# Test qwen2-vl-72b-instruct against Plant Village Dataset

- Test for pest detection and disease detection
- Dataset: https://www.kaggle.com/datasets/emmarex/plantdisease
- I used Claude LLM for help with generating this code: https://claude.ai/chat/f6e03377-0750-4a3a-bdcf-8687fe581edf

```
import kagglehub

# Download latest version
path = kagglehub.dataset_download("emmarex/plantdisease")

print("Path to dataset files:", path)
```

```
from google.colab import files
files.download('pest_disease_results.csv')
```

```
from google.colab import files
uploaded = files.upload()  # Upload your saved pest_disease_results.csv
```

```
import base64
import requests
from pathlib import Path
from PIL import Image
from io import BytesIO
import json
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import os
import time
from typing import Dict
from google.colab import files

class PlantHealthTester:
    def __init__(self, api_key: str, dataset_path: str):
        self.api = "https://api.hyperbolic.xyz/v1/chat/completions"
        self.api_key = api_key
        self.headers = {
            "Content-Type": "application/json",
            "Authorization": f"Bearer {api_key}",
        }
        self.dataset_path = Path(dataset_path)

    def encode_image(self, img_path: str) -> str:
        with Image.open(img_path) as img:
            buffered = BytesIO()
            img.save(buffered, format="PNG")
            return base64.b64encode(buffered.getvalue()).decode("utf-8")

    def analyze_image(self, img_path: str) -> Dict:
        base64_img = self.encode_image(img_path)

        payload = {
            "messages": [
                {
                    "role": "user",
                    "content": [
                        {"type": "text", "text": """Analyze this plant image for health issues. Focus only on detecting pest
                         Provide a structured response in the following format:
                         {
                           "pest_detected": true|false,
                           "disease_detected": true|false,
                           "confidence_score": <number between 0 and 100>,
                           "details": "<brief description of what was found or 'healthy'>"
                         }"""},
                        {
                            "type": "image_url",
                            "image_url": {"url": f"{base64_img}"},
                        },
                    ],
```

```
                }
            ],
            "model": "Qwen/Qwen2-VL-72B-Instruct",
            "max_tokens": 2048,
            "temperature": 0.7,
            "top_p": 0.9,
        }

        response = requests.post(self.api, headers=self.headers, json=payload)
        return json.loads(response.json()['choices'][0]['message']['content'])

    def parse_folder_name(self, folder_name: str) -> Dict:
        """Extract ground truth from folder name"""
        is_healthy = 'healthy' in folder_name.lower()
        has_pest = any(pest in folder_name.lower() for pest in ['spider', 'mites', 'beetle'])
        has_disease = not is_healthy and not has_pest

        return {
            "pest_detected": has_pest,
            "disease_detected": has_disease,
            "is_healthy": is_healthy
        }

    def test_folder(self, folder_name: str) -> pd.DataFrame:
        folder = self.dataset_path / "PlantVillage" / folder_name
        results = []

        ground_truth = self.parse_folder_name(folder_name)
        image_files = list(folder.glob('*.JPG')) + list(folder.glob('*.jpg'))
        total_images = len(image_files)

        print(f"Found {total_images} images in {folder_name}")

        for i, img_path in enumerate(image_files, 1):
            try:
                prediction = self.analyze_image(str(img_path))

                result = {
                    'folder': folder_name,
                    'image': img_path.name,
                    'predicted_pest': prediction['pest_detected'],
                    'true_pest': ground_truth['pest_detected'],
                    'predicted_disease': prediction['disease_detected'],
                    'true_disease': ground_truth['disease_detected'],
                    'confidence_score': prediction['confidence_score'],
                    'details': prediction['details']
                }

                result['pest_detection_correct'] = result['predicted_pest'] == result['true_pest']
                result['disease_detection_correct'] = result['predicted_disease'] == result['true_disease']

                results.append(result)

                # Save after each image
                df = pd.DataFrame([result])
                if os.path.exists("pest_disease_results.csv"):
                    df.to_csv("pest_disease_results.csv", mode='a', header=False, index=False)
                else:
                    df.to_csv("pest_disease_results.csv", index=False)

                print(f"Processed image {i}/{total_images}: {img_path.name}")

            except Exception as e:
                print(f"Error processing {img_path}: {str(e)}")

        return pd.DataFrame(results)

def show_testing_status():
    """Show detailed testing status for each folder"""
    if not os.path.exists("pest_disease_results.csv"):
        print("No results file found")
        return

    results_df = pd.read_csv("pest_disease_results.csv")
    folder_stats = results_df.groupby('folder').agg({
        'image': 'count',
        'pest_detection_correct': 'mean',
        'disease_detection_correct': 'mean',
        'confidence_score': 'mean'
```

```
    }).round(3)

    folder_stats.columns = ['Images Tested', 'Pest Accuracy', 'Disease Accuracy', 'Avg Confidence']
    folder_stats['Pest Accuracy'] *= 100
    folder_stats['Disease Accuracy'] *= 100

    print("\nTesting Status:")
    print(folder_stats)

    return folder_stats

def list_folders():
    """List all available folders and their completion status"""
    dataset_path = "/root/.cache/kagglehub/datasets/emmarex/plantdisease/versions/1"
    plant_village_path = Path(dataset_path) / "PlantVillage"

    # Get previously tested folders
    tested_folders = set()
    if os.path.exists("pest_disease_results.csv"):
        results_df = pd.read_csv("pest_disease_results.csv")
        tested_folders = set(results_df['folder'].unique())

    # List all folders and their status
    print("\nAvailable Folders:")
    print("-" * 80)
    print(f"{'Folder Name':<50} {'Images':<10} {'Status':<10}")
    print("-" * 80)

    available_folders = []
    for folder_path in plant_village_path.iterdir():
        if folder_path.is_dir():
            folder_name = folder_path.name
            images = list(folder_path.glob('*.JPG')) + list(folder_path.glob('*.jpg'))
            num_images = len(images)
            if num_images > 0:
                status = "DONE" if folder_name in tested_folders else "PENDING"
                print(f"{folder_name:<50} {num_images:<10} {status:<10}")
                available_folders.append((folder_name, num_images, status))

    return available_folders

def test_selected_folders():
    """Test specific folders selected by user"""
    api_key = "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJiZWRuaWVAbXN1ZGVudmVyLmVkdSJ9.AB90p0orMr7QnpKbdeRjZYiGIaHBuxk
    dataset_path = "/root/.cache/kagglehub/datasets/emmarex/plantdisease/versions/1"

    # List available folders
    available_folders = list_folders()

    # Get pending folders
    pending_folders = [(name, count) for name, count, status in available_folders if status == "PENDING"]

    if not pending_folders:
        print("\nAll folders have been tested!")
        return

    print("\nPending Folders:")
    for i, (folder, count) in enumerate(pending_folders, 1):
        print(f"{i}. {folder} ({count} images)")

    # Get user selection
    while True:
        try:
            selection = input("\nEnter folder numbers to test (comma-separated) or 'all' for remaining folders: ")
            if selection.lower() == 'all':
                selected_folders = [f[0] for f in pending_folders]
                break

            indices = [int(i.strip()) - 1 for i in selection.split(',')]
            selected_folders = [pending_folders[i][0] for i in indices]
            break
        except (ValueError, IndexError):
            print("Invalid selection. Please try again.")

    # Calculate total images to process
    total_images = sum(count for name, count in pending_folders if name in selected_folders)
    print(f"\nProcessing {len(selected_folders)} folders with {total_images} total images")
    print(f"Estimated time: {total_images * 2.5 / 60:.1f} to {total_images * 3 / 60:.1f} minutes")

```

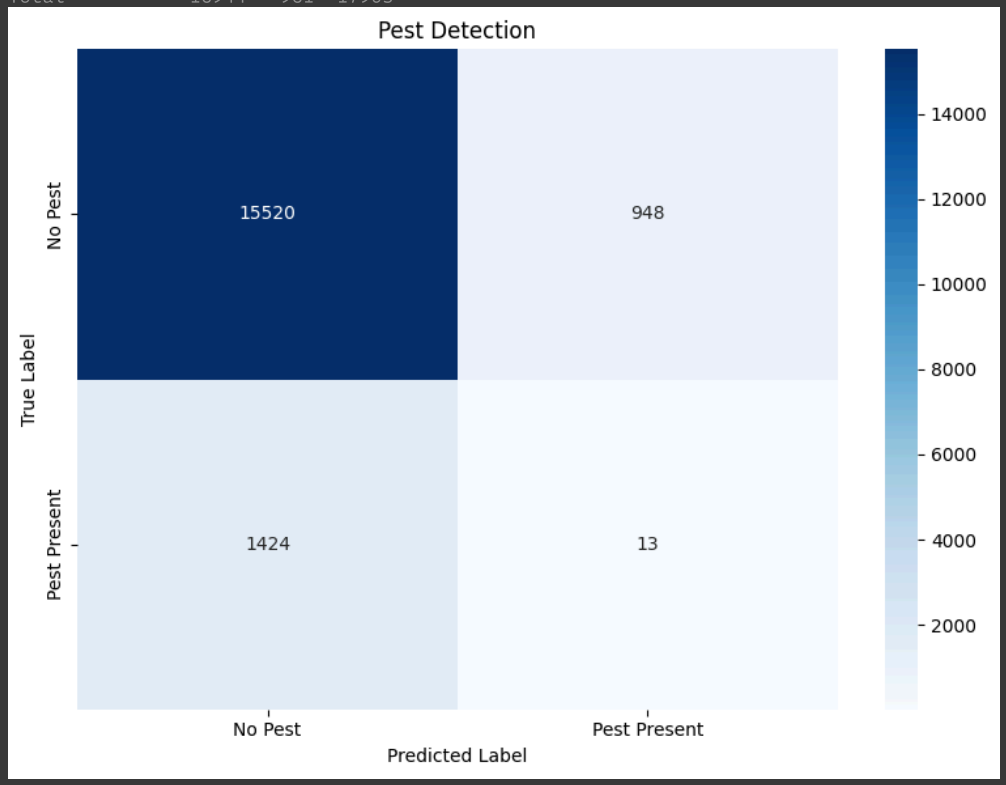
```
    proceed = input("\nProceed? (y/n): ")
    if proceed.lower() != 'y':
        return

    # Initialize tester and process selected folders
    tester = PlantHealthTester(api_key, dataset_path)

    for i, folder in enumerate(selected_folders, 1):
        print(f"\nProcessing folder {i}/{len(selected_folders)}: {folder}")
        try:
            results = tester.test_folder(folder)
            if not results.empty:
                print(f"Successfully processed folder: {folder}")
                files.download('pest_disease_results.csv')

                # Print interim results for this folder
                folder_data = results
                print(f"\nInterim Results for {folder}:")
                print(f"Pest Detection Accuracy: {folder_data['pest_detection_correct'].mean()*100:.2f}%")
                print(f"Disease Detection Accuracy: {folder_data['disease_detection_correct'].mean()*100:.2f}%")
                print(f"Average Confidence: {folder_data['confidence_score'].mean():.2f}%")
        except Exception as e:
            print(f"Error processing folder {folder}: {str(e)}")

    # Show complete results if available
    if os.path.exists("pest_disease_results.csv"):
        print("\nAnalyzing all results so far...")
        final_results = pd.read_csv("pest_disease_results.csv")

        print("\nOverall Detection Accuracy:")
        print(f"Pest Detection Accuracy: {final_results['pest_detection_correct'].mean()*100:.2f}%")
        print(f"Disease Detection Accuracy: {final_results['disease_detection_correct'].mean()*100:.2f}%")
        print(f"Average Confidence: {final_results['confidence_score'].mean():.2f}%")

        print("\nFolders completed so far:", len(final_results['folder'].unique()))
        print("Folders remaining:", len([f for f in available_folders if f[2] == "PENDING"]))

if __name__ == "__main__":
    test_selected_folders()
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import os

def create_confusion_matrix(y_true, y_pred, labels, title):
    """Create and display a confusion matrix visualization"""
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title(title)
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.tight_layout()

    # Display plot in Colab
    plt.show()

    # Also save to file
    plt.savefig(f"{title.lower().replace(' ', '_')}.png")
    plt.close()

def analyze_results():
    """Analyze pest and disease detection results"""
    # Enable inline plotting for Colab
    %matplotlib inline

    if not os.path.exists("pest_disease_results.csv"):
        print("No results file found!")
        return

    df = pd.read_csv("pest_disease_results.csv")

    # [Previous code remains the same until the plots]

    # Pest Detection Matrix
    print("\nPest Detection Results:")
```

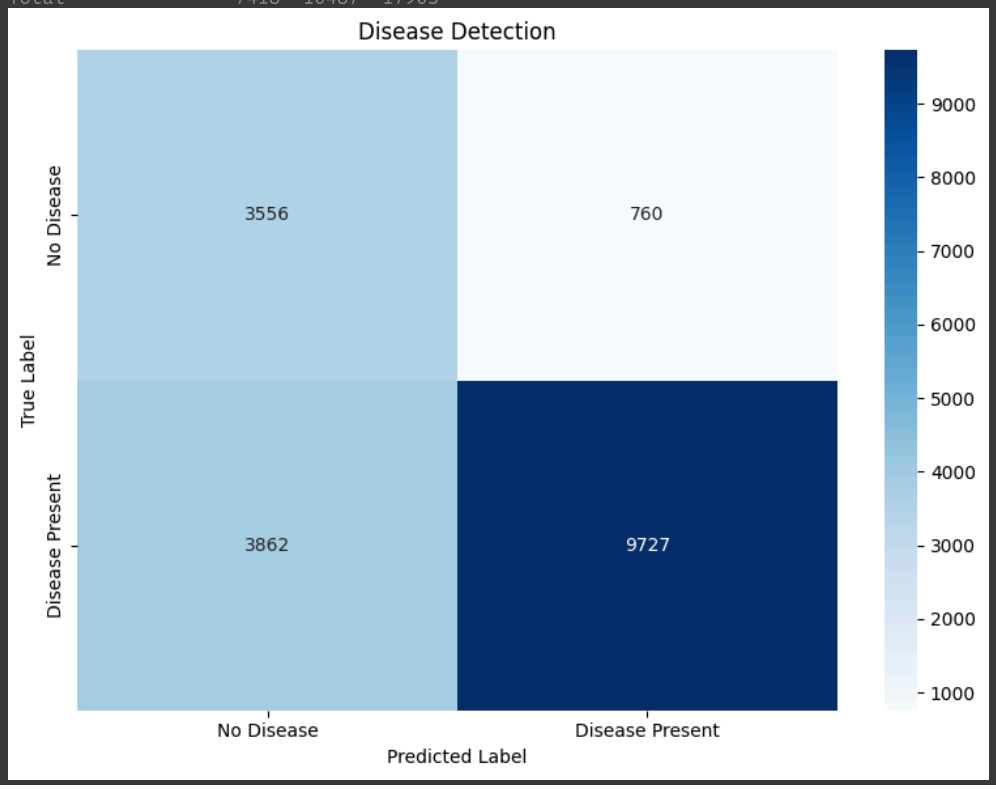
```
    print("-" * 50)
    pest_matrix = pd.crosstab(df['true_pest'], df['predicted_pest'],
                              margins=True, margins_name='Total')
    print("Confusion Matrix:")
    print(pest_matrix)

    create_confusion_matrix(
        df['true_pest'],
        df['predicted_pest'],
        ['No Pest', 'Pest Present'],
        'Pest Detection'
    )

    # Disease Detection Matrix
    print("\nDisease Detection Results:")
    print("-" * 50)
    disease_matrix = pd.crosstab(df['true_disease'], df['predicted_disease'],
                                 margins=True, margins_name='Total')
    print("Confusion Matrix:")
    print(disease_matrix)

    create_confusion_matrix(
        df['true_disease'],
        df['predicted_disease'],
        ['No Disease', 'Disease Present'],
        'Disease Detection'
    )

    # Folder Statistics
    folder_stats = df.groupby('folder').agg({
        'image': 'count',
        'pest_detection_correct': ['mean', 'count'],
        'disease_detection_correct': ['mean', 'count'],
        'confidence_score': ['mean', 'std']
    }).round(3)

    folder_stats.columns = [
        'Images',
        'Pest Accuracy', 'Pest Tests',
        'Disease Accuracy', 'Disease Tests',
        'Avg Confidence', 'Conf Std'
    ]

    folder_stats['Pest Accuracy'] *= 100
    folder_stats['Disease Accuracy'] *= 100

    print("\nDetailed Folder Statistics:")
    print(folder_stats)

    # Create and display summary visualization
    plt.figure(figsize=(15, 7))
    accuracies = folder_stats[['Pest Accuracy', 'Disease Accuracy']].sort_values('Disease Accuracy')
    ax = accuracies.plot(kind='bar')
    plt.title('Detection Accuracy by Folder')
    plt.xlabel('Folder')
    plt.ylabel('Accuracy (%)')
    plt.xticks(rotation=45, ha='right')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()

    # Display plot in Colab
    plt.show()

    # Save to file
    plt.savefig('folder_accuracies.png', bbox_inches='tight')
    plt.close()

    # Save summary statistics
    summary_df = pd.DataFrame({
        'Metric': ['Total Images', 'Total Folders', 'Avg Confidence',
                   'Pest Detection Accuracy', 'Disease Detection Accuracy'],
        'Value': [
            len(df),
            len(df['folder'].unique()),
            df['confidence_score'].mean(),
            df['pest_detection_correct'].mean() * 100,
            df['disease_detection_correct'].mean() * 100
        ]
    })
```

```
    summary_df.to_csv('analysis_summary.csv', index=False)
    print("\nSummary statistics saved to 'analysis_summary.csv'")
    print("\nPlots saved as PNG files")

if __name__ == "__main__":
    analyze_results()
```

```
Pest Detection Results:
--------------------------------------------------
Confusion Matrix:
predicted_pest  False  True  Total
true_pest
False           15520   948  16468
True             1424    13   1437
Total           16944   961  17905
```




```
Disease Detection Results:
--------------------------------------------------
Confusion Matrix:
predicted_disease  False   True  Total
true_disease
False               3556    760   4316
True                3862   9727  13589
Total               7418  10487  17905
```



Detailed Folder Statistics:

| folder | Images | Pest Accuracy | \ |
|---|---|---|---|
| Pepper__bell___Bacterial_spot | 925 | 93.9 | |
| Pepper__bell___healthy | 1323 | 99.6 | |
| Potato___Early_blight | 973 | 94.5 | |
| Potato___Late_blight | 918 | 91.3 | |
| Potato___healthy | 130 | 100.0 | |
| Tomato_Bacterial_spot | 1988 | 92.8 | |
| Tomato_Early_blight | 883 | 86.9 | |
| Tomato_Late_blight | 1647 | 87.8 | |
| Tomato_Leaf_Mold | 845 | 98.3 | |
| Tomato_Septoria_leaf_spot | 1612 | 88.3 | |
| Tomato_Spider_mites_Two_spotted_spider_mite | 1437 | 0.9 | |
| Tomato__Target_Spot | 627 | 93.1 | |
| Tomato__Tomato_YellowLeaf__Curl_Virus | 2845 | 98.9 | |
| Tomato__Tomato_mosaic_virus | 326 | 97.2 | |
| Tomato_healthy | 1426 | 99.4 | |

| folder | Pest Tests | Disease Accuracy | \ |
|---|---|---|---|
| Pepper__bell___Bacterial_spot | 925 | 89.0 | |
| Pepper__bell___healthy | 1323 | 99.2 | |
| Potato___Early_blight | 973 | 99.6 | |
| Potato___Late_blight | 918 | 92.9 | |
| Potato___healthy | 130 | 100.0 | |
| Tomato_Bacterial_spot | 1988 | 76.2 | |
| Tomato_Early_blight | 883 | 87.3 | |
| Tomato_Late_blight | 1647 | 84.9 | |
| Tomato_Leaf_Mold | 845 | 75.7 | |
| Tomato_Septoria_leaf_spot | 1612 | 95.2 | |
| Tomato_Spider_mites_Two_spotted_spider_mite | 1437 | 57.8 | |
| Tomato__Target_Spot | 627 | 37.8 | |
| Tomato__Tomato_YellowLeaf__Curl_Virus | 2845 | 31.9 | |
| Tomato__Tomato_mosaic_virus | 326 | 23.9 | |
| Tomato_healthy | 1426 | 89.9 | |

| folder | Disease Tests | Avg Confidence | \ |
|---|---|---|---|
| Pepper__bell___Bacterial_spot | 925 | 79.876 | |
| Pepper__bell___healthy | 1323 | 88.526 | |
| Potato___Early_blight | 973 | 82.379 | |
| Potato___Late_blight | 918 | 81.345 | |
| Potato___healthy | 130 | 89.038 | |
| Tomato_Bacterial_spot | 1988 | 80.058 | |
| Tomato_Early_blight | 883 | 81.268 | |
| Tomato_Late_blight | 1647 | 81.090 | |
| Tomato_Leaf_Mold | 845 | 79.787 | |
| Tomato_Septoria_leaf_spot | 1612 | 80.890 | |