path=self.config.trained_model_path,  
    model=self.model  
)
```

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

```
End-to-End-Chest-Cancer-Classification-using-MLflow-DVC > logs > E running logs.log
1 [2024-09-05 15:24:59,311: INFO: main: Welcome to cnnClassifier]
2 [2024-09-05 18:02:51,194: INFO: common: yaml file: config\config.yaml loaded successfully]
3 [2024-09-05 18:02:51,197: INFO: common: yaml file: params.yaml loaded successfully]
4 [2024-09-05 18:02:51,199: INFO: common: created directory at: artifacts]
5 [2024-09-05 18:02:51,200: INFO: common: created directory at: artifacts\data_ingestion]
6 [2024-09-05 18:04:58,115: INFO: common: yaml file: config\config.yaml loaded successfully]
7 [2024-09-05 18:04:58,117: INFO: common: yaml file: params.yaml loaded successfully]
8 [2024-09-05 18:04:58,118: INFO: common: created directory at: artifacts]
9 [2024-09-05 18:04:58,120: INFO: common: created directory at: artifacts\data_ingestion]
10 [2024-09-05 18:04:58,121: INFO: 1983665874: Downloading data from https://drive.google.com/file/d/1mSuE90pYbpYd8HybZ_2j-vun7a3DJN11/view?
11 [2024-09-05 18:05:16,096: INFO: 1983665874: Downloaded data from https://drive.google.com/file/d/1mSuE90pYbpYd8HybZ_2j-vun7a3DJN11/view?u
12 [2024-09-05 19:09:57,715: INFO: main: >>>>> stage Data Ingestion stage started <<<<<]
13 [2024-09-05 19:09:57,718: INFO: common: yaml file: config\config.yaml loaded successfully]
14 [2024-09-05 19:09:57,719: INFO: common: yaml file: params.yaml loaded successfully]
15 [2024-09-05 19:09:57,719: INFO: common: created directory at: artifacts]
16 [2024-09-05 19:09:57,720: INFO: common: created directory at: artifacts\data_ingestion]
17 [2024-09-05 19:09:57,720: INFO: data_ingestion: Downloading data from https://drive.google.com/file/d/1mSuE90pYbpYd8HybZ_2j-vun7a3DJN11/v
18 [2024-09-05 19:10:11,862: INFO: data_ingestion: Downloaded data from https://drive.google.com/file/d/1mSuE90pYbpYd8HybZ_2j-vun7a3DJN11/vi
19 [2024-09-05 19:10:12,263: INFO: main: >>>>> stage Data Ingestion stage completed <<<<<]
20
21 x=====x]
22
```

```
# - name: Stop and remove container if running
# run: |
#     docker ps -q --filter "name=cnncls" | grep -q . && docker stop cnncls && docker rm -fv cnncls
```

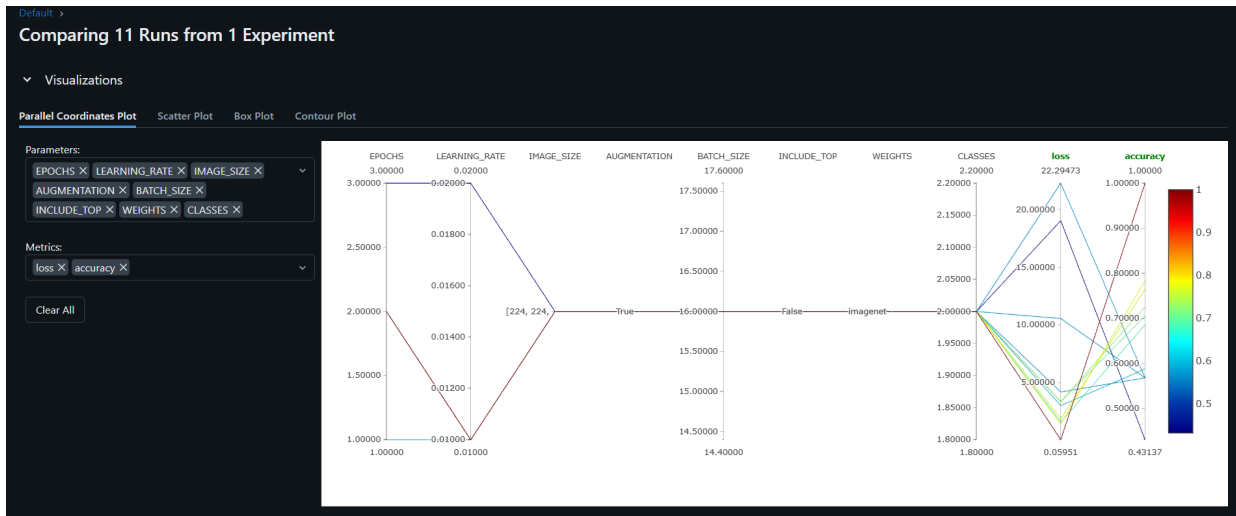
uncomment 2nd time onwards in the main.yaml file when executing cancer project

```
19         evaluation.save_score()
20         # evaluation.log_into_mlflow()
```

uncomment in stage_04_model_evaluation.py file to get logs in MLflow

The screenshot shows the MLflow Experiments interface. The 'Experiments' tab is active, displaying a table of runs. The table has columns for Run Name, Created, Dataset, Duration, Source, and Models. There are 11 runs listed, each with a unique name and a duration. The 'Models' column shows the model name for each run.

Run Name	Created	Dataset	Duration	Source	Models
wise-seal-513	1 month ago	-	1.0min	main.py	VGG16Model/8
wise-bear-165	1 month ago	-	49.9s	main.py	VGG16Model/11
thundering-snipe-320	4 hours ago	-	43.7s	main.py	VGG16Model/16
silent-lark-908	1 month ago	-	42.9s	main.py	VGG16Model/9
popular-lynx-917	1 month ago	-	54.9s	main.py	VGG16Model/6
melodic-dolphin-38	1 month ago	-	42.3s	main.py	VGG16Model/14
indecisive-ox-294	1 month ago	-	45.9s	main.py	VGG16Model/10
hilarious-duck-32	1 month ago	-	40.9s	main.py	VGG16Model/15
flawless-geese-994	1 month ago	-	45.9s	main.py	VGG16Model/12
clean-bug-447	1 month ago	-	58.8s	main.py	VGG16Model/7
adorable-wren-906	1 month ago	-	51.4s	main.py	VGG16Model/13



MLflow comparison of experiments

```

AWS
Services
Search
[Alt+S]
N. Virginia
suhas%20

EC2

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

A runner exists with the same name
Would you like to replace the existing runner? (Y/N) [press Enter for N] y
✓ Successfully replaced the runner
✓ Runner connection is good

# Runner settings
Enter name of work folder: [press Enter for _work]
✓ Settings Saved.

ubuntu@ip-172-31-20-139:~/actions-runner$ ./run.sh
✓ Connected to GitHub

Current runner version: '2.319.1'
2024-09-11 16:43:32: Listening for Jobs
2024-09-11 16:49:25: Running job: Continuous-Deployment
2024-09-11 16:51:22: Job Continuous-Deployment completed with result: Failed
2024-09-11 16:57:39: Running job: Continuous-Deployment
2024-09-11 16:59:21: Job Continuous-Deployment completed with result: Succeeded
2024-09-11 17:17:52: Runner connect error: The GETS request timed out after 300(110)... Retrying until reconnected.

i-055038c9ebaa46b96 (cancer_machine)
PublicIPs: 54.204.231.115 PrivateIPs: 172.31.20.139
  
```

EC2

Instance summary for i-055038c9ebaa46b96 (cancer_machine) info

Updated 18 minutes ago

Connect Instance state Actions

Instance ID i-055038c9ebaa46b96 (cancer_machine)	Public IPv4 address 54.204.231.115 open address	Private IPv4 addresses 172.31.20.139
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-54-204-231-115.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-20-139.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-20-139.ec2.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A) 54.204.231.115 (Public IP)	Instance type t2.large	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendation rs. Learn more
Auto-assigned IP address 54.204.231.115 (Public IP)	VPC ID vpc-0e215a653255b439e	Auto Scaling Group name -
IAM Role -	Subnet ID subnet-0946bd5378b013af2	
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:535304462856:instance/i-055038c9ebaa46b96	

EC2 Dashboard EC2 Global View Events Console-to-Code Preview

Instances

- Instances
- Instance Types
- Launch Templates
- Spot Requests
- Savings Plans
- Reserved Instances
- Dedicated Hosts
- Capacity
- Reservations [New](#)

Images

- AMIs
- AMI Catalog

Elastic Block Store

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Ec2 Instance in AWS

Amazon ECR > Private registry > Repositories

Private repositories

Create repository

Repositories (1)

Search by repository substring

Repository name	URI	Created at	Tag immutability	Encryption type
cancer	535304462856.dkr.ecr.us-east-1.amazonaws.com/cancer	September 10, 2024, 16:30:54 (UTC-04)	Mutable	AES-256

Amazon ECR > Private registry > Repositories > cancer

cancer

View push commands

Images (2)

Search artifacts

Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest
latest	Image	September 11, 2024, 12:57:32 (UTC-04)	2079.11	Copy URI	sha256:2bd6ad27b687a7...
-	Image	September 11, 2024, 12:49:18 (UTC-04)	2079.10	Copy URI	sha256:fa665557858e62...

ECR Registry in AWS to store the docker images

IAM > Users

Users (1)

An IAM user is an identity with long-term credentials that is used to interact with AWS in an account.

Search

User name	Path	Group	Last activity	MFA	Password age	Console last sign-in
Cancer	/	0	23 minutes ago	-	-	-

Permissions | Groups | Tags | Security credentials | Access Advisor

Permissions policies (2)

Permissions are defined by policies attached to the user directly or through groups.

Search

Filter by Type: All types

Policy name	Type	Attached via
AmazonEC2ContainerRegistryFullAccess	AWS managed	Directly
AmazonEC2FullAccess	AWS managed	Directly

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>