MLOps Pipeline Documentation

Complete Guide for Cats vs Dogs Classification MLOps Pipeline

This document provides comprehensive documentation for the end-to-end MLOps pipeline implementation for binary image classification (Cats vs Dogs) for a pet adoption platform.

Table of Contents

Architecture Overview

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                        MLOps Pipeline Architecture                          │
├─────────────────────────────────────────────────────────────────────────────┤
│                                                                             │
│  ┌──────────┐    ┌──────────┐    ┌──────────┐    ┌──────────┐               │
│  │   Git    │───▶│  GitHub  │───▶│  Docker  │───▶│   K8s    │               │
│  │   Repo   │    │  Actions │    │ Registry │    │ Cluster  │               │
│  └──────────┘    └──────────┘    └──────────┘    └──────────┘               │
│       │               │                               │                     │
│       ▼               ▼                               ▼                     │
│  ┌──────────┐    ┌──────────┐                   ┌──────────┐                │
│  │   DVC    │    │  Tests   │                   │  ArgoCD  │                │
│  │  (Data)  │    │  Build   │                   │ (GitOps) │                │
│  └──────────┘    └──────────┘                   └──────────┘                │
│
```

```
│       │                                               │
│       ▼                                               ▼
│  ┌──────────┐                                   ┌──────────┐
│  │  MLflow  │                                   │ Inference│
│  │ Tracking │                                   │ Service  │
│  └──────────┘                                   └──────────┘
│                                                       │
│                                                       ▼
│                                                 ┌──────────┐
│                                                 │Prometheus│
│                                                 │ Grafana  │
│                                                 └──────────┘
│
└──────────────────────────────────────────────────────────────────────
────┘
```

Milestone 1: Model Development & Experiment Tracking {#milestone-1}

1.1 Data & Code Versioning

Git for Source Code

```
# Initialize repository
git init
git add .
git commit -m "Initial commit"

# Create feature branch
git checkout -b feature/model-improvements
```

DVC for Data Versioning

```
# Initialize DVC
dvc init
```

```
# Add data to DVC tracking
dvc add data/raw
git add data/raw.dvc .gitignore
git commit -m "Add raw data tracking"

# Remote storage is configured to Dagshub
# URL:
https://dagshub.com/vishalvishal099/BinaryImageClassification_For_A_Pet_Adoption_Platform.dvc

# Push data to Dagshub
dvc push

# Pull data from Dagshub
dvc pull
```

1.2 Data Preprocessing

The preprocessing pipeline (`src/data/preprocess.py`) handles:

- **Image Loading**: Support for JPEG, PNG, and other formats
- **Resizing**: All images resized to 224×224 pixels
- **Color Space**: Convert to RGB (handles grayscale, RGBA)
- **Split**: 80% train / 10% validation / 10% test
- **Augmentation**: Horizontal flip, brightness/contrast, rotation, noise

```
# Run preprocessing
python src/data/preprocess.py --raw-dir data/raw --processed-dir data/processed
```

1.3 Model Building

Implemented models in `src/models/cnn.py`:

1. **SimpleCNN**: Baseline 4-layer CNN with batch normalization
2. **ImprovedCNN**: Enhanced CNN with residual-like connections
3. **ResNet18**: Transfer learning option with pretrained weights

```
from src.models.cnn import get_model

# Get baseline model
model = get_model("simple_cnn", num_classes=2, dropout_rate=0.5)

# Get improved model
model = get_model("improved_cnn", num_classes=2)
```

```
# Get pretrained ResNet
model = get_model("resnet18", pretrained=True)
```

1.4 Experiment Tracking with MLflow

```
# Train with MLflow tracking
python src/training/train.py --config configs/train_config.yaml

# Start MLflow tracking server (port 5001)
mlflow server --host 0.0.0.0 --port 5001 --backend-store-uri
sqlite:///mlflow.db
# UI: http://localhost:5001
# Dagshub remote:
https://dagshub.com/vishalvishal099/BinaryImageClassification_For_A_Pet_Ad
option_Platform.mlflow
```

Logged Artifacts:

- Model parameters (batch_size, learning_rate, epochs, etc.)
- Metrics (train/val loss, accuracy, precision, recall, F1)
- Confusion matrix
- Trained model file
- Training curves

Milestone 2: Model Packaging & Containerization {#milestone-2}

2.1 Inference Service (FastAPI)

The inference service (`src/inference/app.py`) provides:

Endpoint	Method	Description
`/`	GET	API information
`/health`	GET	Health check with model status
`/predict`	POST	Image classification
`/metrics`	GET	Prometheus metrics

```
# Example API response for /predict
{
    "predicted_class": "cat",
    "predicted_label": 0,
    "confidence": 0.95,
    "probabilities": {
        "cat": 0.95,
        "dog": 0.05
    },
    "processing_time_ms": 45.2
}
```

2.2 Dependencies (requirements.txt)

All dependencies are pinned for reproducibility:

- `torch==2.1.0`
- `fastapi==0.104.1`
- `mlflow==2.8.1`
- `prometheus-client==0.19.0`

2.3 Containerization

```
# Build image
docker build -t cats-dogs-classifier:latest .

# Run container
docker run -p 8000:8000 -v ./models:/app/models cats-dogs-classifier:latest

# Test with curl
curl http://localhost:8000/health

# Test prediction
curl -X POST -F "file=@test_image.jpg" http://localhost:8000/predict
```

Milestone 3: CI Pipeline {#milestone-3}

3.1 Automated Testing

Tests are located in `tests/`:

```
# Run all tests
pytest tests/ -v

# Run with coverage
pytest tests/ --cov=src --cov-report=html
```

Test Coverage:

- `test_preprocess.py`: Data preprocessing functions
- `test_inference.py`: API endpoints and inference logic

3.2 GitHub Actions CI Pipeline

The CI pipeline (`.github/workflows/ci.yml`) runs on every push/PR:

```
Jobs:
1. lint        - Code quality checks (black, isort, flake8)
2. test        - Unit tests with pytest
3. build       - Docker image build and push
4. integration - Docker container integration tests
5. security    - Trivy vulnerability scanning
```

3.3 Artifact Publishing

Docker images are pushed to GitHub Container Registry:

```
ghcr.io/your-username/cats-dogs-classifier:latest
ghcr.io/your-username/cats-dogs-classifier:<sha>
```

Milestone 4: CD Pipeline & Deployment {#milestone-4}

4.1 Kubernetes Deployment

Manifests in `k8s/`:

File	Purpose
`namespace.yaml`	Dedicated namespace
`configmap.yaml`	Environment configuration
`deployment.yaml`	Pod deployment with 2 replicas
`service.yaml`	LoadBalancer and ClusterIP services
`hpa.yaml`	Horizontal Pod Autoscaler

```
# Deploy to Kubernetes
kubectl apply -f k8s/

# Check deployment
kubectl get all -n cats-dogs-classifier
```

4.2 Docker Compose (Local)

```
# Start all services
docker-compose up -d
```

```
# With monitoring stack
docker-compose --profile monitoring up -d

# View logs
docker-compose logs -f classifier
```

4.3 GitOps with ArgoCD

```
# Install ArgoCD
kubectl create namespace argocd
kubectl apply -n argocd -f
https://raw.githubusercontent.com/argoproj/argo-
cd/stable/manifests/install.yaml

# Apply application
kubectl apply -f k8s/argocd-application.yaml
```

4.4 Smoke Tests

```
# Run smoke tests
python scripts/smoke_test.py --url http://localhost:8000 --wait

# Output:
# ✅ Health Check: PASS
# ✅ Root Endpoint: PASS
# ✅ Metrics Endpoint: PASS
# ✅ Prediction Endpoint: PASS
```

Milestone 5: Monitoring & Logging {#milestone-5}

5.1 Structured Logging

Request/response logging with structlog:

```
{
    "event": "prediction_completed",
    "predicted_class": "cat",
    "confidence": 0.95,
    "processing_time_ms": 45.2,
    "timestamp": "2024-01-15T10:30:00Z"
}
```

5.2 Prometheus Metrics

Available metrics at `/metrics`:

Metric	Type	Description
`prediction_requests_total`	Counter	Total predictions by class
`prediction_latency_seconds`	Histogram	Request latency distribution
`health_check_requests_total`	Counter	Health check requests
`prediction_errors_total`	Counter	Prediction errors by type

5.3 Model Performance Tracking

The `PerformanceTracker` class (`src/utils/performance_tracker.py`):

- Collects predictions with timestamps
- Accepts ground truth labels for feedback
- Computes accuracy, precision, recall, F1
- Tracks latency statistics (mean, p95, p99)
- Persists metrics to JSON files

```
from src.utils.performance_tracker import get_tracker

tracker = get_tracker()
tracker.log_prediction(
    prediction="cat",
    confidence=0.95,
    latency_ms=45.2,
    ground_truth="cat"  # Optional
)

stats = tracker.get_current_stats()
```

Workflow Demonstration {#workflow-demonstration}

Complete MLOps Workflow

```
# 1. Setup environment
make setup

# 2. Download and preprocess data
make download
make preprocess

# 3. Train model with MLflow tracking
make train

# 4. Run tests
```

```
make test

# 5. Build Docker image
make build

# 6. Run locally
make run

# 7. Test the API
make smoke-test

# 8. Deploy to Kubernetes
make deploy-k8s

# 9. Monitor with Prometheus
docker-compose --profile monitoring up -d
```

Code Change to Deployed Model

1. **Developer makes code change**

   ```
   git checkout -b feature/improve-model
   # ... make changes ...
   git add .
   git commit -m "Improve model accuracy"
   git push origin feature/improve-model
   ```

2. **CI Pipeline runs automatically**

   - Linting and code quality checks
   - Unit tests with pytest
   - Docker image build
   - Integration tests
   - Security scanning

3. **Merge to main triggers CD**

   - New Docker image pushed to registry
   - Kubernetes deployment updated
   - ArgoCD syncs changes
   - Smoke tests verify deployment

4. **Monitor deployed service**

   - Prometheus collects metrics
   - Grafana dashboards show performance
   - Logs available for debugging

Quick Reference

Important Commands

```
# Setup
make setup            # Full setup
source venv/bin/activate

# Development
make lint            # Run linters
make test            # Run tests
make train           # Train model

# Docker
make build           # Build image
make run             # Run container
make compose-up      # Start docker-compose

# Kubernetes
make deploy-k8s      # Deploy to K8s
make smoke-test      # Run smoke tests

# DVC
dvc pull             # Pull data
dvc push             # Push data
dvc repro            # Run pipeline
```

Environment Variables

Variable	Description	Default
`MODEL_PATH`	Path to model file	`/app/models/best_model.pt`
`PORT`	Service port	`8000`
`LOG_LEVEL`	Logging level	`INFO`

Troubleshooting

Common Issues

1. **Model not loaded**
   - Ensure model file exists at `MODEL_PATH`
   - Check model checkpoint format
2. **Docker build fails**
   - Verify requirements.txt is correct
   - Check Dockerfile stages

3. **Kubernetes deployment fails**
    - Check PVC is bound
    - Verify image pull secrets
4. **Tests failing**
    - Install test dependencies: `pip install pytest pytest-cov`
    - Run with verbose: `pytest -v`
