Cats vs Dogs Classification: An End-to-End MLOps Pipeline

S.No	Register Number	Name	Contribution
1	2024aa05421	Krithika Madhavan	100%
2	2024aa05435	Yarragondla Rugmangadha Reddy	100%
3	2024aa05423	Payel Karmakar	100%
4	2024aa05870	Deepak Sindhu	100%
5	2024ab05227	PARAB PRATHAMESH PRAFULLA PRADNYA	100%

1. Project Overview

This project implements a **complete end-to-end MLOps pipeline** for binary image classification (Cats vs Dogs) designed for a pet adoption platform.

The goal is to:

- Build a reproducible ML training pipeline
- Track experiments using MLflow
- Containerize the inference service
- Implement CI/CD automation
- Deploy and monitor the model
- Perform post-deployment performance validation

The dataset used is the **Kaggle Cats vs Dogs Dataset**, containing labeled RGB images of cats and dogs.

All images are:

- Resized to 224×224
- Converted to RGB
- Split into train/validation/test sets (80/10/10)
- Augmented for better generalization

This project mirrors real-world ML production workflows by integrating:

- Git + DVC for versioning
- MLflow for experiment tracking
- FastAPI for model serving
- Docker for containerization
- GitHub Actions for CI/CD
- Monitoring and post-deployment evaluation

2. Setup & Installation Instructions

2.1 System Requirements

- Python 3.10+
- pip
- Docker
- Git
- GitHub account

2.2 Clone Repository

```
git clone https://github.com/Krithika-Madhavan-5421/MLOPS-ASSIGNMENT2-GROUP44.git
cd cats-dogs-mlops
```

2.3 Create Virtual Environment

```
conda create -n cats-dogs-mlops python=3.10 -y
conda activate cats-dogs-mlops
```

or

```
python -m venv venv
source venv/bin/activate
```

2.4 Install Dependencies

```
pip install --upgrade pip
pip install -r requirements.txt
```

---

3. Data Preparation

Run:

```
python src/data_preparation.py
```

This step:

- Loads raw Kaggle dataset
- Removes corrupt images
- Resizes images to 224×224
- Splits into train/val/test
- Saves processed dataset under `data/processed/`

Dataset versioning is managed using **DVC**.

---

4. Model Development & Experiment Tracking (M1)

4.1 Baseline Model

A Simple CNN was implemented as the baseline model:

- 3 convolutional blocks
- Batch Normalization
- ReLU activation
- MaxPooling
- Global Average Pooling
- Fully connected classifier

Binary classification using:

```
BCEWithLogitsLoss
```

4.2 Training

Run:

```
python src/train.py
```

Training includes:

- Data augmentation
- Weight decay regularization
- Learning rate scheduler
- MLflow logging

4.3 MLflow Experiment Tracking

MLflow logs:

- Parameters
- Training loss
- Validation accuracy
- Confusion matrix
- Serialized model
- PyTorch model artifact

Launch MLflow UI:

```
mlflow ui
```

Access:

```
http://localhost:5000
```

---

5. Model Packaging & Containerization (M2)

5.1 FastAPI Inference Service

Endpoints:

- `GET /health`
- `POST /predict`
- `GET /metrics`

Run locally:

```
uvicorn src.inference_app:app --host 0.0.0.0 --port 8000
```

Swagger UI:

```
http://localhost:8000/docs
```

5.2 Dockerization

Build image:

```
docker build -t cats-dogs-inference:latest .
```

Run container:

```
docker run -p 8000:8000 cats-dogs-inference:latest
```

Image is pushed to Docker Hub via CI.

---

6. CI Pipeline (M3)

Implemented using **GitHub Actions**.

Triggered on push.

Pipeline steps:

1. Checkout code
2. Install dependencies
3. Run unit tests (pytest)
4. Build Docker image
5. Push image to Docker Hub

Unit tests include:

- Preprocessing function test
- Inference utility test

---

7. CD Pipeline & Deployment (M4)

Deployment uses Docker Compose.

On successful CI:

- Pull latest image
- Start container
- Run smoke tests:
  - `/health`
  - `/predict`
- Fail pipeline if smoke test fails

---

8. Monitoring & Post-Deployment Evaluation (M5)

8.1 Logging & Metrics

Inference service logs:

- Request count
- Prediction probability
- Predicted label
- Latency

Metrics endpoint:

```
GET /metrics
```

Returns:

- Total requests
- Average latency

8.2 Post-Deployment Evaluation

Separate workflow performs:

- Calls `/predict` with labeled test images
- Computes accuracy
- Enforces threshold
- Fails workflow if below threshold

Monitoring pipeline now reports:

```
Post-deployment accuracy: 1.0
```

---

9. Architecture Overview

```
Raw Data
   ↓
Data Preparation (DVC)
   ↓
Training (MLflow)
   ↓
Model Artifact
   ↓
```

```
Docker Build (CI)
   ↓
Docker Hub
   ↓
CD Deploy (Docker Compose)
   ↓
FastAPI Service
   ↓
Monitoring & Evaluation
```

---

10. CI/CD Workflow Summary

CI:

- Automated testing
- Docker build
- Image push

CD:

- Auto deployment
- Smoke testing

Monitoring:

- Request logs
- Latency tracking
- Accuracy validation

---

11. Key Design Decisions

- Global Average Pooling to reduce model size
- Models not stored in Git (size compliance)
- Lazy model loading in inference
- Architecture consistency between train and inference
- Separation of deployment and monitoring pipelines

---

12. Code Repository

🔗 https://github.com/Krithika-Madhavan-5421/MLOPS-ASSIGNMENT2-GROUP44

---

13. Conclusion

This project demonstrates a complete industrial-grade MLOps pipeline:

- Reproducible training
- Experiment tracking
- Containerized inference
- Automated CI/CD
- Post-deployment monitoring
- Performance validation

The pipeline follows production best practices and ensures:

- Scalability
- Reproducibility
- Automation
- Deployment readiness

It mirrors real-world ML system design used in production environments.
