
Cats vs Dogs Classification

End-to-End Production-Grade MLOps Pipeline

Team Details

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 production-ready MLOps pipeline** for binary image classification (Cats vs Dogs).

The pipeline demonstrates:

- Reproducible data versioning (DVC)
- Experiment tracking (MLflow)
- Model training and artifact logging
- Containerized inference (FastAPI + Docker)
- CI/CD automation (GitHub Actions)
- Deployment validation
- Monitoring & post-deployment evaluation

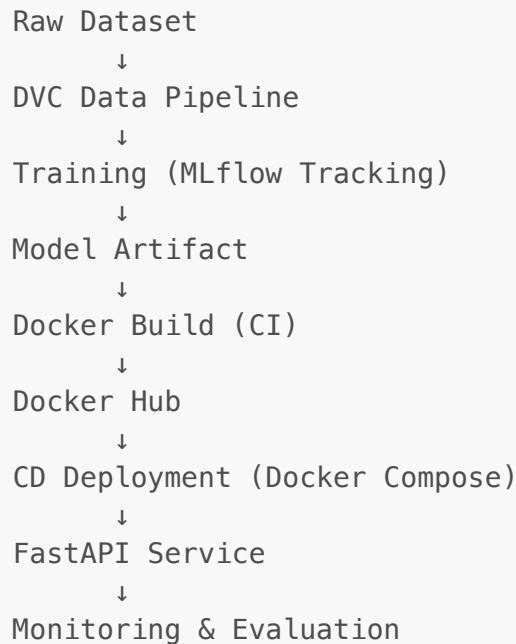
Dataset:

Kaggle Cats vs Dogs Dataset

Images are:

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

2. System Architecture



3. Setup Instructions

3.1 Requirements

- Python 3.10+
- Git
- Docker
- DVC
- MLflow

3.2 Clone Repository

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

3.3 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
```

3.4 Install Dependencies

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

4. Data Versioning & Reproducibility (DVC)

4.1 DVC Pipeline Structure

The project defines reproducible stages in `dvc.yaml`:

```
stages:  
  data_preparation:  
    cmd: python src/data_preparation.py  
    deps:  
      - src/data_preparation.py  
      - data/raw  
    outs:  
      - data/processed  
  
  train:  
    cmd: python src/train.py  
    deps:  
      - src/train.py  
      - data/processed  
    outs:  
      - models/model.pth
```

4.2 Running Full Pipeline

To reproduce the entire pipeline:

```
dvc repro
```

This will:

- Run data preprocessing

- Train model
 - Regenerate artifacts if dependencies changed
-

4.3 Pulling Artifacts from Remote

If DVC remote is configured:

```
dvc pull
```

Restores:

- Processed dataset
 - Model weights
-

Why DVC?

- Large data not stored in Git
 - Reproducible training
 - Controlled dataset versioning
 - Automatic dependency tracking
-

5. Model Development & Experiment Tracking

5.1 Baseline CNN Model

Architecture:

- 3 Conv blocks
- Batch Normalization
- ReLU
- MaxPooling
- Global Average Pooling
- Fully connected output

Loss Function:

```
BCEWithLogitsLoss
```

5.2 Training

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

Includes:

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

5.3 MLflow Tracking

MLflow logs:

- Hyperparameters
- Loss curves
- Validation accuracy
- Confusion matrix
- Model artifacts

Launch UI:

```
mlflow ui
```

Access:

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

The screenshot shows the mlflow UI interface. At the top, there's a navigation bar with 'Experiments' selected. Below it, the 'Experiments' section lists a single experiment named 'cats_vs_dogs_experiment'. A table below shows one run named 'wistful-fly-803' with details like duration (22.1min), source (train.py), and model (pytorch). The 'Experimental' tab is selected in the table header. Below the table, it says '1 matching run'. The bottom half of the screenshot shows the detailed view for the 'wistful-fly-803' run, including sections for Overview, Parameters (5), and Metrics (2).

Experiments

cats_vs_dogs_experiment

Search Experiments

Default

cats_vs_dogs_experiment

Share

Provide Feedback

Add Description

Time created

State: Active

Datasets

Sort: Created

Columns

Group by

Table

Chart

Evaluation

Experimental

Run Name

Created

Dataset

Duration

Source

Models

wistful-fly-803

36 minutes ago

-

22.1min

train.py

pytorch

1 matching run

cats_vs_dogs_experiment > wistful-fly-803

Register model

Overview

Model metrics

System metrics

Artifacts

Duration: 22.1min

Datasets used: -

Tags: Add

Source: train.py → 2d0b7470733ef872d4a97b4ef3b92d91f80b15ca

Logged models: pytorch

Registered models: -

Parameters (5)

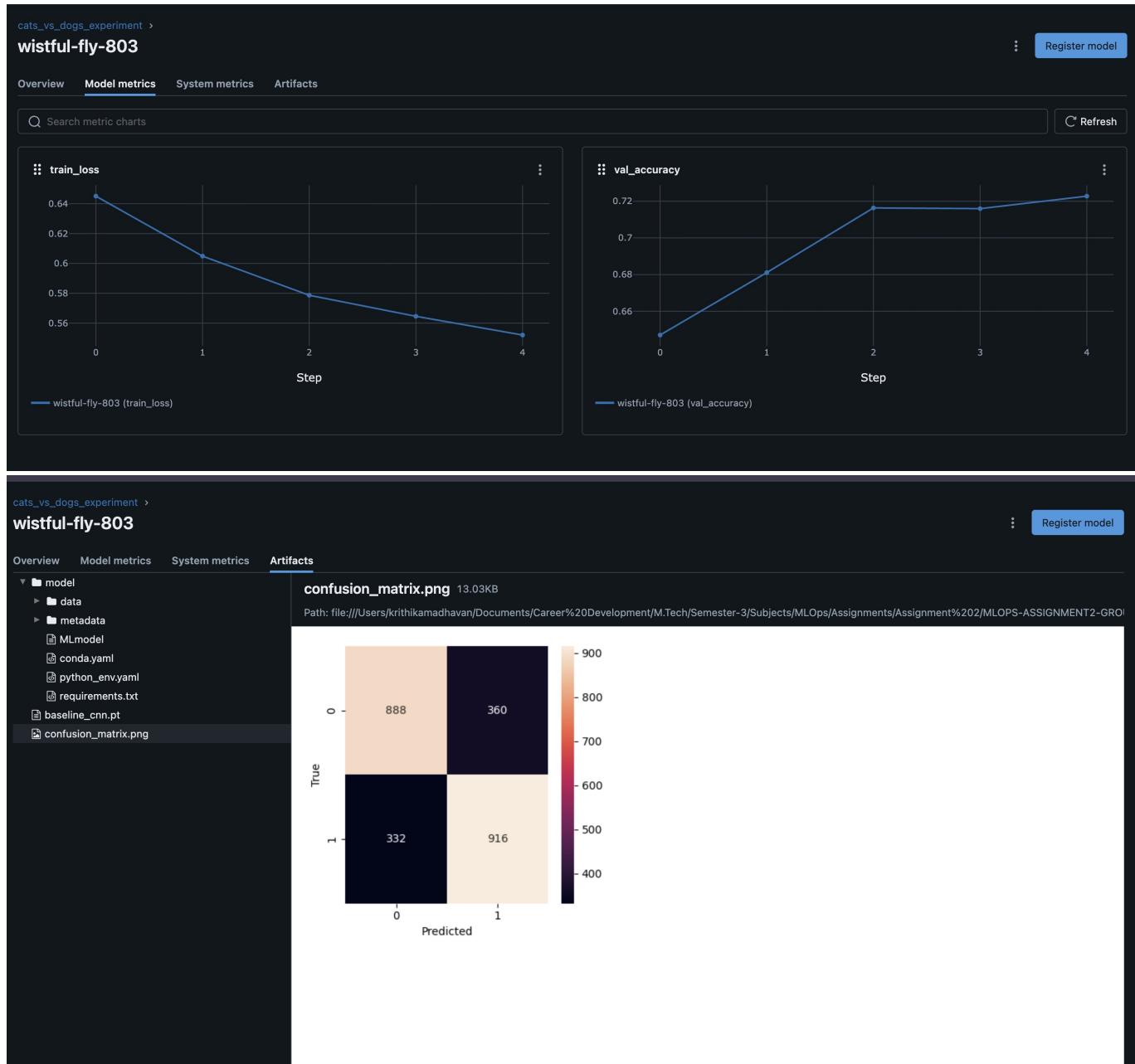
Search parameters

Parameter	Value
weight_decay	0.0001
epochs	5
batch_size	64
lr	0.0001
image_size	128

Metrics (2)

Search metrics

Metric	Value
train_loss	0.5520351620820853
val_accuracy	0.7227564102564102



6. Model Serving & Dockerization

6.1 FastAPI Inference Service

Endpoints:

Endpoint	Purpose
/health	Health check
/predict	Image classification
/metrics	Runtime metrics

Run locally:

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

Swagger:

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

The screenshot shows the Swagger UI for the 'Cats vs Dogs Inference API'. At the top, it displays the URL 'localhost:8000/docs' in a browser header. Below the header, the title 'Cats vs Dogs Inference API' is shown with a version '0.1.0' and an 'OAS 3.1' badge. A link to '/openapi.json' is also present. The main content area is titled 'default'. It lists three operations: 'GET /health' (Health), 'POST /predict' (Predict), and 'GET /metrics' (Metrics). Each operation has a collapse/expand arrow icon to its right. Below this, there is a section titled 'Schemas' which contains definitions for 'Body_predict_predict_post', 'HTTPValidationError', and 'ValidationError', each with an 'Expand all object' link.

6.2 Docker Build

```
docker build -t <dockerhub-username>/cats-dogs-inference:latest .
```

Run:

```
docker run -p 8000:8000 <dockerhub-username>/cats-dogs-inference:latest
```

7. Docker Hub Integration

Manual Push

```
docker login
docker push <dockerhub-username>/cats-dogs-inference:latest
```

GitHub Secrets Required

Add in repository settings:

- DOCKER_USERNAME
- DOCKER_PASSWORD

CI Docker Push Workflow

```

- name: Login to Docker Hub
  run: echo "${{ secrets.DOCKER_PASSWORD }}" | docker login -u "${{ secrets.DOCKER_USERNAME }}"
  --password-stdin

- name: Build Image
  run: docker build -t ${{ secrets.DOCKER_USERNAME }}/cats-dogs-inference:latest .

- name: Push Image
  run: docker push ${{ secrets.DOCKER_USERNAME }}/cats-dogs-inference:latest

```

8. Continuous Integration (CI)

Triggered on push.

Pipeline:

1. Install dependencies
2. Run pytest unit tests
3. Build Docker image
4. Push image to Docker Hub

The screenshot shows a GitHub Actions CI pipeline named "test-and-build". The pipeline has 12 steps listed in the log:

- > Set up job (2s)
- > Checkout code (1s)
- > Set up Python (0s)
- > Install dependencies (1m 20s)
- > Run unit tests (4s)
- > Log in to Docker Hub (1s)
- > Build Docker image (2m 34s)
- > Push Docker image (14s)
- > Post Log in to Docker Hub (1s)
- > Post Set up Python (0s)
- > Post Checkout code (0s)
- > Complete job (0s)

9. Continuous Deployment (CD)

Deployment via Docker Compose.

Example:

```
version: "3.8"
services:
  inference:
    image: <dockerhub-username>/cats-dogs-inference:latest
    ports:
      - "8000:8000"
    restart: always
```

Deploy:

```
docker compose pull
docker compose up -d
```

Smoke tests:

- /health
- /predict

Fail deployment if tests fail.



10. Monitoring & Post-Deployment Evaluation

Runtime Metrics

GET `/metrics` returns:

- Total requests
- Average latency
- Prediction count

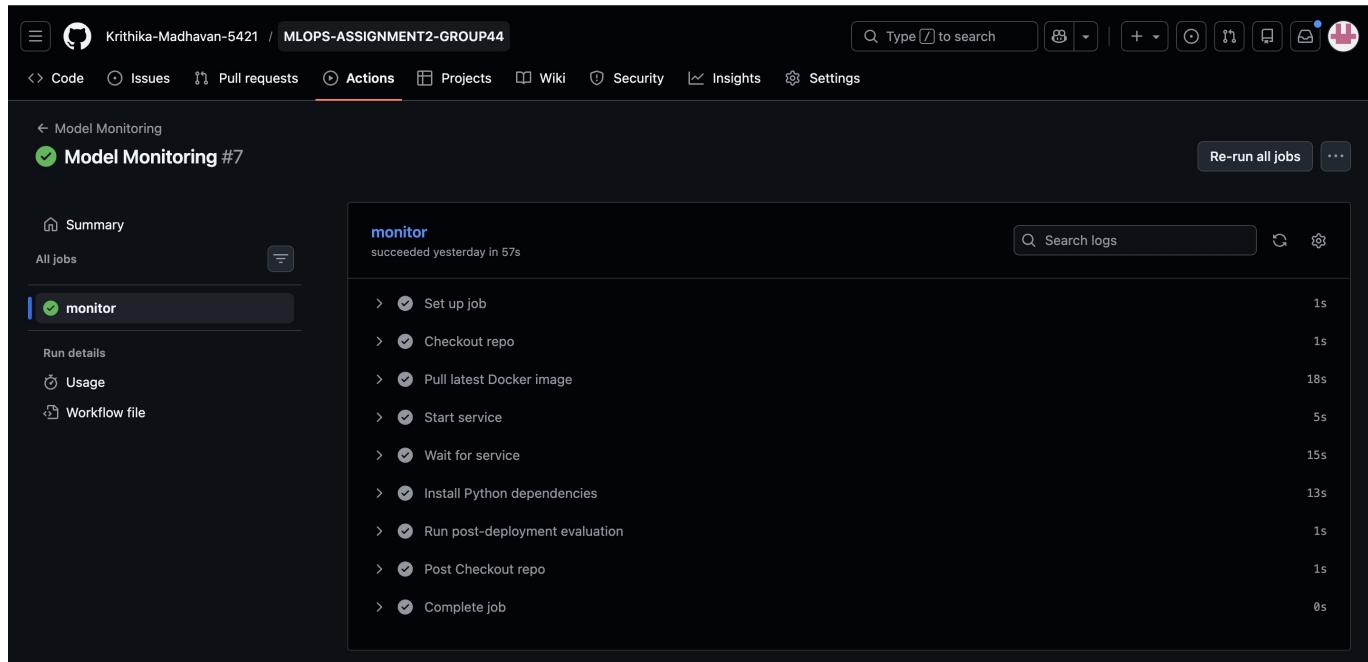
Post-Deployment Accuracy Validation

Separate workflow:

- Sends labeled test data
- Computes accuracy
- Enforces threshold
- Fails if below threshold

Example output:

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



The screenshot shows a GitHub Actions interface for a job named "monitor". The job status is "succeeded yesterday in 57s". The job details list the following steps:

Step	Description	Time
> ✓ Set up job		1s
> ✓ Checkout repo		1s
> ✓ Pull latest Docker image		18s
> ✓ Start service		5s
> ✓ Wait for service		15s
> ✓ Install Python dependencies		13s
> ✓ Run post-deployment evaluation		1s
> ✓ Post Checkout repo		1s
> ✓ Complete job		0s

11. Engineering Decisions

- Global Average Pooling reduces parameters
- DVC ensures reproducibility
- MLflow enables experiment tracking

- Docker ensures portability
 - CI/CD ensures automation
 - Lazy model loading for performance
 - Separate monitoring workflow
-

12. Repository

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

13. Conclusion

This project demonstrates a complete industrial-grade ML lifecycle including:

- Reproducible training
- Experiment management
- Containerized inference
- Automated CI/CD
- Monitoring & validation

It reflects production MLOps best practices and ensures:

- Scalability
 - Automation
 - Reliability
 - Deployment readiness
 - Performance validation
-