ML REPORT - WEED DETECTION IN SOYABEAN CROPS

A.Hemanth(12140230)
B.Rushyendra(12140420)

1 Problem Motivation

Soybean cultivation faces challenges from weed infestations, impacting crop yield, quality, and agricultural costs. Manual weed identification is labor-intensive, prompting the need for automated solutions.

Automated weed detection aims to:

1. Enhance crop management by identifying and managing weed infestations promptly. 2. Optimize resource allocation through targeted interventions based on early weed detection. 3. Improve overall crop health monitoring by enabling proactive measures against potential threats. 4. Reduce production costs and environmental impact associated with manual or chemical control methods.

Our project objectives include developing a robust deep learning model, leveraging pretrained architectures like MobileNetV2 and NasNetMobile, for accurate weed identification. The system aims to provide real-time monitoring, enabling informed decision-making for sustainable soybean farming.

2 Data Preprocessing

- Dataset consists of 4 folders namely broadleaf,grass,soil and soyabean.
- Iterate over each class label in labels.For each class, read image files using glob and loop through them.
- Read each image using OpenCV (cv2.imread).
- Resizing each image to a common size (image_size is set to (100, 110)).Append the resized image (img) to the X list and its corresponding label (i) to the Y list.
- Then we convert the lists X and Y to numpy arrays for compatibility with machine learning frameworks.
- We then create a temporary list Temp_y containing the indices of class labels in the labels list.Used to_categorical from Keras to convert the list of class indices to one-hot encoded format. This is necessary for multi-class classification since we have 4 classes.
- Split the dataset into training and testing sets using train_test_split from scikit-learn. 70% of the data is used for training (X_train and Y_train), and 30% is used for testing (X_test and Y_test)
- Overall, this involves tasks such as resizing images to a consistent size, converting labels to a format suitable for training, and splitting the dataset into training and testing sets to assess model performance. Additionally, converting the data into numpy arrays makes it compatible with deep learning frameworks like TensorFlow and Keras.

3 Modelling

- The neural network consists of convolutional layers followed by max-pooling layers for feature extraction, and then fully connected layers for classification.
- Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(110, 100, 3)): This is the first convolutional layer with 32 filters of size 3x3, ReLU activation function, and input shape of (110, 100, 3), representing the dimensions of the input images (height, width, channels).MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding="valid"): A max-pooling layer with a pool size of 2x2 and a stride of 2, which downsamples the spatial dimensions of the output from the convolutional layer.

- Conv2D(16, kernel_size=(3, 3), activation="relu"): The second convolutional layer with 16 filters of size 3x3 and ReLU activation. MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding="valid"): Another max-pooling layer to downsample the spatial dimensions.
- The third convolutional layer with 8 filters of size 3x3 and ReLU activation followed by max pooling layer of size 2*2 and stride 2*2.
- The fourth convolutional layer with 4 filters of size 3x3 and ReLU activation followed by max pooling layer.
- Then we Flatten the 3D output to 1D before passing it to the fully connected layers.The first fully connected layer with 128 neurons and ReLU activation. The second fully connected layer with 128 neurons and ReLU activation.The output layer with 4 neurons (equal to the number of classes) and softmax activation, which is typical for multi-class classification problems. The softmax function normalizes the output into probability distributions over the classes.
- The model is compiled with categorical crossentropy loss, Adam optimizer, and accuracy as the metric. It is then trained on the training data using the fit method, and the training history is stored for later analysis.
- Finally, the training and validation accuracy and loss curves are plotted to visualize the model's learning progress.
- This custom neural network architecture represents a simple yet effective approach for image classification tasks, providing a good balance between model complexity and computational efficiency.

3.1 Results

- Test accuracy : 98.3 %
- Validation accuracy : 87.44%
- Learning rate : 0.0001

4 Using Autoencoders

4.1 Encoder

- The encoder consists of convolutional layers followed by max-pooling layers, reducing the spatial dimensions of the input image and extracting features.
- The first two Conv2D layers with 20 filters each are followed by max-pooling, reducing the dimensions.
- The next two Conv2D layers with 10 filters each are again followed by max-pooling.
- The last two Conv2D layers with 10 filters each and max-pooling further reduce the dimensions.

4.2 Decoder

- The decoder part of the autoencoder is responsible for reconstructing the input from the compressed representation.
- Conv2D layers with 10 filters each are followed by upsampling layers, increasing the spatial dimensions.
- The last Conv2D layer with 4 filters is followed by an upsampling layer

4.3 Flatten and Fully Connected Layer:

- The flatten layer reshapes the 3D output of the last convolutional layer into a 1D vector.
- The dense layer with softmax activation is used for the final reconstruction and has 6 neurons, matching the input size.

4.4 Results

- Training accuracy : 88.25%
- Test accuracy : 87.54%

5 Conclusion

- We got our best model as custom neural network which gave 98.3% training accuracy and 87.44% test accuracy.We have used 4 models including phase 1 models and we got the best as mentioned above.All the graphs and code are given in notebooks submitted.