🚀 Setup Guide: Binary Image Classification for Pet Adoption Platform

This guide provides step-by-step instructions to set up and run the complete MLOps project locally.

📋 Table of Contents

Prerequisites

Required Software

Software	Version	Purpose	Installation
Python	3.10+	Core runtime	python.org
Git	2.30+	Version control	git-scm.com
Docker	20.0+	Containerization	docker.com
kubectl	1.28+	K8s management (optional)	kubernetes.io

Optional Software

Software	Purpose	Installation
Minikube/Kind	Local Kubernetes	minikube.sigs.k8s.io
Kaggle CLI	Dataset download	`pip install kaggle`
GitHub CLI	CI/CD management	cli.github.com

System Requirements

- **RAM**: Minimum 8GB (16GB recommended for training)
- **Disk**: 10GB free space
- **OS**: macOS, Linux, or Windows (WSL2 recommended)

---

1. Clone Repository

```
# Clone the repository
git clone
https://github.com/vishalvishal099/BinaryImageClassification_For_A_Pet_Adopt
ion_Platform.git

# Navigate to project directory
cd BinaryImageClassification_For_A_Pet_Adoption_Platform
```

---

2. Python Environment Setup

Option A: Using venv (Recommended)

```
# Create virtual environment
python3 -m venv venv

# Activate virtual environment
# macOS/Linux:
source venv/bin/activate

# Windows (PowerShell):
.\venv\Scripts\Activate.ps1

# Windows (Command Prompt):
.\venv\Scripts\activate.bat
```

Option B: Using Conda

```
# Create conda environment
conda create -n cats-dogs python=3.10 -y

# Activate environment
conda activate cats-dogs
```

Verify Python Version

```
python --version
# Should output: Python 3.10.x or higher
```

3. Install Dependencies

Install Runtime Dependencies

```
pip install --upgrade pip

# Install main dependencies
pip install -r requirements.txt
```

Install Development Dependencies (for testing/linting)

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

Verify Installation

```
# Check key packages
python -c "import torch; print(f'PyTorch: {torch.__version__}')"
python -c "import fastapi; print(f'FastAPI: {fastapi.__version__}')"
python -c "import mlflow; print(f'MLflow: {mlflow.__version__}')"
python -c "import dvc; print(f'DVC: {dvc.__version__}')"
```

Expected Output:

```
PyTorch: 2.5.1
FastAPI: 0.115.5
MLflow: 2.18.0
DVC: 3.56.0
```

4. Environment Configuration

Create Environment File

```
# Copy example environment file
cp .env.example .env

# Edit with your settings
nano .env  # or use any editor
```

Key Environment Variables

```
# .env file contents
MODEL_PATH=models/best_model.pt
MODEL_NAME=simple_cnn
PORT=8000
LOG_LEVEL=INFO
MLFLOW_TRACKING_URI=mlruns
MLFLOW_EXPERIMENT_NAME=cats_dogs_classification
```

Load Environment Variables

```
# Automatically loaded by python-dotenv, or manually:
export $(cat .env | xargs)
```

---

5. Data Setup with DVC

Option A: Pull Data from Dagshub (Recommended)

The dataset is hosted on **Dagshub** (10GB free storage). Simply pull:

```
# Pull data from Dagshub
dvc pull

# This downloads ~1.4GB of images to data/raw/
```

> **Note**: DVC remote is already configured in `.dvc/config` pointing to: `https://dagshub.com/vishalvishal099/BinaryImageClassification_For_A_Pet_Adoption_Platform.dvc`

Option B: Download from Kaggle

```
# Install Kaggle CLI
pip install kaggle

# Configure Kaggle API credentials
# Download kaggle.json from https://www.kaggle.com/account
mkdir -p ~/.kaggle
cp kaggle.json ~/.kaggle/
chmod 600 ~/.kaggle/kaggle.json

# Download dataset
kaggle datasets download -d salader/dogs-vs-cats

# Extract to data/raw
unzip dogs-vs-cats.zip -d data/raw
```

Option C: Manual Download

1. Visit: https://www.kaggle.com/datasets/salader/dogs-vs-cats
2. Download and extract to `data/raw/`
3. Ensure structure:

   ```
   data/
   └── raw/
       ├── cats/
       │   ├── 1.jpg
       │   ├── 2.jpg
       │   └── ...
       └── dogs/
           ├── 1.jpg
           ├── 2.jpg
           └── ...
   ```

Track Data with DVC

```
# Add data to DVC tracking
dvc add data/raw

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

# Push data to remote storage (optional)
dvc push
```

---

6. Train the Model

Run DVC Pipeline (Recommended)

```
# Run entire pipeline: preprocess → train → evaluate
dvc repro

# View pipeline DAG
dvc dag
```

Manual Training

```
# Step 1: Preprocess data
python src/data/preprocess.py --raw-dir data/raw --processed-dir
data/processed
```

```
# Step 2: Train model
python src/training/train.py --config configs/train_config.yaml --data-dir
data/processed

# Step 3: Evaluate model
python scripts/evaluate.py --model-path models/best_model.pt --data-dir
data/processed
```

View MLflow Experiments

```
# Start MLflow tracking server (port 5001)
mlflow server --host 0.0.0.0 --port 5001 --backend-store-uri
sqlite:///mlflow.db

# Open browser: http://localhost:5001
# Dagshub remote:
https://dagshub.com/vishalvishal099/BinaryImageClassification_For_A_Pet_Adop
tion_Platform.mlflow
```

MLflow UI shows:

- Training runs with parameters and metrics
- Loss curves and confusion matrix artifacts
- Model artifacts (`best_model.pt`)

Check Training Outputs

```
# View metrics
cat models/metrics.json

# View evaluation report
cat reports/evaluation_metrics.json

# List artifacts
ls -la models/
```

---

7. Run the Application

Option A: Quick Start Script

```
# Make script executable
chmod +x run_local.sh

# Run all services
./run_local.sh
```

This starts:

- MLflow tracking server on http://localhost:5001
- FastAPI inference service on http://localhost:8000

Option B: Manual Start

```
# Terminal 1: Start MLflow tracking server
mlflow server --host 0.0.0.0 --port 5001 --backend-store-uri
sqlite:///mlflow.db

# Terminal 2: Start FastAPI
uvicorn src.inference.app:app --host 0.0.0.0 --port 8000 --reload
```

Test the API

```
# Health check
curl http://localhost:8000/health | jq

# Expected output:
# {"status": "healthy", "model_loaded": true}

# Predict an image
curl -X POST http://localhost:8000/predict \
  -F "file=@data/raw/cats/1.jpg" | jq

# Expected output:
# {"prediction": "cat", "confidence": 0.95, "class_probabilities": {...}}

# View metrics
curl http://localhost:8000/metrics

# Open API docs
open http://localhost:8000/docs  # macOS
xdg-open http://localhost:8000/docs  # Linux
```

Stop Services

```
# Stop all background processes
pkill -f 'mlflow ui'
pkill -f 'uvicorn'
```

---

8. Docker Setup

Build Docker Image

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

# Verify build
docker images | grep cats-dogs
```

Run Docker Container

```
# Run container
docker run -d \
  --name cats-dogs-api \
  -p 8000:8000 \
  -v $(pwd)/models:/app/models:ro \
  cats-dogs-classifier:latest

# Check container status
docker ps

# View logs
docker logs -f cats-dogs-api

# Test the container
curl http://localhost:8000/health
```

Docker Compose (Full Stack)

```
# Start main service
docker-compose up -d

# Start with monitoring stack (Prometheus + Grafana)
docker-compose --profile monitoring up -d

# Start with development tools (MLflow server)
docker-compose --profile dev up -d

# Start everything
docker-compose --profile dev --profile monitoring up -d

# View logs
docker-compose logs -f

# Stop all services
docker-compose down
```

Docker Compose Services

Service	Port	URL	Profile

Service	Port	URL	Profile
classifier	8000	http://localhost:8000	default
mlflow	5001	http://localhost:5001	dev
prometheus	9090	http://localhost:9090	monitoring
grafana	3000	http://localhost:3000	monitoring

---

9. Kubernetes Deployment

Prerequisites

```
# Start minikube (if using minikube)
minikube start

# Or start kind cluster
kind create cluster --name cats-dogs
```

GitOps with ArgoCD (Recommended)

The project uses **ArgoCD** for GitOps-based continuous deployment. When the CD pipeline updates `k8s/local/deployment.yaml` and pushes to `main`, ArgoCD automatically detects the change and syncs the cluster — no manual `kubectl apply` needed.

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

# Wait for ArgoCD to be ready
kubectl wait --for=condition=available deployment/argocd-server -n argocd --
timeout=120s

# Apply ArgoCD application manifest (watches k8s/local/ on main branch)
kubectl apply -f k8s/argocd-application.yaml

# Access ArgoCD UI (port-forward)
kubectl port-forward svc/argocd-server -n argocd 9443:443 &
# UI: https://localhost:9443  (login: admin)

# Get initial admin password
kubectl -n argocd get secret argocd-initial-admin-secret \
  -o jsonpath="{.data.password}" | base64 -d

# Check app status
argocd app get cats-dogs-classifier
argocd app list
```

GitOps Flow:

```
git push → CI Pipeline → CD: updates k8s/local/deployment.yaml → git push
[skip ci]
    └──▶ ArgoCD detects manifest change → auto-syncs to Minikube cluster
```

Manual Deploy to Kubernetes (Alternative)

```
# Apply all manifests
kubectl apply -f k8s/local/

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

# Wait for pod to be ready
kubectl wait --for=condition=ready pod \
  -l app=cats-dogs-classifier \
  -n cats-dogs-classifier \
  --timeout=120s
```

Access the Service

```
# Port forward to access locally
kubectl port-forward svc/cats-dogs-classifier 8000:8000 \
  -n cats-dogs-classifier &

# Test the service
curl http://localhost:8000/health

# View pod logs
kubectl logs -f deploy/cats-dogs-classifier -n cats-dogs-classifier
```

Clean Up Kubernetes

```
# Delete all resources
kubectl delete -f k8s/local/

# Or delete namespace (removes everything in it)
kubectl delete namespace cats-dogs-classifier
```

---

10. Monitoring Setup

Start Monitoring Stack

```
# Prometheus (Podman container) — port 9090
podman run -d --name prometheus \
  -p 9090:9090 \
  -v $(pwd)/monitoring/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
# UI: http://localhost:9090/graph

# Metrics server (push_metrics.py) — port 8081
python scripts/push_metrics.py &

# Grafana (Homebrew) — port 3000
brew services start grafana
# Dashboard: http://localhost:3000/d/pet-adoption-ml-v2
```

Service URLs

Tool	URL	Credentials
Prometheus	http://localhost:9090/graph	N/A
Grafana	http://localhost:3000/d/pet-adoption-ml-v2	admin/admin
Metrics Server	http://localhost:8081/metrics	N/A
FastAPI Metrics	http://localhost:8000/metrics	N/A

Prometheus Metrics Available (31 metric families)

```
# API request metrics
api_requests_total{endpoint, method, status}
api_request_duration_seconds{endpoint, quantile}  # p50/p90/p95/p99

# Prediction metrics
prediction_count{class="cat|dog", model_version}
model_accuracy, model_precision, model_recall, model_f1_score

# System metrics
system_cpu_usage_percent, system_memory_usage_bytes

# Data pipeline metrics
dvc_pull_duration_seconds, preprocessing_duration_seconds
```

Configure Grafana

1. Open http://localhost:3000
2. Login: admin / admin
3. Add Prometheus datasource:
   - URL: http://localhost:9090
4. Dashboard auto-provisioned at `http://localhost:3000/d/pet-adoption-ml-v2`

---

11. Running Tests

Run All Tests

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

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

# Open coverage report
open htmlcov/index.html  # macOS
xdg-open htmlcov/index.html  # Linux
```

Run Specific Tests

```
# Run preprocessing tests
pytest tests/test_preprocess.py -v

# Run inference tests
pytest tests/test_inference.py -v

# Run with verbose output
pytest tests/ -v --tb=short
```

Code Quality Checks

```
# Format code
black src/ tests/
isort src/ tests/

# Lint code
flake8 src/ tests/ --max-line-length=100

# Type check
mypy src/ --ignore-missing-imports

# Run all checks (like CI)
make lint  # if using Makefile
```

Troubleshooting

Common Issues

1. Python Version Mismatch

```
# Error: Package requires Python 3.11+
# Solution: Use Python 3.10 compatible versions (already configured)

# Check Python version
python --version
```

2. Port Already in Use

```
# Error: Address already in use
# Find process using port
lsof -i :8000

# Kill process
kill -9 <PID>

# Or use different port
uvicorn src.inference.app:app --port 8001
```

3. Model Not Found

```
# Error: Model file not found
# Solution: Train model first or download pre-trained

# Check if model exists
ls -la models/best_model.pt

# Train if missing
dvc repro
```

4. Docker Build Fails

```
# Error: pip install fails in Docker
# Solution: Check requirements.txt has valid versions

# Build with no cache
docker build --no-cache -t cats-dogs-classifier .

# Check Docker logs
docker logs <container_id>
```

5. DVC Pull Fails

```
# Error: Unable to pull from remote
# Solution: DVC remote is configured to Dagshub

# Check remote config
cat .dvc/config

# Pull from Dagshub (default remote)
dvc pull

# If authentication required, configure credentials:
dvc remote modify dagshub --local auth basic
dvc remote modify dagshub --local user <your_dagshub_username>
dvc remote modify dagshub --local password <your_dagshub_token>
```

6. CUDA/GPU Issues

```
# If GPU not available, CPU will be used automatically
# Check CUDA availability
python -c "import torch; print(torch.cuda.is_available())"

# Force CPU usage
export CUDA_VISIBLE_DEVICES=""
```

Reset Everything

```
# Clean all temporary files
make clean  # or manually:

# Remove Python cache
find . -type d -name __pycache__ -exec rm -rf {} +
find . -type f -name '*.pyc' -delete

# Remove Docker containers
docker-compose down -v
docker system prune -f

# Remove virtual environment
rm -rf venv/

# Reset DVC
dvc gc -w
```

Quick Reference

Makefile Commands

```
make help          # Show all commands
make setup         # Full project setup
make install       # Install dependencies
make lint          # Run linters
make test          # Run tests
make train         # Train model
make build         # Build Docker image
make run           # Run Docker container
make deploy-k8s    # Deploy to Kubernetes
make clean         # Clean artifacts
```

Common Workflows

```
# Full Development Setup
git clone <repo> && cd <repo>
python3 -m venv venv && source venv/bin/activate
pip install -r requirements-dev.txt
dvc pull  # or download data manually
dvc repro  # train model
./run_local.sh  # start services

# Docker Development
docker build -t cats-dogs-classifier .
docker run -p 8000:8000 cats-dogs-classifier
curl http://localhost:8000/health

# Run Tests Before Commit
black src/ tests/ && isort src/ tests/
flake8 src/ tests/
pytest tests/ -v

# View CI/CD Status
gh run list --limit 5
gh run view <run-id> --log
```

Useful URLs (When Running)

Service	URL
FastAPI Docs	http://localhost:8000/docs
FastAPI Health	http://localhost:8000/health
FastAPI Metrics	http://localhost:8000/metrics
MLflow UI	http://localhost:5001
Prometheus	http://localhost:9090/graph

Service	URL
Grafana	http://localhost:3000/d/pet-adoption-ml-v2
ArgoCD UI	https://localhost:9443
Metrics Server	http://localhost:8081/metrics
Dagshub	https://dagshub.com/vishalvishal099/BinaryImageClassification_For_A_Pet_Adoption_Platform

---

📁 Project Structure

```
BinaryImageClassification_For_A_Pet_Adoption_Platform/
├── .github/workflows/          # CI/CD pipelines
│   ├── ci.yml                  # Continuous Integration
│   └── cd.yml                  # Continuous Deployment
├── configs/                    # Configuration files
│   └── train_config.yaml       # Training hyperparameters
├── data/                       # Data directory (DVC tracked)
│   ├── raw/                    # Raw images
│   └── processed/              # Preprocessed data
├── docs/                       # Documentation
│   ├── ARCHITECTURE_DIAGRAM.md # CI/CD architecture
│   └── SETUP_GUIDE.md          # This file
├── k8s/                        # Kubernetes manifests
│   ├── local/                  # Local K8s deployment
│   └── argocd-application.yaml # GitOps configuration
├── models/                     # Trained models
│   ├── best_model.pt           # Best model checkpoint
│   └── metrics.json            # Training metrics
├── monitoring/                 # Monitoring configs
│   ├── prometheus.yml          # Prometheus config
│   └── grafana/                # Grafana dashboards
├── notebooks/                  # Jupyter notebooks
│   └── eda.ipynb               # Exploratory analysis
├── reports/                    # Generated reports
│   └── evaluation_metrics.json # Model evaluation
├── scripts/                    # Utility scripts
│   ├── smoke_test.py           # Post-deployment tests
│   └── evaluate.py             # Model evaluation
├── src/                        # Source code
│   ├── data/                   # Data processing
│   ├── inference/              # FastAPI application
│   ├── models/                 # Model architectures
│   ├── training/               # Training scripts
│   └── utils/                  # Utilities
├── tests/                      # Unit tests
├── .env.example                # Environment template
├── docker-compose.yml          # Docker Compose config
├── Dockerfile                  # Container definition
├── dvc.yaml                    # DVC pipeline
├── Makefile                    # Build automation
├── requirements.txt            # Runtime dependencies
├── requirements-dev.txt        # Dev dependencies
└── run_local.sh                # Quick start script
```
