Plant Vision Training and Deployment Guide with Google Cloud Vertex AI

Abstract

This document outlines a comprehensive, step-by-step process for building and deploying a machine learning model to classify plant health based on images, utilizing Google Cloud's Vertex AI platform. The goal is to create a Vertex AI Custom Image Classification model to identify plant health states (e.g., "healthy," "needs_water," "nutrient_deficient") from Kaggle images, and to establish an endpoint for integration with a PlantVisionAgent.

1. Assumptions

Before beginning, ensure the following prerequisites are met: * You have a Google Cloud Project set up with billing enabled. * You have enabled the Vertex AI API in your GCP project. * You have the `gcloud` CLI installed and configured, or you are comfortable using the Google Cloud Console. * You have downloaded the Kaggle dataset and can access the image files.

2. Phase 1: Data Preparation & Organization (Crucial for Good Results)

The foundation of any machine learning model is a well-organized dataset. This phase focuses on collecting, structuring, and preparing the image data.

2.1 Understand Your Kaggle Data

1. **Explore the Dataset:** The Kaggle link you provided seems to be a notebook for classification, not the dataset itself. You'll need the actual image dataset. Often, Kaggle datasets have images organized into folders where each folder name represents a class (e.g., `Tomato__healthy`, `Tomato_Late_blight`).
2. **Identify Relevant Classes:** Your project defines states like "healthy," "wilting," "nutrient deficient," "pest infestation." You need to map the classes in your Kaggle dataset to these target states.
    - **Direct Match:** Some Kaggle classes might directly map (e.g., a "healthy" folder).
    - **Grouping:** You might need to group multiple Kaggle disease classes under a broader category like "nutrient_deficient" or "pest_infestation" if the visual symptoms align and you want to keep your hackathon model simple.
    - **"Needs Water" (Wilting):** This specific state might not be explicitly labeled in many disease datasets. You may need to:

        - Find images specifically showing wilting plants (you might need to supplement your Kaggle data with a few such images sourced online).
        - **Hackathon Simplification:** If true wilting images are hard to find quickly, you might initially equate a specific, visually distinct disease state from Kaggle as a proxy for "needs_water" for demo purposes, and explain this limitation. This is less ideal but can save time.

2.2 Select & Organize Images for Your Hackathon Model

1. **Keep it Small & Balanced (for speed):** For a hackathon, you don't need thousands of images per class. Aim for:
    - **Minimum:** 10-20 distinct images per class you want to recognize.
    - **Ideal (Hackathon):** 50-100 images per class if time and data allow for better accuracy.
    - **Balanced:** Try to have roughly the same number of images for each class.
    - **Ensure Class Balance:** Aim for approximately the same number of images (e.g., 10 per folder) in each category to improve model accuracy.
2. **Create a Local Directory Structure:** Organize your selected images locally like this:

    `plant_health_dataset/ ├── healthy/ │ ├── image_001.jpg │ ├── image_002.jpg │ └── ... (e.g., Healthy Tomato Leaves.jpg, Blackberries vibrant green leaves.jpg) ├── needs_water/ │ ├── image_101.jpg │ ├── image_102.jpg │ └── ... (e.g., Wilting Cassava Leaves.jpg, Drooping mountain purple pitcher plant.jpg) ├── nutrient_deficient/ │ ├── image_201.jpg │ ├── image_202.jpg │ └── ... (e.g., Phaseolus vulgaris nitrogen deficiency.jpg, Potassium deficient soyabean Cedara.jpg) ├── overwatered/ │ ├── image_301.jpg │ ├── image_302.jpg │ └── ... (e.g., Soft Yellow Rush lily.jpg, Root rot in cicer arietinum.jpg) └── (other_classes_if_any)/` 3. **Image Quality:** Ensure images are reasonably clear and representative of the condition.

3. Phase 2: Upload Data to Google Cloud Storage (GCS)

This step makes your prepared dataset accessible to Vertex AI for model training.

3.1 Create a GCS Bucket

1. Go to the Google Cloud Console -> Cloud Storage -> Buckets -> Create Bucket.
2. Give it a unique name (e.g., `floraos-plant-images-yourinitials` or `plantvision-dataset`).
3. Choose a region (e.g., `us-central1`). Keep this consistent with Vertex AI.
4. Standard storage class is fine.
5. Click "Create."

3.2 Upload Your Organized Image Folders

1. Inside your new bucket, create a parent folder (e.g., `plant_health_data_for_training`).

2. Upload your class folders (`healthy`, `needs_water`, etc.) into this parent folder. The structure in GCS should mirror your local structure:

   `gs://your-bucket-name/plant_health_data_for_training/ ├── healthy/ ├── needs_water/ ├── nutrient_deficient/ ├── overwatered/ └── ...` 3. Wait for the upload to complete.

4. Phase 3: Create a Vertex AI Dataset for Image Classification

This phase involves registering your image data with Vertex AI and labeling it for training.

4.1 Go to Vertex AI in Google Cloud Console

1. Go to `https://console.cloud.google.com/vertex-ai`.
2. Verify the correct Google Cloud project is selected (top left).

4.2 Navigate to "Datasets" and Click "Create Dataset"

1. In the left-hand menu, click "Datasets" (under "Data").
2. Click the blue "Create Dataset" button.
3. **Dataset name:** E.g., `floraos_plant_health_dataset` or `plant-vision-dataset`.
4. **Data type and objective:** Select "Image" and then "Image classification (Single-label)" (assuming one primary health status per image).
5. **Region:** Choose the same region as your GCS bucket.
6. Click "Create."

4.3 Import Data

1. Once the dataset is created, you'll be prompted to import data. Click "Import."
2. **Select import method:** "Select image files from Cloud Storage."
3. **Cloud Storage path:** Browse to your GCS bucket and select the parent folder containing your class subfolders (e.g., `gs://your-bucket-name/plant_health_data_for_training/`). Vertex AI will automatically use the subfolder names as labels.
   - Alternatively, you can select "Upload images from your computer" and individually select all images from your local subfolders.

4. **Data split:** Let Vertex AI automatically split your data (e.g., 80% train, 10% validation, 10% test). This is fine for a hackathon. "Auto split" is recommended.
5. Click "Continue" and the import process will begin. This might take a few minutes depending on the number of images.

4.4 Verify Images and Labels / Add Labels to Images

1. Once imported, browse your dataset in Vertex AI to ensure images are correctly labeled according to their folder names.
2. If images are "unlabeled" after import, manually select images from the unlabeled section and assign them to their appropriate labels (e.g., `healthy`, `needs_water`, `nutrient_deficient`, `overwatered`). Repeat for all categories.
3. Go back to the dataset in Vertex AI and verify that images are correctly categorized under their respective labels.

5. Phase 4: Train a Custom Image Classification Model with AutoML

This phase leverages Vertex AI's AutoML capabilities to train your image classification model without writing code.

5.1 From your Vertex AI Dataset page, click "Train New Model."

1. Go to `https://console.cloud.google.com/vertex-ai`.
2. Click "Datasets" from the left menu.
3. Select your `plant-vision-dataset`.
4. Go to the "Browse" option.
5. Ensure all labeled pictures are selected.
6. Click the blue "Train new model" button (top right).
7. If prompted for "Objective," confirm "Classification."

5.2 Training Method

1. Select "AutoML."
2. Click "Continue."

5.3 Model Details

1. **Model name:** E.g., `floraos_plant_classifier_automl`.
2. Click "Continue."

5.4 Training Options (Optimization)

1. For a hackathon, you can choose "Optimize for faster training" to get a model quicker. "Best accuracy" will take longer.
2. Click "Continue."

5.5 Compute and Pricing (Budget)

1. **Node hours:** For a small dataset (e.g., <1000 images total), 1-2 node hours is often sufficient for AutoML to train a decent initial model. AutoML will stop early if it converges. Be mindful of your GCP free tier credits or budget.
2. You can set a maximum node hour budget.
3. Click "Start Training."

5.6 Wait for Training / Monitor Training and Review Metrics

1. This can take anywhere from 30 minutes to a few hours, depending on dataset size and selected node hours. You can monitor the progress in the Vertex AI "Models" section.
2. Once training is complete, go to the "Models" section in Vertex AI, click on your trained model.
3. Review the "Evaluate" tab. Look at metrics like precision, recall, and the confusion matrix. This will give you an idea of how well it's performing on the test set. Don't expect perfection, especially with limited data and time.

6. Phase 5: Evaluate & Deploy the Trained Model

After training, the model needs to be deployed to an endpoint to make predictions.

6.1 Evaluate Model Performance

1. Once training is complete, go to the "Models" section in Vertex AI, click on your trained model.
2. Review the "Evaluate" tab. Look at metrics like precision, recall, and the confusion matrix. This will give you an idea of how well it's performing on the test set. Don't expect perfection, especially with limited data and time.

6.2 Deploy Model to an Endpoint

1. From your model's page, click "Deploy & Test" tab, then "Deploy to endpoint."
2. **Endpoint name:** E.g., `floraos_classifier_endpoint` or `plantvision-endpoint`.
3. **Region:** Use the same region.
4. **Access:** Standard.
5. **Machine type:** A small machine type like `n1-standard-2` is usually sufficient for an image classification endpoint with moderate traffic (which is all you'll have for the demo).
6. **Traffic split:** Set your new model version to 100%.
7. Enable access logging (optional but good for debugging).
8. Click "Deploy." Endpoint creation can take 10-20 minutes.

6.3 Test the Deployed Endpoint

1. Once deployed, go to Vertex AI → "Endpoints."
2. Select your endpoint (e.g., `plantvision-endpoint`).
3. Navigate to the "Test & Use" tab.
4. You can now upload an image for prediction and see the model's output (e.g., "Healthy," "Overwatered," along with confidence scores).

7. Phase 6: Integrate with Your PlantVisionAgent (Python Code)

7.1 Get Your Endpoint ID

In Vertex AI -> Endpoints, find your deployed endpoint. You'll need its ID. It will look something like: `projects/YOUR_PROJECT_ID/locations/YOUR_REGION/endpoints/YOUR_ENDPOINT_NUMBER`.

7.2 Vertex AI Plant Vision Endpoint Example

```
 Name: floraos_classifier_endpoint
project="29067641139"
endpoint_id="1529936345189842944"
location="us-central1"
```

7.3 REST API

1. Make sure you have the Google Cloud SDK installed.
2. Run the following command to authenticate with your Google account. `bash gcloud auth application-default login`
3. Create a JSON object to hold your image data. Your image data should be a base64-encoded string. `json { "instances": [{ "content": "YOUR_IMAGE_BYTES" }], "parameters":{ "confidenceThreshold": 0.5, "maxPredictions": 5 } }`
4. Create environment variables to hold your endpoint and project IDs, as well as your JSON object. `bash ENDPOINT_ID="1529936345189842944" PROJECT_ID="29067641139" INPUT_DATA_FILE="INPUT-JSON"`
5. Execute the request: `bash curl \ -X POST \ -H "Authorization: Bearer $(gcloud auth print-access-token)" \ -H "Content-Type: application/json" \ "https://1529936345189842944.us-central1-29067641139.prediction.vertexai.goog/v1/projects/$`{PROJECT_ID}/locations/us-central1/endpoints/`${ENDPOINT_ID}:predict" \ -d "@${INPUT_DATA_FILE}"`

7.4 Python Script / Python Code for PlantVisionAgent to Call the Endpoint

You can use a simple Python script to get predictions from your active endpoint.

1. Follow the setup instructions in the Python Client for Cloud AI Platform quick start.

2. Copy and paste the sample code on Github into a new Python file: https://github.com/googleapis/python-aiplatform/blob/main/samples/snippets/prediction_service/predict_image_classification_sample.py
3. Execute your request in Python.

Example Prediction Call:

```
predict_image_classification_sample(
    project="29067641139",
    endpoint_id="1529936345189842944",
    location="us-central1",
    filename="YOUR_IMAGE_FILE"
)
```

In this case, the request will return a series of labels and their confidence scores.

Install Google Cloud AI Platform Client Library:

```
pip install google-cloud-aiplatform
```

Python Code for PlantVisionAgent to Call the Endpoint:

```python
import base64
from google.cloud import aiplatform
from google.protobuf import json_format
from google.protobuf.struct_pb2 import Value

# Configuration (Move to a config file or environment variables ideally)
PROJECT_ID = "your-gcp-project-id"
REGION = "your-gcp-region" #e.g., "us-central1"
ENDPOINT_ID = "your-vertex-ai-endpoint-id-number" # Just the number part

def predict_plant_health(image_file_path):
    """
    Sends an image to the Vertex AI endpoint for classification.
    Args:
        image_file_path (str): The local path to the image file.
    Returns:
        dict: The prediction results (e.g., {"status": "healthy", "confidence": 0.95})
              or None if prediction fails.
    """
    aiplatform.init(project=PROJECT_ID, location=REGION)
    endpoint = aiplatform.Endpoint(
        endpoint_name=f"projects/{PROJECT_ID}/locations/{REGION}/endpoints/{ENDPOINT_ID}"
    )

    try:
        with open(image_file_path, "rb") as f:
            image_bytes = f.read()
        encoded_content = base64.b64encode(image_bytes).decode("utf-8")

        # For AutoML Image Classification, instances typically:
        instances = [
            {
                "content": encoded_content
            }
        ]
        # If your model expects a different format, you might need to adjust.

        prediction = endpoint.predict(instances=instances)
        # print("Raw prediction:", prediction) # For debugging

        # Process the prediction response
        # The response structure can vary slightly. Inspect 'prediction.predictions[0]'
        # It usually contains 'displayNames' and 'confidences'
        if prediction.predictions:
            # Assuming single-label classification, take the top prediction
            top_prediction = prediction.predictions[0]
            # Check actual keys in your response, might be 'displayNames' or similar
            predicted_class = top_prediction.get('displayNames', [None])[0]
            confidence = top_prediction.get('confidences', [0.0])[0]

            if predicted_class and confidence > 0: # Basic check
                return {
                    "status": predicted_class,
                    "confidence": float(confidence)
                }
        return None

    except Exception as e:
        print(f"Error during Vertex AI prediction: {e}")
        return None

# Example Usage (within your PlantVisionAgent)
if __name__ == "__main__":
    # This is a sample image path, replace with how your agent gets images
    # For the hackathon, you'll likely have a list of test images.
    test_image = "path/to/your/test_image.jpg" # e.g., from your plant_health_dataset/healthy/

    # Ensure the test_image path is correct and accessible
    import os
    if not os.path.exists(test_image):
        print(f"Test image not found at: {test_image}")
    else:
        result = predict_plant_health(test_image)
        if result:
            print(f"Plant Health Assessment:")
            print(f"  Status: {result['status']}")
            print(f"  Confidence: {result['confidence']:.2f}")
            # Here your agent would publish this result to ADK
            # e.g., publish_to_adk({ "timestamp": "...", "plant_id": "Plant X", "image_ref": 
test_image, **result })
        else:
            print("Failed to get prediction.")
```

7.5 Authentication

- **Local Development:** If you're running this code locally and have authenticated with `gcloud auth application-default login`, the client library should pick up your credentials.
- **Cloud Environment (if deploying agent):** If your agent runs in a GCP environment (like Cloud Run or GCE), it will use the service account associated with that environment. Ensure this service account has the "Vertex AI User" role.

8. Troubleshooting Deployment Issues

If you encounter problems during deployment:

- **IAM Permissions:** Ensure your user account has `roles/aiplatform.admin` permissions. `bash gcloud projects add-iam-policy-binding YOUR_PROJECT \ --member="user:your-email@gmail.com" \ --role="roles/aiplatform.admin"`
- **Quota Limits:** Check your Cloud Quotas and request increases if necessary.
- **Model Compatibility:** Verify that the model format is supported by Vertex AI.

9. Key Hackathon Tips for This Task

- **Start Small with Data:** Get the pipeline working with just 2-3 classes and 10-20 images per class first. You can always add more data and retrain if time permits.
- **AutoML is Your Friend:** Don't try to build a custom training pipeline unless you have extensive experience and time.
- **Endpoint Takes Time:** Be patient when deploying the endpoint.
- **Iterate on the Agent Code:** Get the prediction call working, then integrate it into your ADK publishing logic.
- **Mock if Necessary:** If Vertex AI setup becomes a major blocker, have a fallback in your PlantVisionAgent that returns hardcoded/simulated health assessments based on image filename patterns, so the rest of your system can still be demoed. You can explain that Vertex AI integration is "in progress" or show the console part.

10. Next Steps

- **Integrate the API:** Incorporate the prediction API into various applications (e.g., Flask web apps, React frontends, mobile apps).

- **Set up Monitoring:** Utilize Vertex AI Model Monitoring to track model performance in production and detect data drift or concept drift.
- **Optimize Performance:** Explore techniques like model quantization or batch predictions for improved efficiency.

This detailed guide should help you connect your PlantVisionAgent to a powerful Vertex AI model! Good luck!