MLOps Laboratory Assignment 3

Experiment Tracking and Versioning with MLFlow

Machine Learning Operations
December 13, 2025

Contents

1 Introduction

This report presents the work completed for Laboratory Assignment 3 of the MLOps course, focusing on experiment tracking and model versioning using MLFlow. This assignment builds upon the previous laboratories (Lab 1 and Lab 2) by replacing the random prediction model with a deep learning classifier trained using transfer learning on the Oxford-IIIT Pet dataset.

The main objectives accomplished in this laboratory are:

- Implementation of transfer learning using lightweight deep learning models
- Experiment tracking and model versioning with MLFlow
- Model serialization in ONNX format for production deployment
- Integration of the serialized model into the inference API

2 Repository and Deployment Links

2.1 GitHub Repositories

- **Lab 1:** github.com/MariaGoico/MLOps-Lab1
- **Lab 2:** github.com/MariaGoico/MLOps-Lab2
- **Lab 3:** github.com/MariaGoico/MLOps-Lab3

2.2 HuggingFace Spaces

- **Lab 2 Deployment:** huggingface.co/spaces/spaces/MGoico/Lab-2-Mlops
- **Lab 3 Deployment:** huggingface.co/spaces/MGoico/MLOps-Lab3

3 Testing Strategy

3.1 Overview

The testing strategy for this project ensures the reliability and correctness of the MLOps pipeline, from data preprocessing to model inference.

3.2 Test Architecture

The test suite is organized into four main test modules:

1. **test_logic.py** - Unit tests for core business logic and preprocessing functions
2. **test_cli.py** - Integration tests for the command-line interface
3. **test_api.py** - Integration tests for the FastAPI endpoints
4. **test_artifacts.py** - Pre-deployment validation tests for model artifacts

3.3 Test Categories

3.3.1 Unit Tests (test_logic.py)

The unit tests validate the core functionality of the image processing and prediction pipeline:

Prediction Functions:

- **predict_simple():** Tests both ONNX classifier usage and fallback to random prediction
  - Validates that predictions always return valid class labels from `class_labels.json`
  - Tests error handling when classifier is unavailable
  - Verifies graceful fallback on runtime errors
- **predict():** Tests the confidence-aware prediction function
  - Validates tuple return format (class, confidence)
  - Tests fallback behavior returning `None` confidence
  - Verifies ONNX classifier integration with mocked predictions

Image Preprocessing Functions:

- **resize():** Tests image resizing with various parameter combinations
  - Specific width and height dimensions
  - Random dimensions when parameters not provided
  - Partial specification (width-only or height-only)
  - Error handling for invalid dimensions (negative, zero)
- **to_grayscale():** Validates conversion to grayscale mode
- **normalize():** Tests autocontrast application and pixel value scaling to [0,1]
- **random_rotate():** Verifies rotation within expected angle range (-20° to +20°)
- **random_flip():** Tests conditional horizontal flipping based on probability
- **blur():** Validates Gaussian blur filter application
- **preprocess():** Tests complete preprocessing pipeline execution order

Utility Functions:

- **ensure_output_dir():** Tests directory creation and idempotency

Class Labels Validation:

- Validates 37 expected classes from Oxford-IIIT Pet dataset
- Confirms presence of known breed names

3.3.2 CLI Integration Tests (test_cli.py)

Tests for the Click-based command-line interface covering two main command groups:

Classify Group:

- **predict command:** Tests image classification via CLI
  - Success cases with valid images
  - Error handling for non-existent files
  - Mocked prediction validation

Preprocess Group:

- **resize command:** Tests with explicit dimensions, random dimensions, and default output paths
- **grayscale command:** Validates grayscale conversion
- **rotate command:** Tests random rotation functionality
- **flip command:** Tests random horizontal flip
- **blur command:** Tests Gaussian blur application
- **pipeline command:** Tests complete preprocessing pipeline

Edge Cases:

- Partial dimension specification (width-only, height-only)
- Default output path handling
- Help message validation for all command groups

Testing Strategy:

- Uses Click's `CliRunner` for isolated command execution
- Temporary directories for output isolation
- Mocking with `unittest.mock` for unit-level CLI testing
- Fixtures for reusable test images and directory mocking

3.3.3 API Integration Tests (test_api.py)

Tests for the FastAPI web service endpoints:

Endpoints Tested:

1. **Home Page (GET /):**
   - Validates HTML response and correct content type
2. **Predict Endpoint (POST /predict):**
   - Tests with real images returning valid class predictions
   - Tests with mocked predictions for deterministic validation
   - Error handling for invalid file types
   - Validates JSON response structure (`predicted_class`, `filename`)
3. **Resize Endpoint (POST /resize):**
   - Fixed size resizing with explicit width/height
   - Random size resizing when dimensions not provided
   - Validates returned image dimensions
   - Error handling for invalid dimensions (negative values)
   - Mocked resize function testing
4. **Get Output File (GET /outputs/{filename}):**
   - Tests retrieval of existing files
   - Error handling for non-existent files

Testing Approach:

- Uses FastAPI's `TestClient` for HTTP request simulation
- In-memory image generation with PIL for test isolation
- Temporary output directories to avoid filesystem pollution
- **Fixtures for expected class labels** with fallback for CI/CD environments
- Both real integration tests and mocked unit tests for comprehensive coverage

3.3.4 Artifact Existence Tests (test_artifacts.py)

Critical pre-deployment validation ensuring required artifacts exist:

```
def test_model_artifacts_exist():
    """Test that required model artifacts exist before deployment"""
    model_path = Path("model.onnx")
    labels_path = Path("class_labels.json")

    assert model_path.exists(), f"Model file not found at {model_path}"
    assert labels_path.exists(), f"Class labels file not found at {labels_path}"
```

These tests run before containerization to catch missing artifacts early, preventing deployment failures on platforms like Render or HuggingFace Spaces.

3.4 Testing Fixtures and Utilities

Shared Fixtures Across Test Modules:

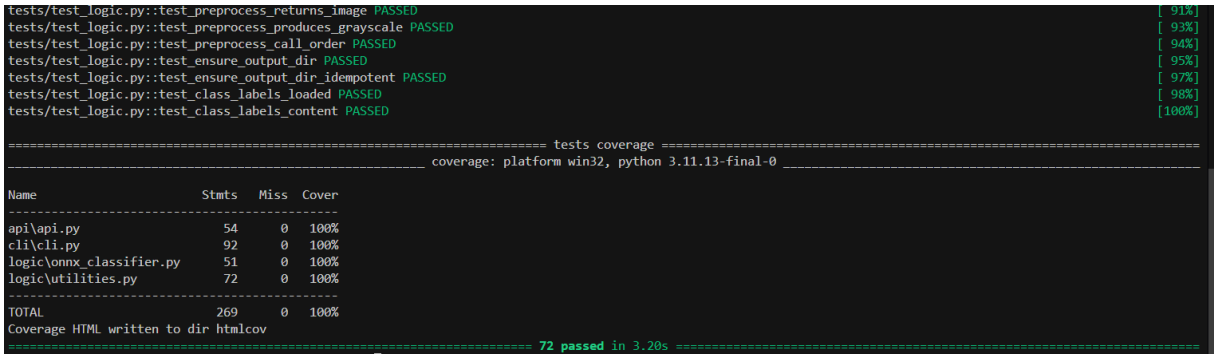
- **expected_classes:** Session-scoped fixture that loads class labels from `class_labels.json` with fallback to hardcoded list for CI environments
- **test_image / dummy_image:** Creates temporary test images for validation
- **tmp_outputs_dir / mock_outputs_dir:** Provides isolated temporary directories for output files
- **runner:** CLI testing with Click's `CliRunner`

Mocking Strategy:

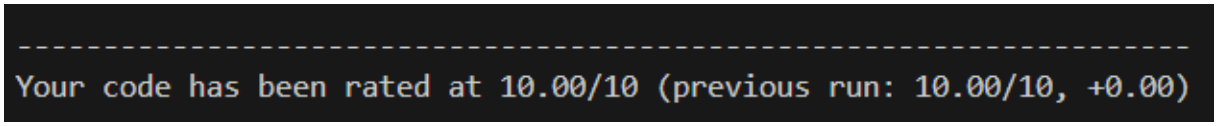
The test suite uses `unittest.mock` to:

- Mock ONNX classifier availability and predictions
- Mock random number generation for deterministic testing
- Patch file system paths for isolated testing
- Mock image processing functions to test integration without full pipeline execution

COVERAGE



LINT



4 Experiments Conducted

4.1 Experimental Setup

4.1.1 Dataset

- **Dataset:** Oxford-IIIT Pet Dataset
- **Number of classes:** 37
- **Image preprocessing:** Resized to 256x256, take center crop of 224x224 and normalized using ImageNet statistics (mean and std)
- **Train/Validation split:** 80/20
- **Reproducibility:** Fixed random seeds (42) for dataset splitting and model initialization

4.1.2 Base Model Architecture

Describe the models you experimented with:

- **Primary model:** MobileNet_v2 with IMAGENET1K_V1 pretrained weights
- **Transfer learning approach:** Frozen feature extractor, modified classifier head
- **Optimizer:** Adam
- **Loss function:** Cross-entropy loss

4.2 Hyperparameter Configurations

Run Name	Model	Batch Size	Learning Rate	Epochs
mobilenet_v2_bs32_lr0.001	MobileNet_v2	32	0.001	3
mobilenet_v2_bs32_lr0.0001	MobileNet_v2	64	0.0001	3
mobilenet_v2_bs32_lr0.0005	MobileNet_v2	32	0.0005	5

Table 1: Experimental configurations tested

4.3 Logged Artifacts

4.3.1 Parameters Logged

For each experiment run, the following parameters were logged to MLFlow:

- Batch Size
- Dataset
- Epochs
- Image Size
- Learning Rate
- Loss Functions
- Model Name
- Number of classes
- Optimizer
- Seeds
- Train split

4.3.2 Metrics Logged

The following metrics were tracked throughout training:

- **Training accuracy** (per epoch)
- **Validation accuracy** (per epoch)
- **Training loss** (per epoch)
- **Validation loss** (per epoch)
- **Final training accuracy**
- **Final validation accuracy**

4.3.3 Artifacts Logged

Additional artifacts saved with each run:

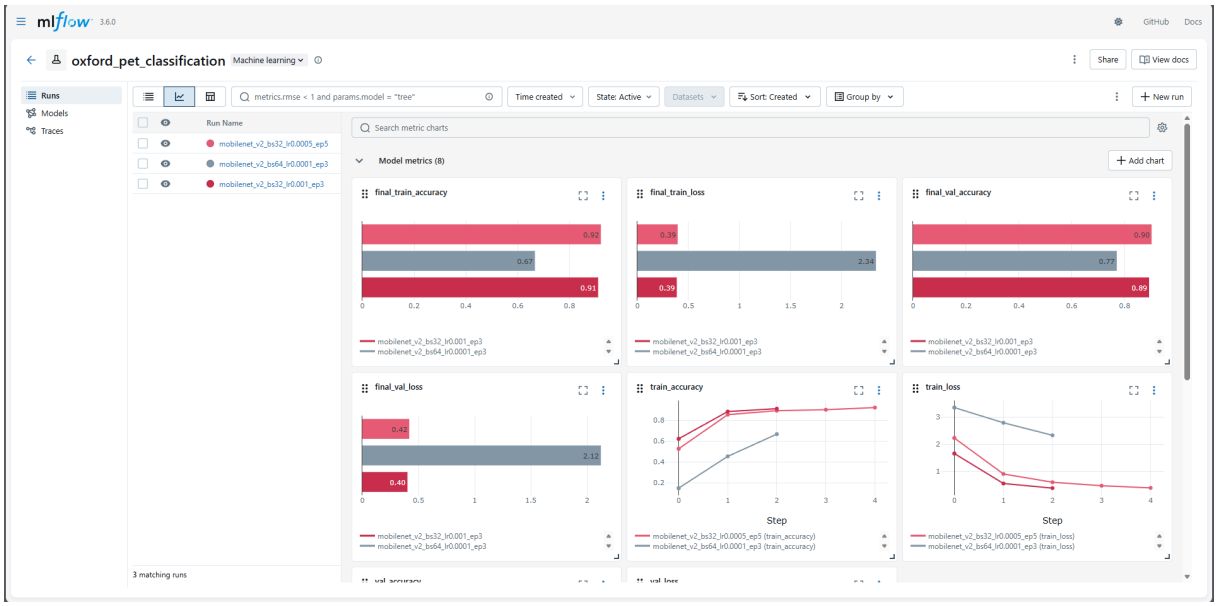
- **Training loss curve** (PNG image)
- **Validation loss curve** (PNG image)
- **Accuracy curves** (PNG image)
- **Class labels** (JSON file)
  - Example content:

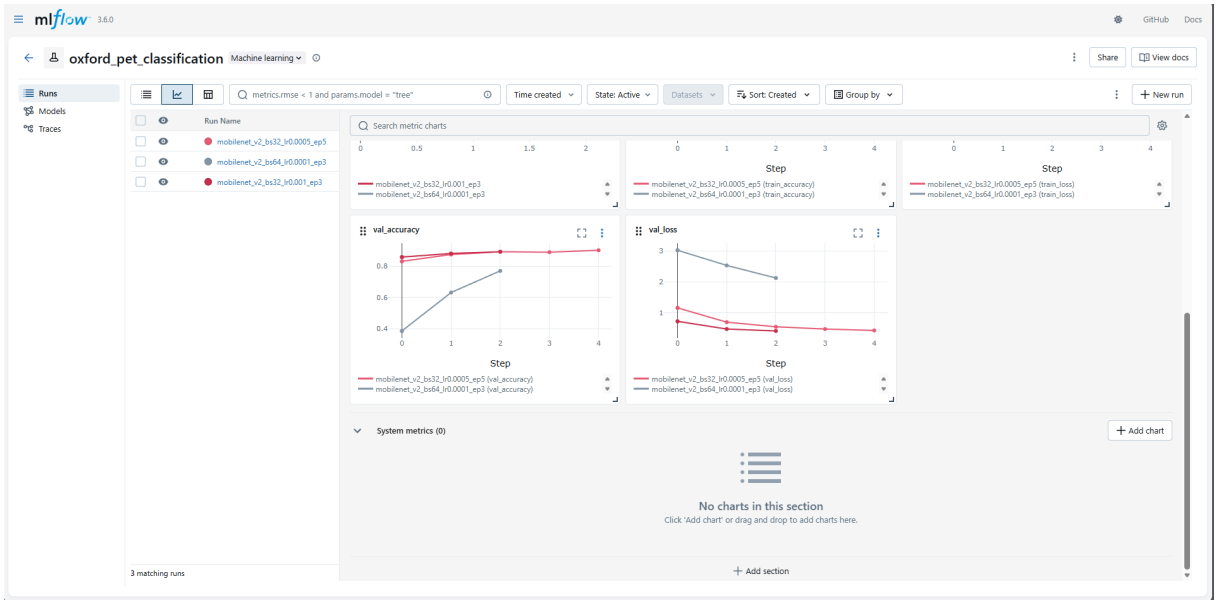
```
[
    "Abyssinian",
    "american_bulldog",
    "american_pit_bull_terrier",
    ...
]
```

- **Trained model** (PyTorch format, registered with MLFlow)

5 Results Analysis

5.1 MLFlow UI Analysis





5.1.1 Model Selection

Based on the `final_val_accuracy` metric comparison, the models ranked as follows:

Metric	Best Model	Second Best	Third Best
Run Name	**...bs32_lr0.0005_ep5**	...bs32_lr0.001_ep3	...bs64_lr0.0001_ep3
Validation Accuracy	**90%**	89%	77%
Training Accuracy	**91%**	91%	67%
Final Val Loss	**0.40**	0.40	2.12
Final Train Loss	**0.30**	0.30	0.94

Table 2: Model performance comparison from MLFlow UI

Selected Model for Production:

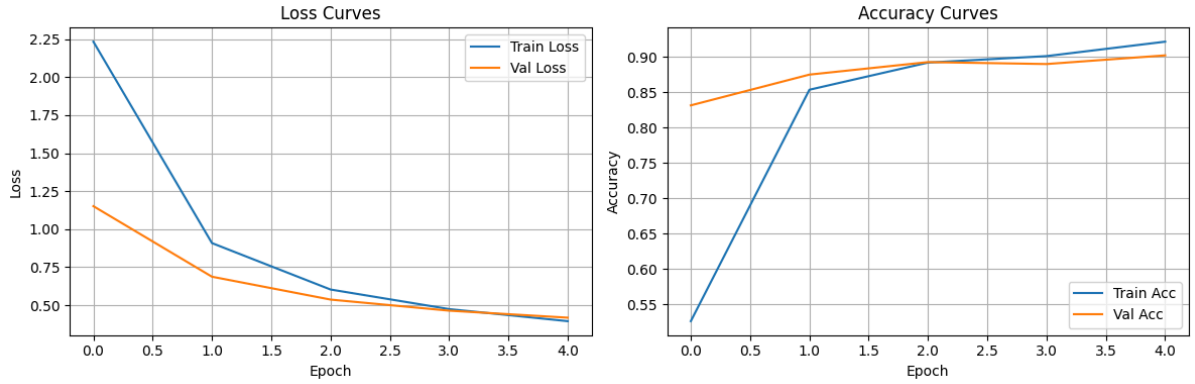
- **Run Name:** `mobilenet_v2_bs32_lr0.0005_ep5`
- **Configuration:** Batch size 32, Learning rate 0.0005, 5 epochs
- **Final Validation Accuracy:** 90%
- **Final Training Accuracy:** 91%

Justification for Selection:

Achieved the highest validation accuracy with strong generalization (1% gap), stable training curves, low validation loss (0.40), and well-balanced hyperparameters (lr=0.0005, batch size=32).

5.2 Performance Analysis

5.2.1 Training Curves



5.2.2 Final Model Performance

The selected model achieved:

- **Validation Accuracy:** 90%
- **Training Accuracy:** 91%
- **Generalization Gap:** 1% (good generalization)

5.3 Model Serialization

The best performing model was:

1. Loaded from MLFlow model registry
2. Converted to evaluation mode
3. Serialized to ONNX format (opset version 18)
4. Validated for inference compatibility

ONNX model size: 0.145 MB

ONNX model data size: 8.83 MB

6 Implementation Details

6.1 Model Selection Script

Automated selection process:

- Query all registered model versions
- Extract validation metrics for each version
- Select model with highest validation accuracy
- Serialize selected model to ONNX format
- Export class labels as JSON

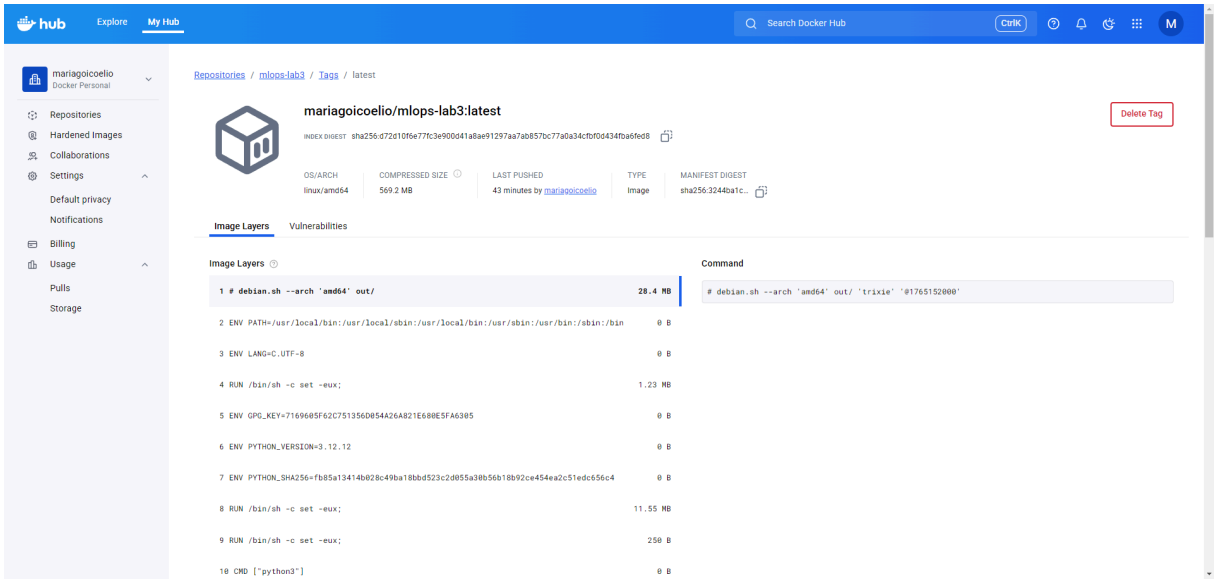
6.2 Inference Script

ONNX Runtime integration:

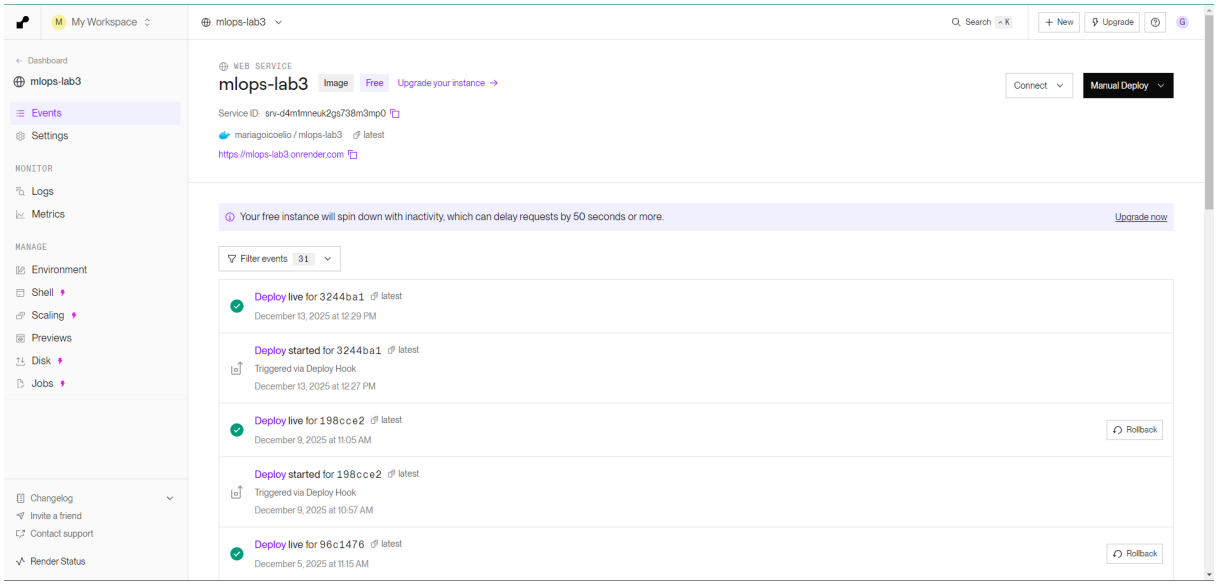
- InferenceSession with CPU execution provider
- Image preprocessing pipeline
- Prediction with class label mapping
- API integration for production deployment

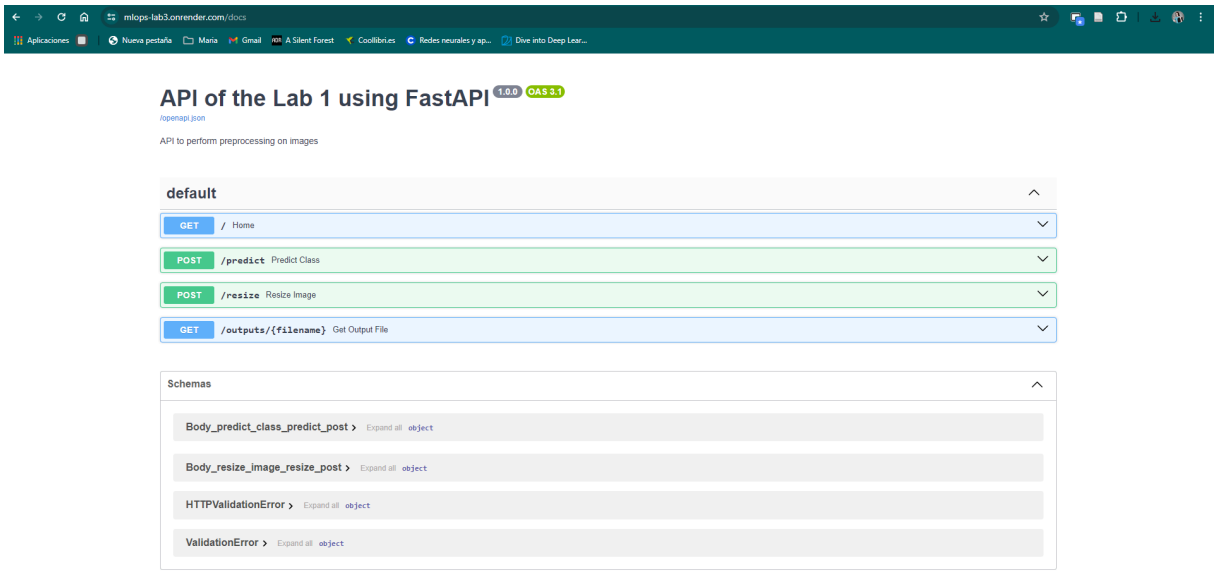
7 Services Working

7.1 DockerHub

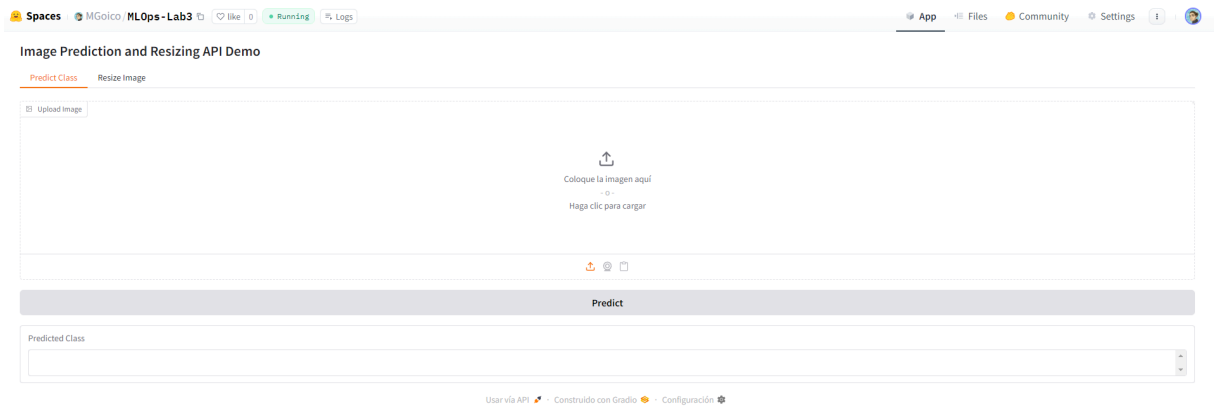


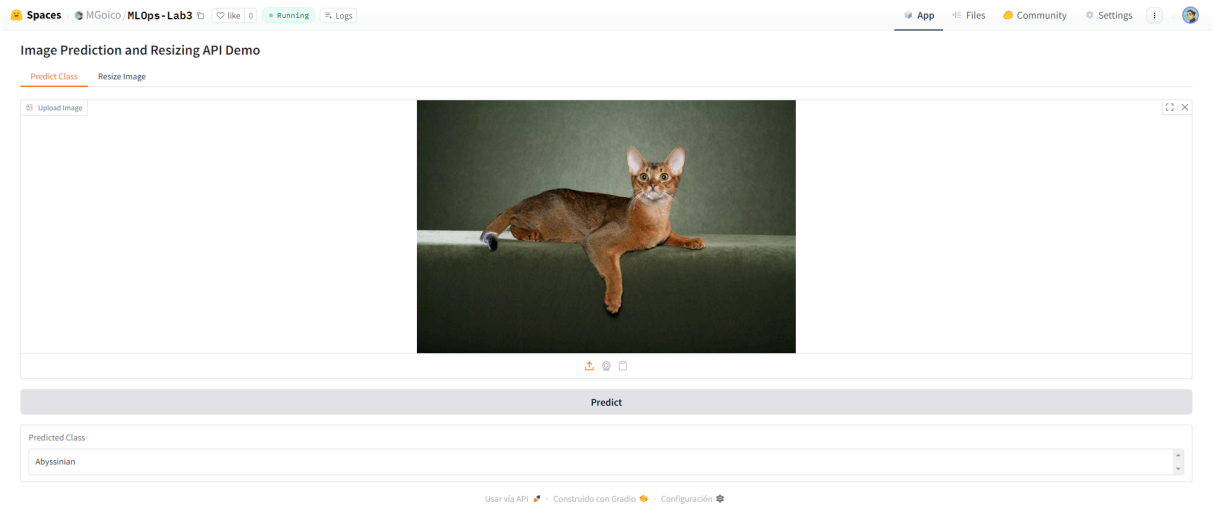
7.2 Render





7.3 HuggingFace





8 Challenges and Solutions

8.1 Docker

Challenge: Took too long to build and deployment failed.

Solution: Modified the pytorch version so that it does not include the nvidia CUDA libraries since our model will only be used for inference and no further training will be performed.