Ensemble Deep Learning for Enhanced Crop and Weed Classification in Agriculture

By

Varnika Milind Mulay (MST03-0060)

Submitted to

Meta Scifor Technologies

Under guidance of

Urooj Khan

Contents

Abstract

This paper investigates the categorization of crops and weeds using state-of-the-art deep learning methods, namely Convolutional Neural Networks (CNNs) and ResNet50, in addition to an ensemble approach that combines both models. Whereas ResNet50 uses residual learning to increase accuracy with deeper structures, CNN uses hierarchical features extracted from images. The effectiveness of an ensemble technique, which averages CNN and ResNet50 predictions, is assessed in terms of how well it improves classification performance. The results show that the ensemble approach works better than the individual models, with an accuracy of 95% achieved as opposed to 94% for CNN and 70% for ResNet50 alone. This method demonstrates how well several deep learning approaches may be used to improve classification jobs.

The study also compares computational efficiency and resource requirements, assesses the effect of various preprocessing techniques on model performance, examines the scalability of these models for large-scale agricultural deployments, and examines the robustness of the models under varied environmental conditions. All things considered, the results provide insightful information about how well these models work in real agricultural settings and how feasible it is to put them into practice.

The results of this study should greatly advance the creation of increasingly complex and automated agricultural systems, ultimately enhancing agricultural output and sustainability.

Keywords: Agriculture, crop, weed, Machine Learning, Deep Learning, image classification, CNN, ResNet50, Ensemble method, transfer learning.

Abbreviations:

CNN – Convolutional Neural Networks

ResNet50 – Residual Neural Networks

ReLU – Rectified Linear Unit

JSON – JavaScript Object Notation

Introduction

Providing the foundation for the development and subsistence of many nations, agriculture is an essential component of the world economy. Accurately identifying and classifying weeds and crops is a major challenge in agriculture, as it is necessary for both maximal crop yield and efficient weed management. The manual labor and specialized expertise used in traditional crop and weed classification systems are very labor-intensive, time-consuming, and prone to human error. Automating this important process with more accuracy and efficiency is now possible thanks to the development of advanced machine learning techniques, especially deep learning.

In agricultural applications as well as other image classification tasks, Convolutional Neural Networks (CNNs) have proven to be an effective tool. CNNs are quite good at differentiating between weeds and crops because they can learn hierarchical information from photos. Apart from CNNs, deeper architectures like ResNet50, which employ residual learning to alleviate the vanishing gradient problem, have shown better results in many image classification tasks.

In order to capitalize on the advantages of both models, we investigate the use of CNNs and ResNet50 for the categorization of crops and weeds in this study. Moreover, we suggest an ensemble technique that averages the predictions made by CNN and ResNet50. We believe that this strategy can improve classification accuracy by incorporating complementing information from both models.

This work aims to accomplish three goals: (1) create and assess CNN and ResNet50 models for crop and weed classification; (2) examine performance gains made possible by the ensemble approach; and (3) offer insights into the usefulness of applying such models in actual agriculture.

By achieving these goals, this study adds to the expanding corpus of research on the use of deep learning in agriculture and provides a solid method for classifying crops and weeds that has the potential to completely transform weed control techniques. It is anticipated that the results of this study will open the door for increasingly sophisticated and automated agricultural systems, which will ultimately boost agricultural sustainability and output.

Technology Used

1. **Programming Language:**
    - Python: The primary programming language used for implementing machine learning models and data processing tasks.

2. **Libraries and Frameworks:**
    - TensorFlow and Keras:
        - TensorFlow: An open-source machine learning framework developed by Google, used for building and training neural networks.
        - Keras: A high-level API of TensorFlow used for building and training deep learning models.
        - Layers Used: Conv2D, MaxPooling2D, Flatten, Dense, Dropout, GlobalAveragePooling2D.
        - Models: Sequential, ResNet50 (pre-trained on ImageNet dataset).
    - Scikit-learn:
        - Metrics and Utilities: train_test_split, accuracy_score, classification_report, confusion_matrix.
    - Pandas: For data manipulation and analysis, specifically for handling the dataset of image filenames and labels.
    - NumPy: For numerical computations and array manipulations.
    - Matplotlib and Seaborn: For data visualization, including plotting training/validation accuracy and loss, as well as confusion matrices.
    - OpenCV: For image processing tasks.
    - Json: For reading and processing annotation files in JSON format.

3. **Tools and Environments:**
    - Google Colab: A cloud-based Jupyter notebook environment that provides free access to GPUs for training deep learning models.
    - Google Drive: Used for storing the dataset, accessed via Google Colab.
    - Operating System: Linux-based environment provided by Google Colab.

4. **Hardware:**
    - GPU: Leveraged through Google Colab for accelerating the training of deep learning models.

5. **Data Handling:**
    - Loading and Preprocessing Functions: Custom functions to load images from the dataset, resize them, and normalize pixel values.

<u>Dataset Description</u>

The dataset utilized in this study includes 1300 carefully categorized photos of different types of weeds and sesame crops. The color format used to display these pictures is 512x512. It was released in the year 2020 and has been extensively used for image classification purposes in the agricultural domain [1].

The dataset is in the form of two separate folders: one containing the images and the other containing the annotations in JSON format.

Methodology

The methodology followed in the study is explained in this section.

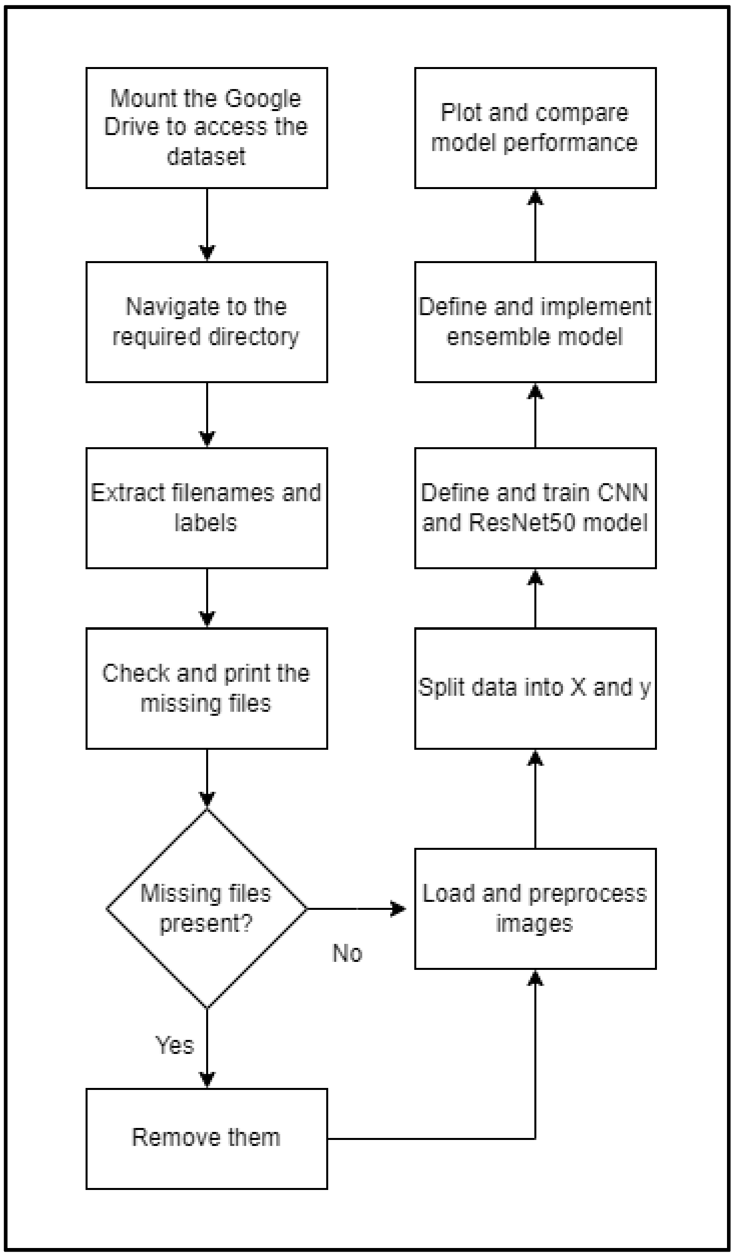

Mount the Google Drive to access the dataset
Navigate to the required directory
Extract filenames and labels
Check and print the missing files
Missing files present?
Yes
Remove them
No
Load and preprocess images
Split data into X and y
Define and train CNN and ResNet50 model
Define and implement ensemble model
Plot and compare model performance

Figure 1. Workflow of the study

This research follows the steps as mentioned in **Figure 1.** The detailed working of the same is explained below:

1. **Importing necessary libraries**

The first step is to import the necessary libraries required for the implementation of this project. Some of the major libraries are:

- os: Operating system interactions
- pandas: Data manipulation and analysis
- json: Parsing JSON files
- matplotlib.pyplot: Plotting graphs and visualizing data
- seaborn: Statistical graphics
- sklearn.model_selection: Data splitting
- sklearn.metrics: Model evaluation
- tensorflow & Keras:
    - Image preprocessing: tensorflow.keras.preprocessing.image
    - Model building and loading: tensorflow.keras.models
    - Neural network layers: tensorflow.keras.layers
    - Pre-trained models: tensorflow.keras.applications
    - Model training utilities: tensorflow.keras.callbacks

2. Mount Google Drive

After loading the necessary libraries, the next step is to mount the Google Drive i.e., allowing Google Colab to access the files present in the Google Drive. This step is necessary as the dataset used in this study is stored on the Drive. Following this step, list the folders present in the directory.

3. Extract the filenames and labels

As mentioned earlier, the dataset is in the form of two folders each containing the images and annotations respectively. As these annotations are present in json format, it is necessary to extract the filenames and labels from the same for further processing tasks. Hence, the next step is to iterate over the annotations folder extract the required filenames and labels and store them in a data frame.

Another important step in this is to convert the extracted labels into binary format. As it is a binary classification project, the labels are crop and weed which are converted into 1 and 0 respectively.

4. Identify the missing filenames and remove them

After successfully creating a data frame containing the filenames and their corresponding labels, the next step is to identify any missing files from the same. To achieve this purpose, the following process deals with creating a list to store the missing filenames. Then, the code iterates over the filenames mentioned in the data frame and checks whether the file exists for the specified file path and is considered "missing" if it doesn't exist. The missing files are then removed from the data frame.

5. Create a function to load and preprocess the images

In this step, a function has been created to read image files from the given directory, resize them to the specified target size (256 x 256), and convert them to NumPy arrays. It also handles missing files and reports any errors encountered during loading.

6. Split data into X and y

The next step is to split the data into X and y where X represents the images and y represents the labels obtained from the data frame created earlier. The images stored in X are normalized by dividing the pixel values by 255.0 to scale the pixel values to a range of 0 to 1 to prevent any bias and to achieve better model performance.

7. Split the data into training and validation

In this step, the data is divided into a training set and validation set with a split ratio of 70:20 i.e., 70% of the data is used for training and 20% is used for validation purposes. With this, the number of images in each set came to be:

Training set – 1040 images

Validation set – 260 images

8. Define a Convolutional Neural Network (CNN) model

Subsequently, the next step is to create a Convolutional Neural Network (CNN) model by defining the layers required for this study.

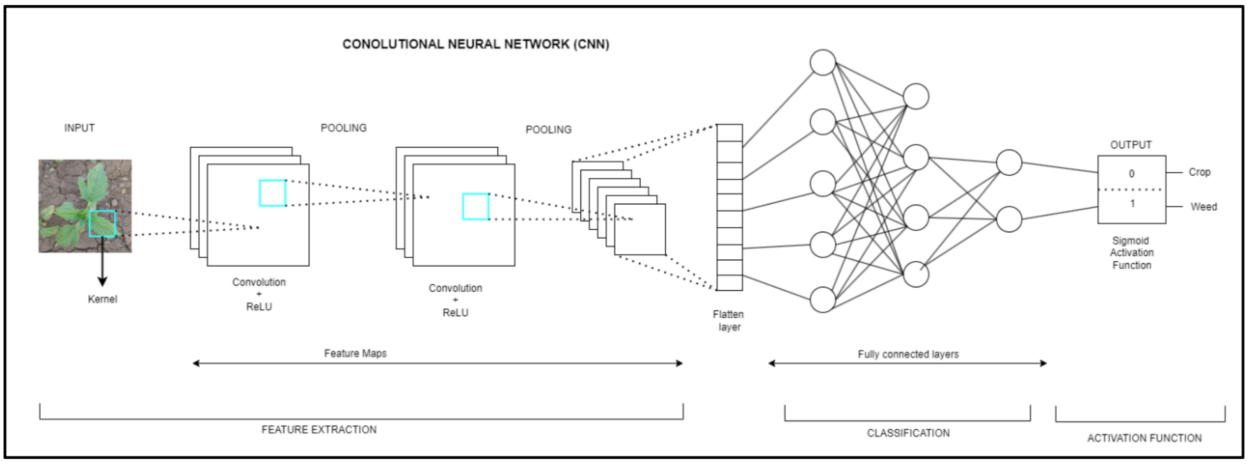


Figure 2. CNN architecture

A CNN is a deep learning model that is intended to handle input that resembles a grid, like images. It makes use of fully connected layers for classification or regression, pooling layers to minimize spatial dimensions, and convolutional layers to identify features. CNNs are very useful for problems involving the recognition of images and videos because they automatically determine the spatial hierarchies of characteristics from data [3].

As shown in **Figure 2** the layers used are:

- Convolutional Layers: These layers apply convolution operations to the input image, using multiple filters to extract different features such as edges, textures, and patterns. The activation function RELU introduces non-linearity.

- Pooling Layers: Max pooling layers downsample the feature maps, reducing their dimensions and computational complexity while retaining important features. This also helps in making the model invariant to small translations in the input image.
- Flatten Layer: The flattening layer prepares the data for the fully connected layers by converting the 2D feature maps into a 1D vector.
- Fully Connected Layers: The fully connected layers perform decision-making and high-level reasoning based on the features extracted by the above layers. The ReLU activation function is used to help in learning complex patterns.

  $$ReLU(x) = \max(0, x) \quad (1)$$

  Equation (1) defines the ReLU activation function where x is the input to the ReLU activation function and it outputs x if x is positive, and 0 otherwise.
- Dropout Layer: Dropout is a regularization technique that prevents overfitting by randomly setting a fraction of the input units to 0 at each update during training time, forcing the network to learn more robust features.
- Output Layer: The final dense layer with a single neuron and sigmoid activation function outputs a probability value for binary classification, indicating the likelihood of the input belonging to a particular class.

  $$\sigma(x) = \frac{1}{1+ e^{-x}} \quad (2)$$

  Equation (2) represents the sigmoid activation function in which x is the input provided to it.

9. Compile the model

After successfully creating a CNN model, the next step is to compile it. The CNN model is compiled with the following parameters:

- Loss Function (Binary Cross-Entropy): This function measures the difference between actual and predicted values for binary classification, guiding the model to minimise errors.
- Optimiser (Adam): This function effectively adjusts learning rates based on gradients, improving training speed and performance.
- Metric (Accuracy): This function assesses the model's accuracy by calculating the ratio of correct predictions to total predictions.

10. Model training

The model is trained on the training data by implementing Early Stopping criteria.

- Early Stopping: During training, this method keeps an eye on the validation loss. If no progress is observed for five consecutive epochs, it stops training and resets the model to its optimal weights to avoid overfitting.
- Model Training: Using the training data, the CNN model is trained for 10 epochs before being validated using the validation set. The purpose of early stopping is to maximise training efficiency and duration.

11. Model evaluation

The model is then evaluated on the validation set to obtain its validation loss and accuracy. This step is crucial to observe the model's performance and to verify whether the model is performing well.

The model is also evaluated using the classification report which uses functions like Precision, Recall, F1-score, and accuracy to provide a concise report of the model's behavior.

- Precision: The ratio of predicted true positives to the sum of true positives and false positives (total predicted positives). It measures how many of the predicted positive cases are actually positive as seen in (3).

$$Precision = \frac{TP}{TP+FP} \tag{3}$$

- Recall: The ratio of predicted true positives to the sum of true positives and false negatives (total actual positives). It measures how many of the actual positive cases are correctly identified as seen in (4).

$$Recall = \frac{TP}{TP+FN} \tag{4}$$

- F1-Score: The harmonic mean of precision and recall, which balances both metrics and provides a single measure of model performance as seen in (5).

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision+Recall} \tag{5}$$

- Accuracy: The ratio of correct predictions i.e., the sum of true positives and true negatives to the total number of predictions as seen in (6).

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{6}$$

where,
TP = True Positive
FP = False Positive
TN = True Negative
FN = False Negative

The model is also evaluated using the confusion matrix which gives a matrix containing the TP, TN, FP, and FN values.

- True Positives (TP): The number of correctly predicted positive values.
- False Positives (FP): The number of incorrectly predicted positive values i.e., actual negative values predicted as positive.
- True Negatives (TN): The number of correctly predicted negative values.

- False Negatives (FN): The number of incorrectly predicted negative values i.e., actual positive values predicted as negative.

12. Plot the obtained results

Lastly, plot the obtained validation loss and validation accuracy values against the training loss and accuracy values to observe the variations in the same and also the trends i.e., whether the loss and accuracy values are increasing or decreasing with change in epochs.

13. Define the Residual Neural Networks (ResNet50) model

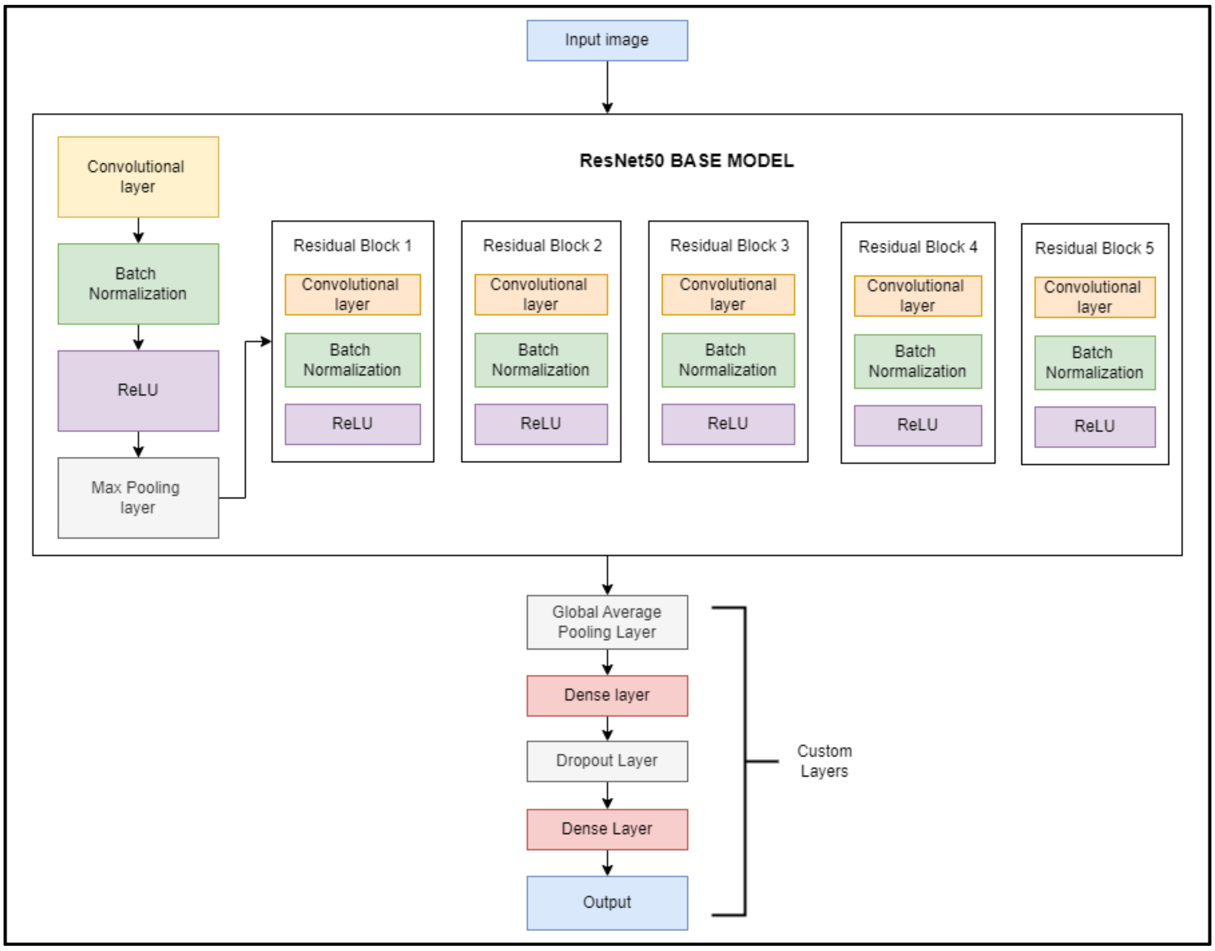


Figure 3. ResNet50 model architecture

Figure 3 represents the ResNet50 model architecture used in this study to perform binary image classification on crop and weed data.

ResNet50 is a kind of CNN that makes training deeper networks easier by utilizing shortcut connections and residual learning. Its 50 layers incorporate residual blocks that aid in resolving the vanishing gradient issue, enabling the network to efficiently learn more intricate features. It uses ImageNet's pre-trained weights, which enable it to perform a wide range of computer vision tasks accurately.

In this, the first step is to load the ResNet50 model without the top layers. The model is used with weights pre-trained on the ImageNet dataset which has millions of images and thousands of classes. This comes under the method of transfer learning, and it applies features that have already been learned to perform new tasks. In this case, these features are used for crop and weed classification and hence, the top classification layers are excluded from the ResNet50 model.

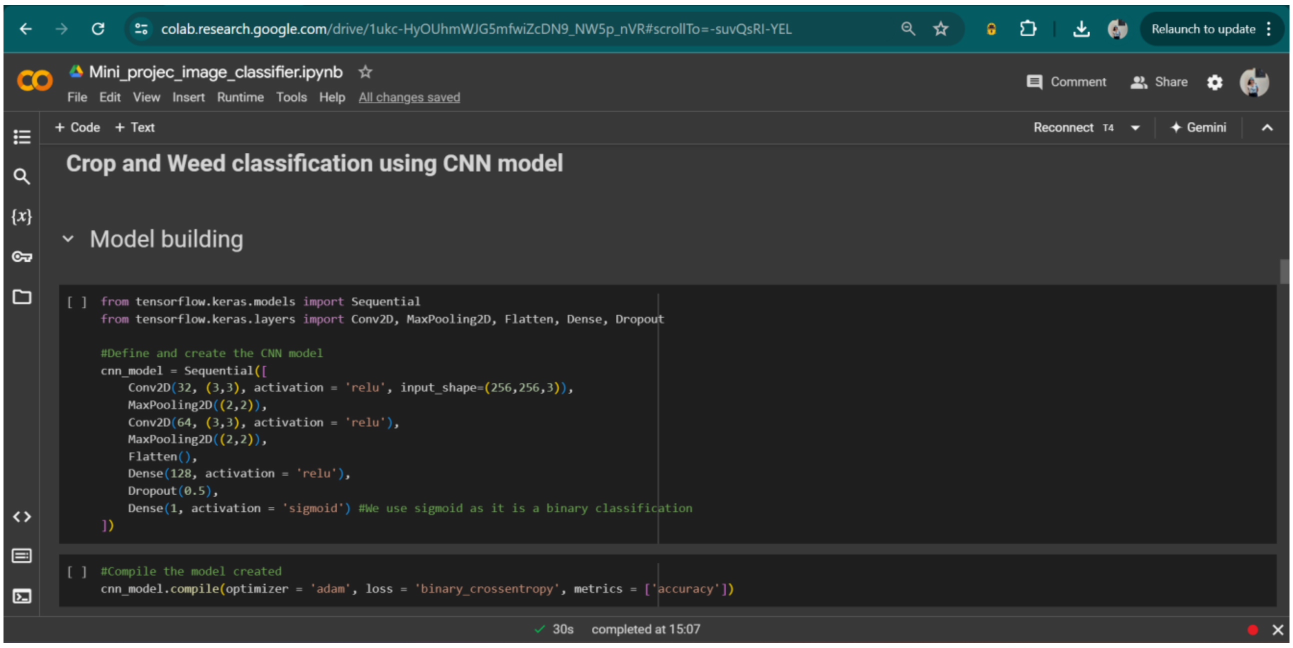
The ResNet50 model layers are also frozen to focus on training new layers and retain the learned characteristics by not changing the weights of the pre-trained ResNet50 layers during training [5].

14. **Repeat the above steps for the ResNet50 model**
15. **Create an ensemble model**

An ensemble model is created by averaging the model predictions of CNN and ResNet50 to achieve higher performance and accuracy [2, 4].

Code Snippet

1. CNN model



colab.research.google.com/drive/1ukc-HyOUhmWJG5mfwiZcDN9_NW5p_nVR#scrollTo=-suvQsRI-YEL
Relaunch to update
Mini_projec_image_classifier.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect T4
Gemini

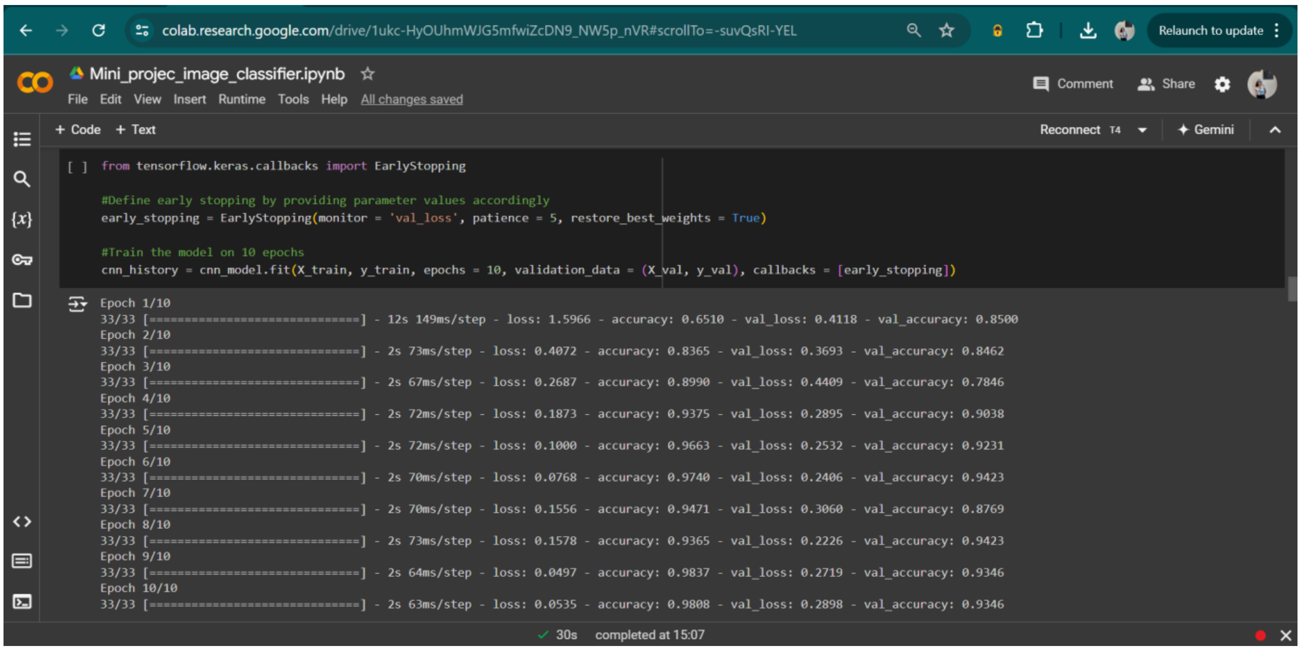
Crop and Weed classification using CNN model

Model building

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

#Define and create the CNN model
cnn_model = Sequential([
    Conv2D(32, (3,3), activation = 'relu', input_shape=(256,256,3)),
    MaxPooling2D((2,2)),
    Conv2D(64, (3,3), activation = 'relu'),
    MaxPooling2D((2,2)),
    Flatten(),
    Dense(128, activation = 'relu'),
    Dropout(0.5),
    Dense(1, activation = 'sigmoid') #We use sigmoid as it is a binary classification
])
```

```
#Compile the model created
cnn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

✓ 30s completed at 15:07



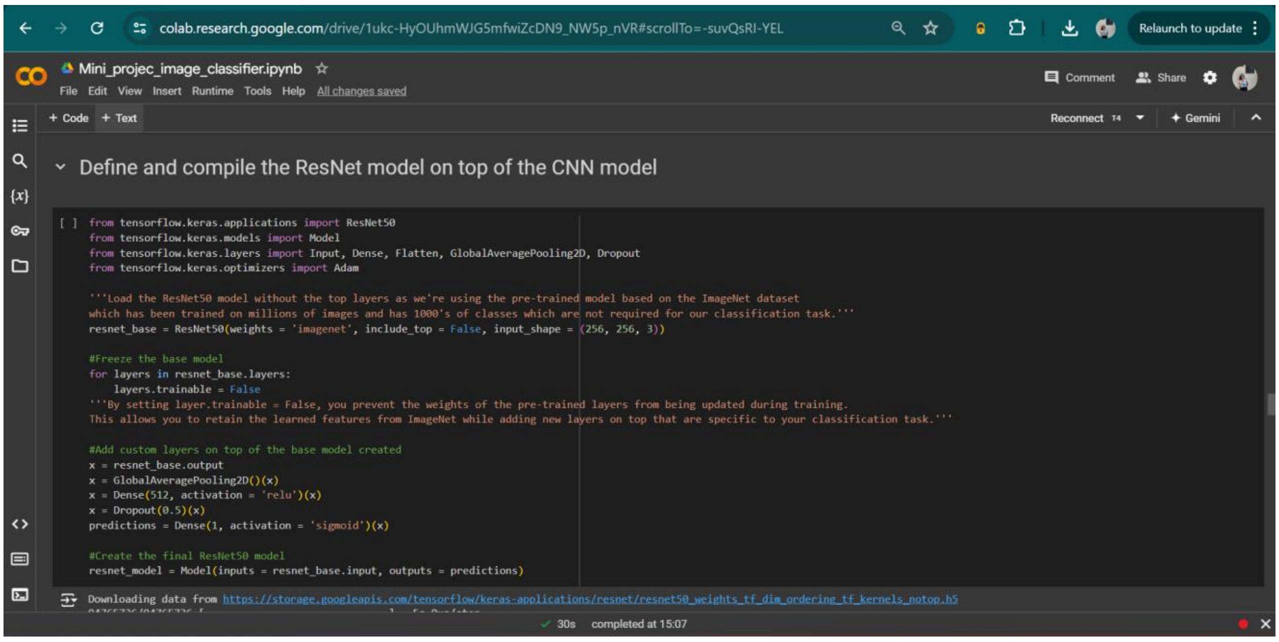
colab.research.google.com/drive/1ukc-HyOUhmWJG5mfwiZcDN9_NW5p_nVR#scrollTo=-suvQsRI-YEL
Relaunch to update
Mini_projec_image_classifier.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect T4
Gemini

```
from tensorflow.keras.callbacks import EarlyStopping

#Define early stopping by providing parameter values accordingly
early_stopping = EarlyStopping(monitor = 'val_loss', patience = 5, restore_best_weights = True)

#Train the model on 10 epochs
cnn_history = cnn_model.fit(X_train, y_train, epochs = 10, validation_data = (X_val, y_val), callbacks = [early_stopping])
```

```
Epoch 1/10
33/33 [==============================] - 12s 149ms/step - loss: 1.5966 - accuracy: 0.6510 - val_loss: 0.4118 - val_accuracy: 0.8500
Epoch 2/10
33/33 [==============================] - 2s 73ms/step - loss: 0.4072 - accuracy: 0.8365 - val_loss: 0.3693 - val_accuracy: 0.8462
Epoch 3/10
33/33 [==============================] - 2s 67ms/step - loss: 0.2687 - accuracy: 0.8990 - val_loss: 0.4409 - val_accuracy: 0.7846
Epoch 4/10
33/33 [==============================] - 2s 72ms/step - loss: 0.1873 - accuracy: 0.9375 - val_loss: 0.2895 - val_accuracy: 0.9038
Epoch 5/10
33/33 [==============================] - 2s 72ms/step - loss: 0.1000 - accuracy: 0.9663 - val_loss: 0.2532 - val_accuracy: 0.9231
Epoch 6/10
33/33 [==============================] - 2s 70ms/step - loss: 0.0768 - accuracy: 0.9740 - val_loss: 0.2406 - val_accuracy: 0.9423
Epoch 7/10
33/33 [==============================] - 2s 70ms/step - loss: 0.1556 - accuracy: 0.9471 - val_loss: 0.3060 - val_accuracy: 0.8769
Epoch 8/10
33/33 [==============================] - 2s 73ms/step - loss: 0.1578 - accuracy: 0.9365 - val_loss: 0.2226 - val_accuracy: 0.9423
Epoch 9/10
33/33 [==============================] - 2s 64ms/step - loss: 0.0497 - accuracy: 0.9837 - val_loss: 0.2719 - val_accuracy: 0.9346
Epoch 10/10
33/33 [==============================] - 2s 63ms/step - loss: 0.0535 - accuracy: 0.9808 - val_loss: 0.2898 - val_accuracy: 0.9346
```

✓ 30s completed at 15:07

2. ResNet50 model

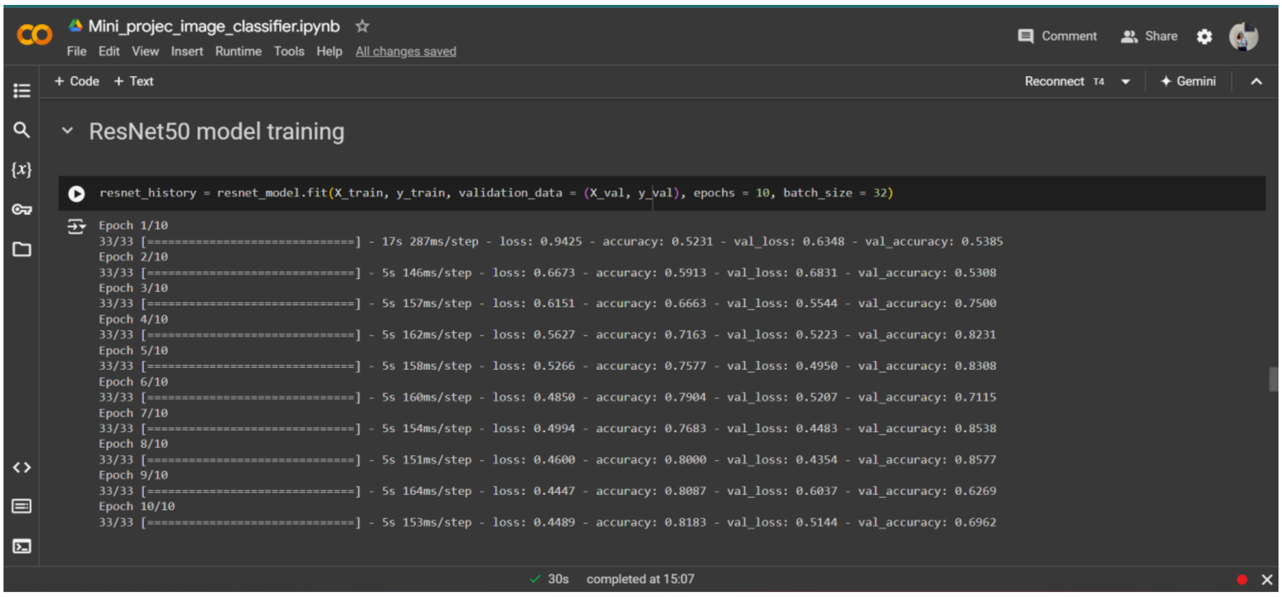

colab.research.google.com/drive/1ukc-HyOUhmWJG5mfwiZcDN9_NW5p_nVR#scrollTo=-suvQsRI-YEL
Relaunch to update
Mini_projec_image_classifier.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect T4
Gemini

Define and compile the ResNet model on top of the CNN model

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, Flatten, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam

'''Load the ResNet50 model without the top layers as we're using the pre-trained model based on the ImageNet dataset
which has been trained on millions of images and has 1000's of classes which are not required for our classification task.'''
resnet_base = ResNet50(weights = 'imagenet', include_top = False, input_shape = (256, 256, 3))

#Freeze the base model
for layers in resnet_base.layers:
    layers.trainable = False
'''By setting layer.trainable = False, you prevent the weights of the pre-trained layers from being updated during training.
This allows you to retain the learned features from ImageNet while adding new layers on top that are specific to your classification task.'''

#Add custom layers on top of the base model created
x = resnet_base.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation = 'relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation = 'sigmoid')(x)

#Create the final ResNet50 model
resnet_model = Model(inputs = resnet_base.input, outputs = predictions)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
```

✓ 30s completed at 15:07


Mini_projec_image_classifier.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
Reconnect T4
Gemini

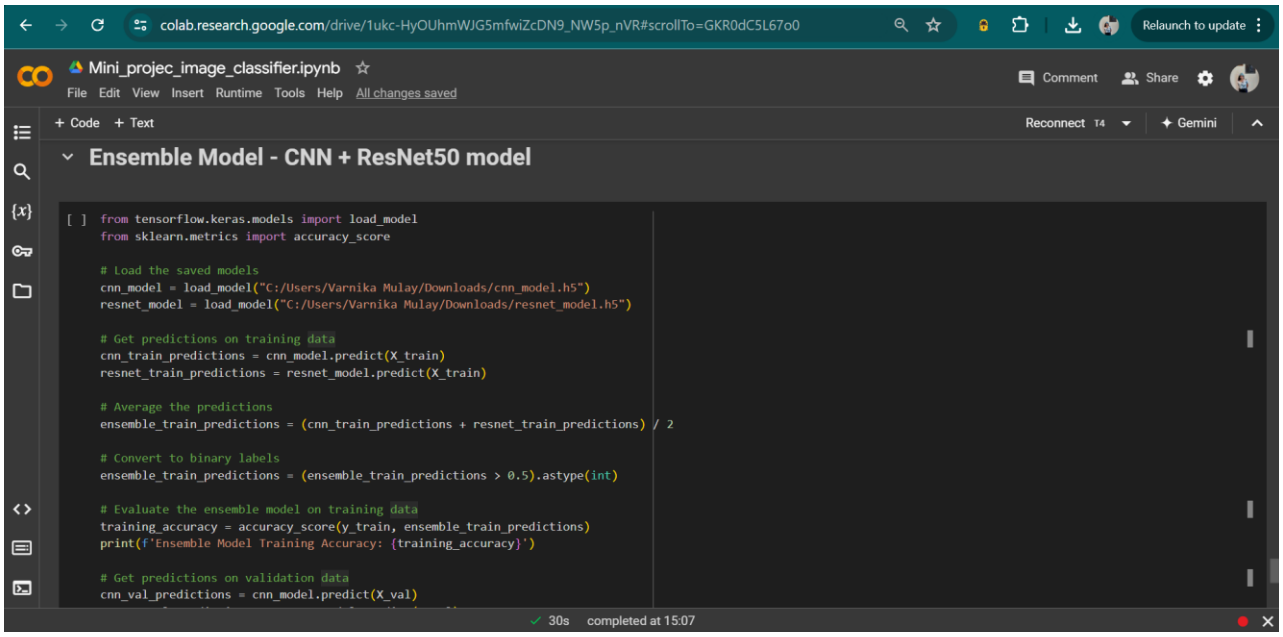
ResNet50 model training

```
resnet_history = resnet_model.fit(X_train, y_train, validation_data = (X_val, y_val), epochs = 10, batch_size = 32)
```

```
Epoch 1/10
33/33 [==============================] - 17s 287ms/step - loss: 0.9425 - accuracy: 0.5231 - val_loss: 0.6348 - val_accuracy: 0.5385
Epoch 2/10
33/33 [==============================] - 5s 146ms/step - loss: 0.6673 - accuracy: 0.5913 - val_loss: 0.6831 - val_accuracy: 0.5308
Epoch 3/10
33/33 [==============================] - 5s 157ms/step - loss: 0.6151 - accuracy: 0.6663 - val_loss: 0.5544 - val_accuracy: 0.7500
Epoch 4/10
33/33 [==============================] - 5s 162ms/step - loss: 0.5627 - accuracy: 0.7163 - val_loss: 0.5223 - val_accuracy: 0.8231
Epoch 5/10
33/33 [==============================] - 5s 158ms/step - loss: 0.5266 - accuracy: 0.7577 - val_loss: 0.4950 - val_accuracy: 0.8308
Epoch 6/10
33/33 [==============================] - 5s 160ms/step - loss: 0.4850 - accuracy: 0.7904 - val_loss: 0.5207 - val_accuracy: 0.7115
Epoch 7/10
33/33 [==============================] - 5s 154ms/step - loss: 0.4994 - accuracy: 0.7683 - val_loss: 0.4483 - val_accuracy: 0.8538
Epoch 8/10
33/33 [==============================] - 5s 151ms/step - loss: 0.4600 - accuracy: 0.8000 - val_loss: 0.4354 - val_accuracy: 0.8577
Epoch 9/10
33/33 [==============================] - 5s 164ms/step - loss: 0.4447 - accuracy: 0.8087 - val_loss: 0.6037 - val_accuracy: 0.6269
Epoch 10/10
33/33 [==============================] - 5s 153ms/step - loss: 0.4489 - accuracy: 0.8183 - val_loss: 0.5144 - val_accuracy: 0.6962
```

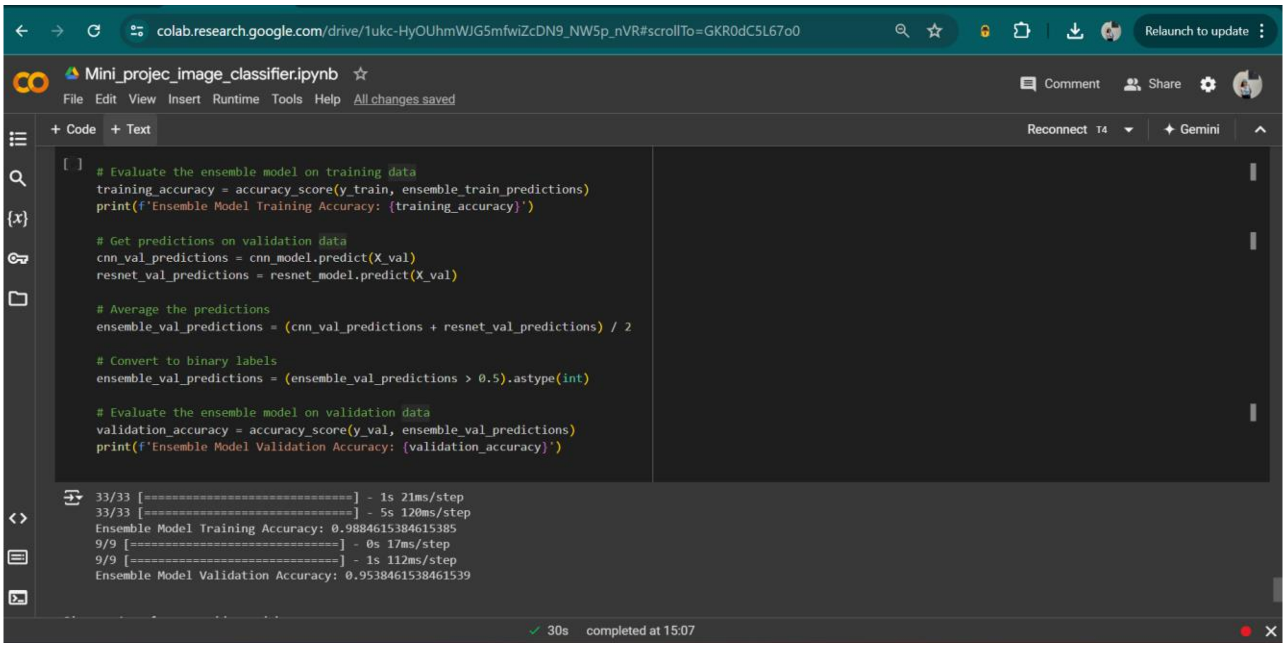
✓ 30s completed at 15:07

3. Ensemble model


colab.research.google.com/drive/1ukc-HyOUhmWJG5mfwiZcDN9_NW5p_nVR#scrollTo=GKR0dC5L67o0
Relaunch to update
Mini_projec_image_classifier.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect T4
Gemini
30s completed at 15:07

Ensemble Model - CNN + ResNet50 model

```
from tensorflow.keras.models import load_model
from sklearn.metrics import accuracy_score

# Load the saved models
cnn_model = load_model("C:/Users/Varnika Mulay/Downloads/cnn_model.h5")
resnet_model = load_model("C:/Users/Varnika Mulay/Downloads/resnet_model.h5")

# Get predictions on training data
cnn_train_predictions = cnn_model.predict(X_train)
resnet_train_predictions = resnet_model.predict(X_train)

# Average the predictions
ensemble_train_predictions = (cnn_train_predictions + resnet_train_predictions) / 2

# Convert to binary labels
ensemble_train_predictions = (ensemble_train_predictions > 0.5).astype(int)

# Evaluate the ensemble model on training data
training_accuracy = accuracy_score(y_train, ensemble_train_predictions)
print(f'Ensemble Model Training Accuracy: {training_accuracy}')

# Get predictions on validation data
cnn_val_predictions = cnn_model.predict(X_val)
```


colab.research.google.com/drive/1ukc-HyOUhmWJG5mfwiZcDN9_NW5p_nVR#scrollTo=GKR0dC5L67o0
Relaunch to update
Mini_projec_image_classifier.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
+ Code + Text
Reconnect T4
Gemini
30s completed at 15:07

```
# Evaluate the ensemble model on training data
training_accuracy = accuracy_score(y_train, ensemble_train_predictions)
print(f'Ensemble Model Training Accuracy: {training_accuracy}')

# Get predictions on validation data
cnn_val_predictions = cnn_model.predict(X_val)
resnet_val_predictions = resnet_model.predict(X_val)

# Average the predictions
ensemble_val_predictions = (cnn_val_predictions + resnet_val_predictions) / 2

# Convert to binary labels
ensemble_val_predictions = (ensemble_val_predictions > 0.5).astype(int)

# Evaluate the ensemble model on validation data
validation_accuracy = accuracy_score(y_val, ensemble_val_predictions)
print(f'Ensemble Model Validation Accuracy: {validation_accuracy}')
```

```
33/33 [==============================] - 1s 21ms/step
33/33 [==============================] - 5s 120ms/step
Ensemble Model Training Accuracy: 0.9884615384615385
9/9 [==============================] - 0s 17ms/step
9/9 [==============================] - 1s 112ms/step
Ensemble Model Validation Accuracy: 0.9538461538461539
```

Results and Discussion

This section provides a detailed explanation of the obtained results and observations.

Table 1. Model comparison based on accuracy and loss values

Models	Training accuracy (%)	Training Loss (%)	Validation Accuracy (%)	Validation Loss (%)
CNN	98.07	5.35	93.46	28.97
ResNet50	81.82	44.89	69.61	51.44
Ensemble	98.84	-	95.38	-

Table 1 represents the training and validation loss and accuracy values for the CNN and ResNet50 models used in this study. It can be observed that the CNN model performs better than the ResNet50 model as it gives a higher accuracy of 98% and provides a lower loss value of 5.35% as compared to the values obtained by the ResNet50 model. Both the models were iterated on 10 epochs with a batch size of 32.

Hence, it can be said that the CNN model is preferred over the ResNet50 model for crop and weed classification tasks.

The ensemble model provides a training accuracy of almost 99% which is higher than the individual accuracies of the CNN and ResNet50 models. It also provides a validation accuracy of about 95% which is also higher than CNN and ResNet50 models.

Hence, it can be said that the ensemble model performs better than the other two models and is the most suitable choice for this project.

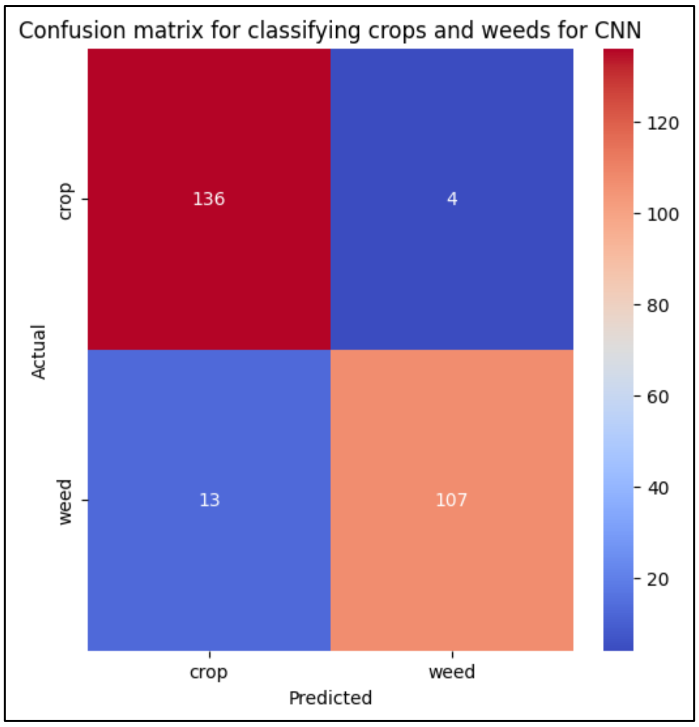


Figure 4. Confusion matrix for CNN model

Figure 4 represents the confusion matrix for the CNN model. It can be seen that:

- True Negatives - TN: The model correctly predicted 136 weed images as weed.
- False Positives - FP: The model incorrectly predicted 4 weed images as crop.
- False Negatives - FN: The model incorrectly predicted 13 crop images as weed.
- True Positives - TP: The model correctly predicted 107 crop images as crop.

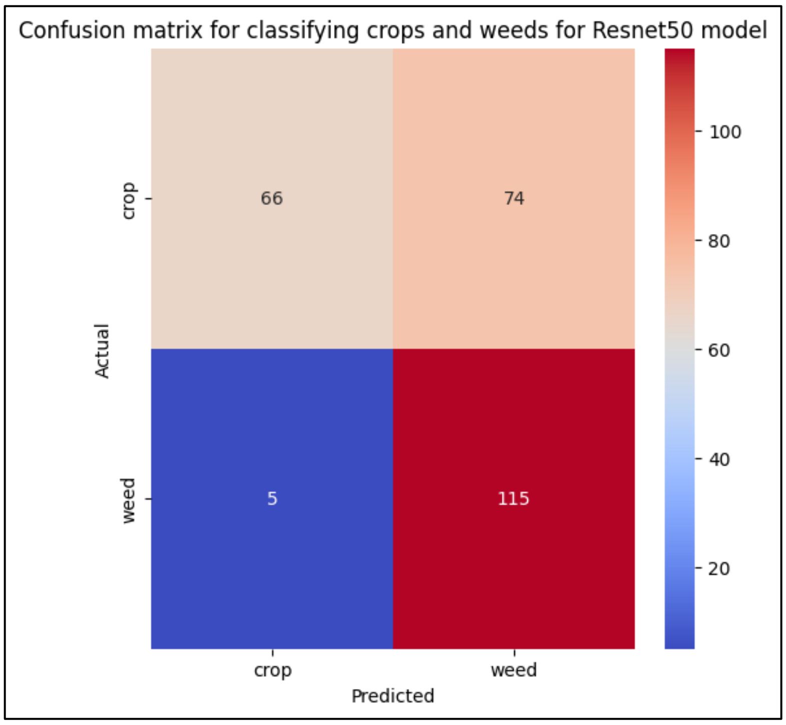


Figure 5. Confusion matrix for ResNet50 model

Figure 5 represents the confusion matrix for the CNN model. It can be seen that:

- True Negatives - TN: The model correctly predicted 66 weed images as weed.
- False Positives - FP: The model incorrectly predicted 74 weed images as crop.
- False Negatives - FN: The model incorrectly predicted 5 crop images as weed.
- True Positives - TP: The model correctly predicted 115 crop images as crop.

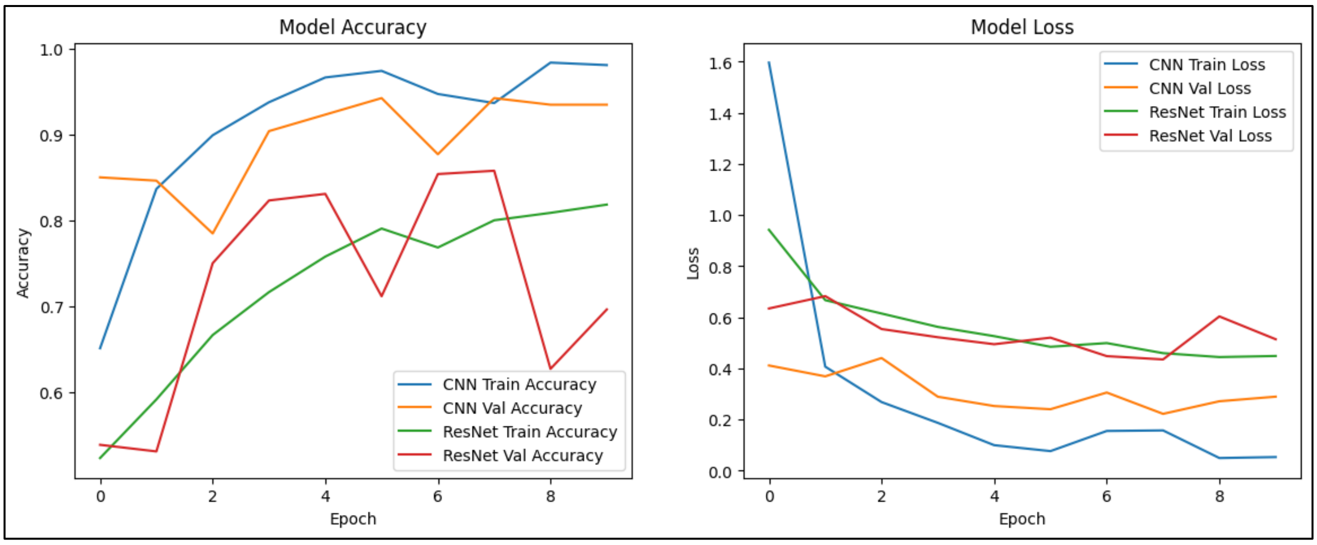


Figure 6. CNN and ResNet50 model evaluation

Figure 6 depicts the comparison between the model performances by plotting their accuracies and loss values. Below are the detailed observations of the same:

CNN Model:

- The CNN model's training accuracy rises quickly, approaching 100% by epoch 10. After epoch 5, the CNN model's validation accuracy is more consistent, averaging 90% with just minor variations.
- The CNN model's training loss shows a rapid and continuous drop, suggesting that the training data are being effectively learnt. The CNN model shows strong generalisation to the validation data when its validation loss first drops and then stabilises at a low value.

ResNet50 Model:

- The ResNet50 model's training accuracy rises gradually, peaking at 80% by epoch 10. The ResNet50 model's validation accuracy varies somewhat, however it usually stabilises between 80 and 85 percent with notable variations.
- The ResNet50 model's training loss gradually drops but stays larger than the CNN model's, indicating a slower rate of learning. The ResNet50 model's validation loss varies considerably, suggesting possible problems with stability and generalisation throughout training.

Conclusion

This work investigated the advantages of an ensemble strategy that integrates both models by developing and evaluating Convolutional Neural Networks (CNNs) and ResNet50 models for the categorization of crops and weeds. While the ResNet50 model showed steady but marginally lower accuracy, the CNN model showed fast learning with excellent training accuracy. Nonetheless, the enhanced accuracy and resilience of the ensemble approach, which combined the forecasts from the two models, made it perform better than the individual models.

The outcomes highlight the possibility of utilizing various deep-learning methods to improve categorization assignments in farming environments. Along with bettering classification performance, the ensemble model also offered increased stability and generalization in a variety of environmental settings. This suggests that integrating CNN and ResNet50 models can result in crop and weed classification methods that are more dependable and efficient.

The study also emphasized the importance of preprocessing methods, computational effectiveness, and model scalability, offering thorough insights into their practical application in actual agricultural settings. By correctly differentiating between crops and weeds, these sophisticated deep learning models—especially when utilized in an ensemble—can greatly help precision agriculture, leading to improved management techniques and higher agricultural yield.

In summary, the amalgamation of CNN, ResNet50, and ensemble approaches presents a formidable methodology for enhancing classification problems in the agricultural domain, exhibiting both pragmatic relevance and prospects for forthcoming progress in the area.

References

[1] Panara, U., Pandya, R., Rayja, M. (2020). Crop and Weed Detection data with bounding boxes. Kaggle. Retrieved from https://www.kaggle.com/datasets/ravirajsinh45/crop-and-weed-detection-data-with-bounding-boxes.

[2] V. S. Babu and N. Venkatram, "Ensemble Learning for Weed Detection in Soybean Crop," 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2024, pp. 867-871, doi: 10.23919/INDIACom61295.2024.10498220.

[3] Rozendo, G.B., Roberto, G.F., do Nascimento, M.Z., Alves Neves, L., Lumini, A. (2024). Weeds Classification with Deep Learning: An Investigation Using CNN, Vision Transformers, Pyramid Vision Transformers, and Ensemble Strategy. In: Vasconcelos, V., Domingues, I., Paredes, S. (eds) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. CIARP 2023. Lecture Notes in Computer Science, vol 14469. Springer, Cham. https://doi.org/10.1007/978-3-031-49018-7_17

[4] Asad, Muhammad Hamza, Saeed Anwar, and Abdul Bais. "Improved Crop and Weed Detection with Diverse Data Ensemble Learning in Agriculture." arXiv preprint arXiv:2310.01055 (2023).

[5] Gourisaria, M.K., Sahoo, V.K., Sahoo, B., Sarangi, P.P., Singh, V. (2023). Empirical Study of Multi-class Weed Classification Using Deep Learning Network Through Transfer Learning. In: Roy, S., Sinwar, D., Dey, N., Perumal, T., Tavares, J.M.R.S. (eds) Innovations in Computational Intelligence and Computer Vision. ICICV 2022. Lecture Notes in Networks and Systems, vol 680. Springer, Singapore. https://doi.org/10.1007/978-981-99-2602-2_1