Detecting Swimming Pools from Satellite Images with Azure AutoML for Images

Making Object Detection Accessible to Everyone

Afficher l'image

Object detection has traditionally been a complex task requiring deep expertise in machine learning, computer vision, and deep learning frameworks. But what if I told you that you could build a production-ready object detection model to identify swimming pools from satellite images in just a few hours, without writing a single line of neural network code?

Welcome to the world of **Azure AutoML for Images** — where cutting-edge computer vision meets simplicity and accessibility.

The Use Case: Swimming Pool Detection

Imagine you're working for a municipality, insurance company, or real estate firm. You need to identify properties with swimming pools across thousands of satellite images. Doing this manually would take weeks. Training a custom object detection model from scratch? That's months of work requiring specialized ML expertise.

With Azure AutoML for Images, you can accomplish this in an afternoon. Let me show you how.

What is AutoML for Images?

Azure AutoML for Images is a powerful feature within Azure Machine Learning that automatically:

- **Selects the best model architecture** (YOLO, Faster R-CNN, RetinaNet, etc.)
- **Optimizes hyperparameters** through intelligent search
- **Handles data preprocessing** and augmentation
- **Trains and evaluates models** on powerful GPU compute
- **Deploys models** as REST APIs for real-time inference

All you need to do is provide labeled training data and configure a few simple parameters. AutoML does the heavy lifting.

The 4-Step Journey

Our swimming pool detection project follows a clean, four-step workflow:

1. Download Images and Labels 📥

2. Train the Model with AutoML 🎯

3. Run Inference on New Images 🔍

4. Deploy to Edge with ONNX 🚀

Let's dive into each step!

Step 1: Download Images and Labels

First, we need training data. For object detection, this means:

- **Images**: Satellite imagery containing swimming pools
- **Labels**: Bounding box annotations in a structured format

Azure AutoML for Images expects data in a specific format called **MLTable** with annotations in JSONL (JSON Lines) format.

Setting Up Your Environment

python

```python
# Import essential libraries
from azure.ai.ml import MLClient
from azure.identity import DefaultAzureCredential
from azure.ai.ml import Input
from azure.ai.ml.constants import AssetTypes
import os

# Connect to your Azure ML workspace
ml_client = MLClient(
    DefaultAzureCredential(),
    subscription_id="<your-subscription-id>",
    resource_group_name="<your-resource-group>",
    workspace_name="<your-workspace-name>"
)

print("Connected to workspace:", ml_client.workspace_name)
```

Data Format: JSONL Annotations

Each image needs an annotation file that looks like this:

json

```json
{
  "image_url": "azureml://subscriptions/.../satellite_001.jpg",
  "label": [
    {
      "label": "swimming_pool",
      "topX": 0.25,
      "topY": 0.30,
      "bottomX": 0.45,
      "bottomY": 0.50,
      "isCrowd": 0
    }
  ]
}
```

The bounding boxes are defined as **normalized coordinates** (0 to 1) representing the position of each swimming pool in the image.

Uploading Your Dataset

python

```python
# Create training and validation data inputs
training_mltable_path = "./data/training/"
validation_mltable_path = "./data/validation/"

my_training_data_input = Input(
    type=AssetTypes.MLTABLE,
    path=training_mltable_path
)

my_validation_data_input = Input(
    type=AssetTypes.MLTABLE,
    path=validation_mltable_path
)

print("✓ Training and validation data prepared")
```

Pro tip: Split your data into 80% training and 20% validation to get reliable performance metrics.

Step 2: Train Your Object Detection Model

This is where the magic happens! With just a few lines of code, AutoML will:

- Test multiple model architectures
- Tune hyperparameters automatically
- Find the best model for your specific dataset

Configure Your Compute

First, ensure you have GPU compute available (object detection models are compute-intensive):

python

```python
from azure.ai.ml.entities import AmlCompute

# Create or retrieve GPU compute cluster
compute_name = "gpu-cluster"
try:
    gpu_cluster = ml_client.compute.get(compute_name)
    print(f"Found existing compute: {compute_name}")
except:
    print(f"Creating new compute: {compute_name}")
    gpu_cluster = AmlCompute(
        name=compute_name,
        size="Standard_NC6",  # 1 GPU
        min_instances=0,
        max_instances=4,
    )
    ml_client.compute.begin_create_or_update(gpu_cluster)
```

Create the AutoML Job

Now for the exciting part — creating your automated ML training job:

python

```python
from azure.ai.ml import automl
from azure.ai.ml.automl import ObjectDetectionPrimaryMetrics

# Define the AutoML object detection job
image_object_detection_job = automl.image_object_detection(
    compute=compute_name,
    experiment_name="swimming-pool-detection",
    training_data=my_training_data_input,
    validation_data=my_validation_data_input,
    target_column_name="label",
    primary_metric=ObjectDetectionPrimaryMetrics.MEAN_AVERAGE_PRECISION,
    tags={"project": "swimming-pools", "version": "1.0"},
)

print("✓ AutoML job configured")
```

Set Training Limits

Control how long and how many experiments AutoML runs:

python

```python
# Set job limits to control training time and cost
image_object_detection_job.set_limits(
    timeout_minutes=60,      # Maximum 1 hour
    max_trials=10,           # Try 10 different configurations
    max_concurrent_trials=2  # Run 2 experiments in parallel
)

print("✓ Training limits set")
```

Define the Search Space (Optional)

Want more control? Specify which models and hyperparameters to try:

python

```python
from azure.ai.ml.sweep import Choice, Uniform

# Extend search space for specific models
image_object_detection_job.extend_search_space([
    {
        "model_name": Choice(["yolov5"]),
        "learning_rate": Uniform(0.0001, 0.01),
        "model_size": Choice(["small", "medium"]),
    },
    {
        "model_name": Choice(["fasterrcnn_resnet50_fpn"]),
        "learning_rate": Uniform(0.0001, 0.001),
        "optimizer": Choice(["sgd", "adam", "adamw"]),
        "min_size": Choice([600, 800]),
    }
])
```

Configure Hyperparameter Sweep

Tell AutoML how to search for the best hyperparameters:

python

```python
from azure.ai.ml.sweep import BanditPolicy

# Configure sweep strategy
image_object_detection_job.set_sweep(
    sampling_algorithm="random",
    early_termination=BanditPolicy(
        evaluation_interval=2,
        slack_factor=0.2,
        delay_evaluation=6
    ),
)

print("✓ Hyperparameter sweep configured")
```

Submit the Training Job

Finally, kick off the training!

python

```python
# Submit the job
returned_job = ml_client.jobs.create_or_update(
    image_object_detection_job
)

print(f"✓ Job submitted! Job name: {returned_job.name}")
print(f"View job in Azure ML Studio: {returned_job.studio_url}")
```

What happens next?

AutoML will:

1. Provision GPU compute resources
2. Train multiple models (YOLO, Faster R-CNN, etc.)
3. Test different hyperparameter combinations
4. Evaluate each model on your validation data
5. Select the best performing model based on mean Average Precision (mAP)

This typically takes **30-60 minutes** depending on your data size and compute power.

Step 3: Model Inference — Detecting Pools in New Images

Congratulations! Your model is trained. Now let's use it to detect swimming pools in new satellite images.

Retrieve the Best Model

python

```python
# Get the best model from the AutoML run
best_child_run_id = returned_job.name + "_best_model"
best_run = ml_client.jobs.get(best_child_run_id)

print(f"Best model run ID: {best_child_run_id}")
print(f"Best model mAP: {best_run.properties['score']}")
```

Download the Trained Model

python

```python
# Download the model to local directory
model_output_dir = "./outputs"
ml_client.jobs.download(
    name=best_child_run_id,
    download_path=model_output_dir,
    output_name="model_output"
)

print(f"✓ Model downloaded to {model_output_dir}")
```

Run Inference

Now let's detect swimming pools in a new image:

python

```python
import torch
from PIL import Image
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Load the model
model_path = os.path.join(model_output_dir, "model.pt")
model = torch.load(model_path)
model.eval()

# Load and preprocess a new satellite image
test_image_path = "./test_images/satellite_new_001.jpg"
image = Image.open(test_image_path).convert("RGB")

# Run inference
with torch.no_grad():
    predictions = model([image])

# Extract predictions
boxes = predictions[0]['boxes'].cpu().numpy()
labels = predictions[0]['labels'].cpu().numpy()
scores = predictions[0]['scores'].cpu().numpy()

# Filter predictions by confidence threshold
confidence_threshold = 0.5
filtered_indices = scores >= confidence_threshold

final_boxes = boxes[filtered_indices]
final_scores = scores[filtered_indices]

print(f"✓ Detected {len(final_boxes)} swimming pools")
```

Visualize the Results

python

```python
# Visualize detections
fig, ax = plt.subplots(1, figsize=(12, 9))
ax.imshow(image)

for box, score in zip(final_boxes, final_scores):
    x1, y1, x2, y2 = box
    width = x2 - x1
    height = y2 - y1

    # Draw bounding box
    rect = patches.Rectangle(
        (x1, y1), width, height,
        linewidth=2, edgecolor='cyan', facecolor='none'
    )
    ax.add_patch(rect)

    # Add confidence score
    ax.text(
        x1, y1 - 5,
        f'Pool: {score:.2f}',
        color='cyan',
        fontsize=10,
        weight='bold',
        bbox=dict(facecolor='black', alpha=0.5)
    )

ax.axis('off')
plt.title("Swimming Pool Detection Results", fontsize=16)
plt.tight_layout()
plt.savefig("detection_results.png", dpi=150, bbox_inches='tight')
plt.show()

print("✓ Visualization saved as detection_results.png")
```

Step 4: Edge Deployment with ONNX

Want to run your model on edge devices or in environments without Azure connectivity? Convert it to **ONNX** format!

Why ONNX?

ONNX (Open Neural Network Exchange) is an open format that allows models to run on various platforms:

- Edge devices (IoT, drones, cameras)
- Mobile phones
- Web browsers

- On-premises servers

Export to ONNX

python

```python
from azure.ai.ml import Input, Output

# Create an ONNX export job
onnx_export_job = ml_client.jobs.create_or_update(
    inputs={
        "model_name": "fasterrcnn_resnet50_fpn",
        "batch_size": 1,
        "height_onnx": 600,
        "width_onnx": 800,
        "job_name": returned_job.name,
        "task_type": "image-object-detection",
    }
)

print("✓ ONNX export job submitted")
```

Run Inference with ONNX

python

```python
import onnxruntime as ort
import numpy as np

# Load ONNX model
onnx_model_path = "./outputs/model.onnx"
session = ort.InferenceSession(onnx_model_path)

# Prepare input
image = Image.open(test_image_path).convert("RGB")
image = image.resize((800, 600))
image_array = np.array(image).astype(np.float32)
image_array = np.transpose(image_array, (2, 0, 1))  # HWC to CHW
image_array = np.expand_dims(image_array, axis=0)  # Add batch dimension

# Run inference
input_name = session.get_inputs()[0].name
outputs = session.run(None, {input_name: image_array})

boxes = outputs[0]
labels = outputs[1]
scores = outputs[2]

print(f"✓ ONNX inference complete")
print(f"✓ Detected {len(boxes[scores > 0.5])} swimming pools")
```

Best Practices and Tips

1. Data Quality Matters

- Use high-resolution satellite images (at least 1024x1024 pixels)
- Ensure diverse examples: different pool sizes, shapes, and environments
- Aim for at least 100-200 labeled images for good results

2. Handling Small Objects

Swimming pools in satellite images can be small. Use **tiling** to improve detection:

python

```
image_object_detection_job.extend_search_space([
    {
        "model_name": Choice(["fasterrcnn_resnet50_fpn"]),
        "tile_grid_size": Choice(["3x2", "5x3"]),
        "tile_overlap_ratio": 0.25,
    }
])
```

3. Monitor Training Progress

Track your training in **Azure ML Studio**:

- View real-time metrics (mAP, loss curves)
- Compare different model runs
- Visualize predictions on validation data

4. Cost Optimization

- Start with smaller compute (Standard_NC6) for experimentation
- Use `max_trials=5` initially to test quickly
- Scale up to larger GPU clusters only when needed

5. Model Evaluation Metrics

Focus on these metrics:

- **mAP (mean Average Precision)**: Overall detection accuracy
- **Recall**: How many pools were found (important for completeness)
- **Precision**: How many detections were correct (important to avoid false alarms)

Real-World Applications

This swimming pool detection approach extends to countless use cases:

🏗️ **Construction Monitoring**: Detect building progress from aerial imagery
🚗 **Parking Management**: Count available parking spaces
🌳 **Environmental Monitoring**: Track deforestation or vegetation health
🏭 **Industrial Inspection**: Detect equipment or structural defects
🚁 **Agricultural Analysis**: Monitor crop health and field conditions
⚠️ **Disaster Response**: Identify damaged structures after natural disasters

---

Conclusion

Azure AutoML for Images democratizes object detection. You don't need a PhD in computer vision or years of experience with deep learning frameworks. With just:

- ✅ Labeled training data
- ✅ Azure Machine Learning workspace
- ✅ Basic Python knowledge

- ✅ A few hours of time

You can build production-ready object detection models that rival solutions built by specialized ML teams.

The swimming pool detection example we walked through can be adapted to virtually any object detection problem. The key is having quality labeled data and letting AutoML handle the complex model training and optimization.

Resources and Next Steps

📚 **Learn More:**

- Azure AutoML for Images Documentation
- Object Detection Tutorial
- Sample Notebooks on GitHub

🚀 **Try It Yourself:**

1. Clone the swimming pool detection notebooks from the repository
2. Set up your Azure ML workspace (free tier available!)
3. Label 100 images using Azure ML Data Labeling
4. Run the notebooks and see your model in action!

💬 **Questions?** Drop a comment below or reach out on LinkedIn

---

Happy detecting! 🏊 🛰️

Serge Retkowsky | Microsoft | serge.retkowsky@microsoft.com

Code Repository

Find all the code from this tutorial and more examples at: **github.com/retkowsky/object-detection-azure-automl-for-images**

The complete workflow includes:

- `1 Download images files and labels.ipynb` - Data preparation
- `2 AutoML for Object Detection.ipynb` - Model training
- `3 Object detection model inferencing.ipynb` - Running predictions
- `4 ONNX edge object detection model inferencing.ipynb` - Edge deployment

Tags: #MachineLearning #ComputerVision #Azure #ObjectDetection #AI #AutoML #Python #DataScience #ArtificialIntelligence #SatelliteImagery