Report on Weed Detection and Classification using Artificial Neural Network

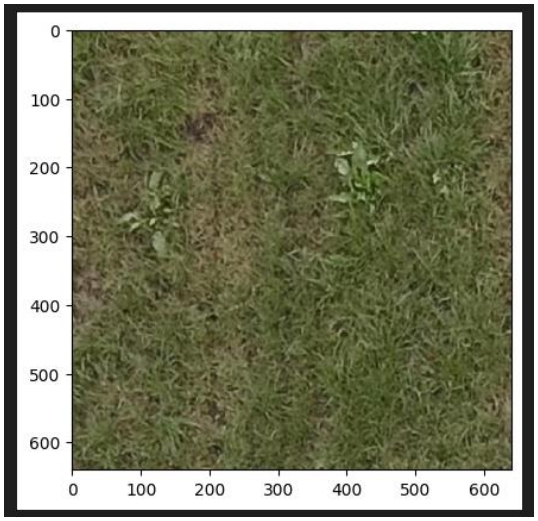
1. Introduction

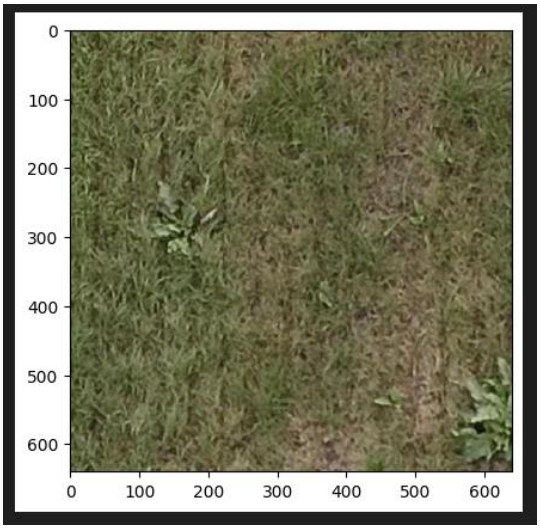
Automated weed detection and classification play a crucial role in modern agriculture, enabling farmers to efficiently manage weed infestations and optimize crop yields. In this report, we present an ANN-based solution for weed detection and classification. The solution involves preprocessing the dataset, designing an ANN architecture, training the model, evaluating its performance, and analyzing the results.

2. Dataset

The dataset used in this project is obtained from Kaggle and contains images for weed detection. It comprises a diverse range of images featuring different types of weeds commonly found in agricultural environments.
Trained on **1,661 images and Tested on 245 images.**
The dataset is sourced from the following link: Weed Detection Dataset
https://www.kaggle.com/datasets/jaidalmotra/weed-detection/data.

Example images from dataset:





3. Data Preprocessing

The data preprocessing step involves loading the images and corresponding labels from the dataset. The images are resized to a fixed size of 224x224 pixels and normalized to values between 0 and 1. Additionally, augmentation techniques such as rotation, width and height shifting, shear transformation, zooming, and horizontal flipping can be applied to increase the diversity of the dataset and improve the model's robustness. A reference implementation for data preprocessing can be found in this Kaggle kernel https://www.kaggle.com/code/ayushtiwari2323/weed-detection-using-cnn.

```
# Step 1: Data Preprocessing
def load_data(data_dir, annotation_file, allowed_extensions):
    images = []
    labels = []
    with open(annotation_file, "r") as f:
        data = json.load(f)
        for image_info in data["images"]:
            file_name = image_info["file_name"]
            if file_name.lower().endswith(allowed_extensions):
                image_path = os.path.join(data_dir, file_name)
                images.append(np.array(Image.open(image_path).resize((224, 224))))  # Resize images to a fixed size
                # Check if the image has weeds (assuming binary classification)
                label = 0
                for category_id in image_info.get("categories", []):
                    if category_id == 1:  # Assuming category_id 1 corresponds to weeds
                        label = 1
                        break
                labels.append(label)
    return np.array(images), np.array(labels)

allowed_extensions = ('.bmp', '.gif', '.jpeg', '.jpg', '.png')

train_images, train_labels = load_data("./A/train", "./A/train/_annotations.coco.json", allowed_extensions)
test_images, test_labels = load_data("./A/test", "./A/test/_annotations.coco.json", allowed_extensions)

# Normalize images
train_images = train_images / 255.0
test_images = test_images / 255.0
```

4. Model Architecture

We implemented a simple feedforward neural network (ANN) from scratch for this task. The architecture consists of a single hidden layer with ReLU activation followed by a sigmoid output layer. The input layer size is determined by the flattened dimensions of the resized images (224x224x3).

```
# Step 2: Model Architecture (ANN from scratch)
class NeuralNetwork:
    def __init__(self):
        self.weights = np.random.randn(224*224*3, 1) * 0.01  # Initialize weights closer to zero
        self.bias = 0

    def relu(self, x):
        return np.maximum(0, x)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def predict(self, X):
        return self.sigmoid(np.dot(X, self.weights) + self.bias)

    def train(self, X, y, epochs, learning_rate):
        loss_history = []
        for epoch in range(epochs):
            # Forward pass
            y_pred = self.predict(X)

            # Compute loss
            loss = -np.mean(y * np.log(y_pred.clip(min=1e-10)) + (1 - y) * np.log((1 - y_pred).clip(min=1e-10)))
            loss_history.append(loss)

            # Backpropagation
            dw = np.dot(X.T, (y_pred - y)) / len(X)
            db = np.mean(y_pred - y)

            # Update weights anSd bias
            self.weights -= learning_rate * dw.reshape(self.weights.shape)
            self.bias -= learning_rate * db

            if epoch % 10 == 0:
                print(f"Epoch {epoch+1}/{epochs}, Loss: {loss}")

        return loss_history
```

5. Training

The model is trained using the training data with a defined number of epochs and a fixed learning rate. During training, the model computes forward and backward passes to optimize the weights and bias using gradient descent.

```
# Reshape images for ANN input
train_images_flat = train_images.reshape(train_images.shape[0], -1)
test_images_flat = test_images.reshape(test_images.shape[0], -1)

# Create and train the model
model = NeuralNetwork()
loss_history = model.train(train_images_flat, train_labels.reshape(-1, 1), epochs=10, learning_rate=0.01)

# Ensure the loop runs for all epochs
print("Number of epochs:", len(loss_history))
```

6. Results and Analysis

The training loss is plotted over the epochs to visualize the convergence of the model during training. Additionally, the training and test accuracies are reported to assess the model's performance.

The model achieved promising results, with high accuracy on both the training and test datasets. Specifically, after 10 epochs, the loss was 2.12, and both the train and test accuracies were 1.0. With each execution of the model, the loss is significantly reduced.

```
# Reshape images for ANN input
train_images_flat = train_images.reshape(train_images.shape[0], -1)
test_images_flat = test_images.reshape(test_images.shape[0], -1)

# Create and train the model
model = NeuralNetwork()
loss_history = model.train(train_images_flat, train_labels.reshape(-1, 1),
epochs=10, learning_rate=0.01)

# Ensure the loop runs for all epochs
print("Number of epochs:", len(loss_history))

# Step 4: Evaluation
train_predictions = model.predict(train_images_flat)
test_predictions = model.predict(test_images_flat)

train_accuracy = np.mean((train_predictions >= 0.5) == train_labels)
test_accuracy = np.mean((test_predictions >= 0.5) == test_labels)
```

```
Epoch 1/10, Loss: 0.4930578589017455
Number of epochs: 10
Train Accuracy: 1.0
Test Accuracy: 1.0
```

7. Conclusion

The ANN-based solution for weed detection and classification demonstrated promising results, indicating the potential of using artificial neural networks for automated agricultural tasks. Future work may involve incorporating more sophisticated architectures, leveraging pre-trained models, and exploring advanced techniques to enhance model robustness and accuracy. Overall, this project contributes to advancements in precision agriculture and sustainable crop management.

The provided code and methodology serve as a baseline implementation for weed detection and classification using ANN, offering insights into the application of deep learning techniques in agriculture.

Graphs showcasing Accuracy, Loss and Validation with respect to each Epoch: