Agris on-line Papers in Economics and Informatics

Volume XV Number 1, 2023

Developing an Efficient System with Mask R-CNN for Agricultural Applications

Brahim Jabir, Khalid El Moutaouakil, Noureddine Falih

LIMATI, Sultan Moulay Slimane University, Beni Mellal, Morocco

Abstract

In order to meet the world's demand for food production, farmers and producers have improved and increased their agricultural production capabilities, leading to a profit acceleration in the field. However, this growth has also caused significant environmental damage due to the widespread use of herbicides. Weeds competing with crops result in lower crop yields and a 30% increase in losses. To rationalize the use of these herbicides, it would be more effective to detect the presence of weeds before application, allowing for the selection of the appropriate herbicide and application only in areas where weeds are present. The focus of this paper is to define a pipeline for detecting weeds in images through the use of a Mask R-CNN-based weed classification and segmentation module. The model was initially trained locally on our machine, but limitations and issues with training time prompted the team to switch to cloud solutions for training.

Keywords

Deep learning, CNN, Mask R-CNN, precision agriculture, weed detection.

Jabir, B., Moutaouakil, K. E. and Falih, N. (2023) "Developing an Efficient System with Mask R-CNN for Agricultural Applications", *AGRIS on-line Papers in Economics and Informatics*, Vol. 15, No. 1, pp. 61-72. ISSN 1804-1930. DOI 10.7160/aol.2023.150105.

Introduction

Agriculture is a crucial sector in Morocco, providing livelihoods for millions of people, especially in rural areas with limited industrial activity. Sustainable agriculture practices, including soil conservation, environmental resource management, and biodiversity protection, are essential for overall rural development. With over 4 million jobs, agriculture is one of the most important drivers of Morocco's economy (Jabir et al., 2021).

Precision Agriculture is an innovative approach that leverages technologies and data to optimize crop management strategies, such as fertilizer inputs, irrigation, and pesticide use. It involves collecting field data, analyzing the information, and making informed decisions to improve crop yields and reduce costs (Tiwari and Jaga, 2012). Site-specific weed management is a key component of Precision Agriculture that aims to reduce herbicide usage, improve weed control, and minimize environmental pollution (Fernández-Quintanilla et al., 2018). Weeds are a major challenge for farmers, as they compete with crops for resources and can lead to reduced yields. Chemical weed control is the most common approach, but the use of herbicides is increasingly being scrutinized due to environmental concerns. Therefore, it is important for farmers to inspect their fields and apply only the necessary amount of herbicides. This manual process is time-consuming, and an automated solution is needed to streamline the workflow.

In this study, we propose a pipeline for detecting and masking weeds in images. The images undergo preprocessing and are fed into a Mask R-CNN model. The training stage involves exploring two approaches: local training using local resources on our machine, and cloud-based training using remote resources. Finally, both approaches are compared and evaluated based on time and memory allocation. The framework acts as a feature extractor, capable of discovering complex patterns in the data that are then inputted into a multi-class classifier for classification and instance segmentation.

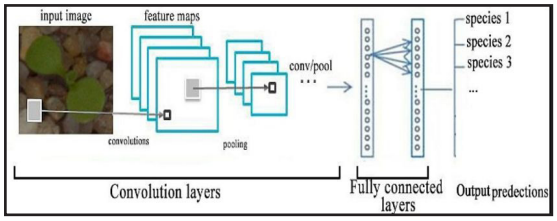
The structure of this paper is organized as follows: we start by discussing the basics of Convolutional Neural Network (CNN) and the algorithms used in our study. Then, we present the datasets and tools used in our implementation of the Mask R-CNN algorithm. After that, we describe the implementation process and the two approaches to training our model - local training using

local resources and cloud-based training using remote resources. Next, we present the results and discussion, evaluating the performance of our model and comparing the results obtained from both training approaches. Finally, we conclude our study by summarizing our findings and offering insights into the potential of this approach for future work in precision agriculture.

Materials and methods

The Convolutional Neural Network (CNN) is a widely used Deep Learning algorithm, particularly in image classification. CNNs work by attributing significance (weights and learnable biases) to objects in an input image, enabling their differentiation from one another (Sewak et al., 2020). The structure of the CNN is modeled after the human brain's neuron connectivity and is inspired by the regulation of the visual cortex. Just like in the human brain, individual neurons in a CNN only respond to stimuli within a restricted area of the visual field, known as the receptive field. The combination of these fields covers the entire visible area. In computer vision, a grayscale image is represented as a two-dimensional array of pixel values, with brightness ranging from 0 to 255. 0 indicates black, 255 indicates white, and all other values are shades of gray. Color images are represented by a third dimension of depth, with 3 values representing the fundamental colors Red, Green, and Blue (Jabir et al., 2021). Simply put, what a CNN does is extract features from an image and convert it into a lower dimension representation, while preserving its characteristics, by passing the image through a set of layers that define the algorithm's architecture.

As Figure 1 shows, an input image is passed into the input layer. The image will be handled by the Convolutional layer to extract the features of the image, but you are probably wondering how this convolution operation actually relates to feature extraction. First of all, a part of the image is connected to the Convolutional layer to perform convolution operation and calculates the dot product between the receptive field (it is a local region of the input image that has the same size as that of filter) and the filter which is formed from a set of weights. The output of the process is a one-number integer of the volume of the output. Then, we slide the filter to the following receptive field of the same entry image by one pitch and do the same operation again. We will repeat the same process repeatedly until we have traversed the entire image. The result called the convolution maps will be the input to the subsequent layer.



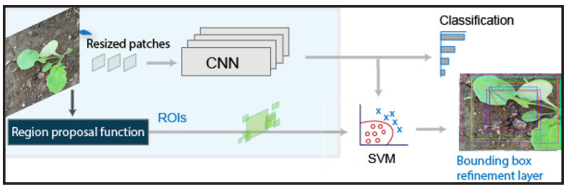
Source: (Jabir et al., 2021).

Figure 1: Standard architecture of a CNN.

After the feature extraction, an activation function called ReLU (y = max(x, 0)) is applied on the convolution map to handle highly non-linear data. The ReLU function inputs any real number and sets all values less than zero to zero, while keeping values greater than zero unchanged (Zhu et al., 2020). The pooling layer is then used to reduce the size of the input image after convolution. Multiple convolutional layers and one pooling layer may be used before the fully connected layer performs classification. The last layer of the CNN is either the Softmax or the Logistic layer, which is located at the end of the fully connected layer. Logistic is used for binary classification and Softmax is used for multi-classification (Khachnaoui et al., 2020). Transfer learning is a machine learning technique where a model trained for one task is reused as the basis for a model in a different task. This is a popular method in deep learning, where pre-trained models can serve as starting points for computer vision and natural language processing tasks due to the vast computing resources and time required to build neural network models (Hoo-Chang et al., 2016).

R-CNN

The R-CNN merges the region proposal with the CNN, based on the principle that a single object of interest would dominate in a particular region. This is achieved by using a selective search algorithm to generate category-independent region proposals, which retrieves approximately 2000 proposals. Each proposal is then warped and passed to a large convolutional neural network that acts as a feature extractor, producing a fixed-length feature vector from each region. The R-CNN extracts a 4096-dimensional feature vector from each region proposal, as shown in Figure 2. Subsequently, a Support Vector Machine (SVM) is applied to the features extracted from the CNN to rank the objects in each region. Finally, regression is used to predict the four bounding box values necessary for object detection (Hoeser and Kuenzer, 2020).



Source: Author's illustration

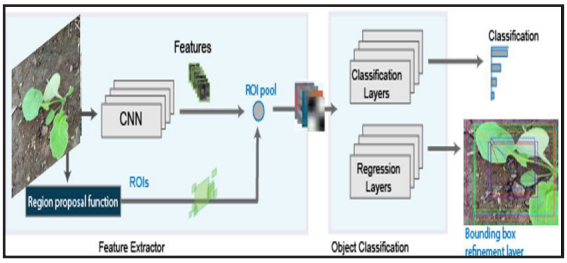
Figure 2: R-CNN: Regions with CNN features.

Fast R-CNN

Fast R-CNN is an improved version of R-CNN with a few changes. It replaces the resizing operation with a RoI pooling layer to obtain a fixed-size feature map. Secondly, it replaces the SVM with fully connected layers responsible for classification and bounding-box regression, while the region proposal is still based on selective search. Fast R-CNN takes as input an image and a set of object proposals and uses convolutional and max pooling to produce a convolutional feature map. For each object proposal, Fast R-CNN extracts a fixed-length feature vector from the feature map using a region of interest (RoI). Each feature vector is fed into a fully connected network with two sibling output layers: the first one produces softmax probability estimates over K object classes plus a catch-all "background" class, while the second one gives the four real values for the bounding-box positions for the K classes (Han et al., 2022).

Faster R-CNN

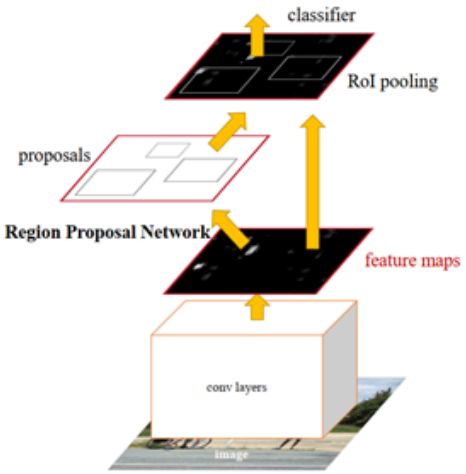
Faster R-CNN is an object detection system that consists of two modules: the first is a fully convolutional network responsible for proposing regions, and the second is a Fast R-CNN detector that uses the image and region proposals to give object classification and bounding-box positions (as shown in Figure 3). Faster R-CNN is a single, unified network for object detection.



Source: Author's illustration

Figure 3: Fast R-CNN architecture.

With the addition of the Region Proposal Network (RPN) module, Faster R-CNN is able to determine precisely where to look, which is a key advantage of Faster R-CNN (as shown in Figure 4).

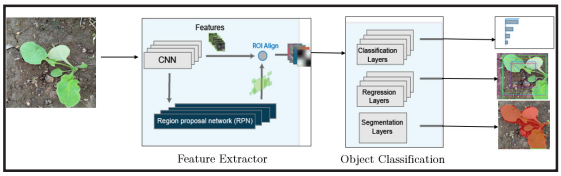


Source: (Rajeshwari g et al., 2019).

Figure 4: Faster R-CNN (single, unified network for object detection).

Mask R-CNN

Mask R-CNN is a state-of-the-art object detection and instance segmentation model developed by Facebook using Python. It extends Faster R-CNN by adding a mask prediction branch to its final stage (Liu et al., 2022). As shown in Figure 5, Mask R-CNN has three outputs: class label, bounding-box offset, and an object mask. The ROI pooling layer in Faster R-CNN has been replaced with the ROI Align layer, which performs better in mask prediction. Mask R-CNN is divided into two parts: the backbone, responsible for feature extraction, and the head of the network, which performs classification, regression, and mask prediction. Mask R-CNN is built on the Feature Pyramid Network (FPN) and uses RestNet101 as its backbone. Unlike traditional models that use a single feature map, FPN architecture utilizes features from multiple convolution layers to provide a better prediction (Lin et al., 2021). The network head classifies the proposed RPN bounding box and generates the segmentation mask. In the training stage, Mask R-CNN employs transfer learning by using pre-trained weights from the MS COCO dataset, which has 80 classes and 115,000 training images (Lin et al., 2018).



Source: Author's illustration

Figure 5: The architecture of the Mask R-CNN.

Dataset

The main dataset used by Mask R-CNN is an MS COCO dataset, which has 80 classes and one hundred fifteen thousand training images. Evaluation metrics for bounding boxes and segmentation mask is based on Intersection over Union. The pretend weights learned on MS COCO dataset are used such as pre-trained weights to train our model with our own datasets. The dataset we use is composed of 150 images, after the augmentation it became a 300 images. We divided our dataset used in this study to three sets, the first one consisting of 200 image for treating our model, the second one include 20 images for he validation and the last one be composed of 80 images for testing. All the images that are used in the study are pictures of weeds found in the fields and the corps in Morocco (Timpanaro et al., 2021).

Data augmentation

After the step of collecting images for our study. We need to ensure we have a wide variation in angles, brightness, scale, etc. and to make sure, there are a several data augmentation techniques, In our case we increase the amount of images by adding slightly modified copies of already existing images by adjusting the capture angles and brightness. This process used mostly in the case we have only small data sets to train our Deep learning models. The objective behind feeding the model with varied data is to improve the overall training procedure and performance generalization purposes (Shorten and Khoshgoftaar, 2019).

Data pre-processing

Our study involved some image pre-processing steps, before the image or particular characteristics / features / statistics of the image were fed as an input to the DL model. Our pre-processing procedure was creating pixel level mask annotations to define the boundaries of the objects in the dataset (Huang et al., 2021). Among various available tools, we choose an intuitive and well-done tool: VGG Image Annotator (VIA) (see the Figure 6). This tool does not need any installation; we lunch it via html file with a modern browser. The output of pre-processing step will be an annotated dataset and a JSON files include the annotation's metadata for the annotation for both training and validation datasets. This operation take approximately 420 min to manually annotate all image, by the average of a minute and a half for each image separately.

Source: Author's illustration

Figure 6: Image annotation.

The model architecture

The Mask R-CNN framework consists of two stages. In the first stage, the framework inputs an image and uses a Region Proposal Network (RPN) to identify potential object regions. The second stage then predicts the classes, refines the bounding boxes, and generates segmentation masks for each object. In the Mask R-CNN system, the convolutional backbone is composed of the backbone network, region proposal network, and object classification module. The network head includes the boundary box regression module and the mask segmentation module. An input image is transformed into a feature map through the backbone network, a standardized convolutional neural network (CNN) that extracts features. The feature map is then used as input for the RPN to detect potential object areas. The feature extraction step is based on the original implementation of Faster R-CNN with ResNet-101 (Lei and Sui, 2019).

```
Configurations:
BACKBONE                        resnet101
BACKBONE_STRIDES                [4, 8, 16, 32, 64]
BATCH_SIZE                      1
BBOX_STD_DEV                    [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE          None
DETECTION_MAX_INSTANCES         100
DETECTION_MIN_CONFIDENCE        0.9
DETECTION_NMS_THRESHOLD         0.3
FPN_CLASSIF_FC_LAYERS_SIZE      1024
```

Source: Author's compilation

Figure 7: Backbone of Mask R-CNN.

The ResNet-101 architecture, as shown in Figure 8, includes a total of 104 convolutional layers. It is comprised of 33 blocks of layers, with 29 of these blocks utilizing the output of the previous block as a direct input via residual connections. The remaining 4 blocks perform an additional operation of a 1x1 convolution layer with a stride of 1 followed by batch normalization before being added to the output of the previous block. The Mask R-CNN framework extends the Faster R-CNN box heads from ResNet and FPN by adding a fully convolutional mask prediction branch as part of its network head (Hafiz and Bhat, 2020).

```
Selecting layers to train
fpn_c5p5                (Conv2D)
fpn_c4p4                (Conv2D)
fpn_c3p3                (Conv2D)
fpn_c2p2                (Conv2D)
fpn_p5                  (Conv2D)
fpn_p2                  (Conv2D)
fpn_p3                  (Conv2D)
fpn_p4                  (Conv2D)
In model:  rpn_model
    rpn_conv_shared         (Conv2D)
    rpn_class_raw           (Conv2D)
    rpn_bbox_pred           (Conv2D)
mrcnn_mask_conv1        (TimeDistributed)
mrcnn_mask_bn1          (TimeDistributed)
mrcnn_mask_conv2        (TimeDistributed)
mrcnn_mask_bn2          (TimeDistributed)
mrcnn_class_conv1       (TimeDistributed)
mrcnn_class_bn1         (TimeDistributed)
mrcnn_mask_conv3        (TimeDistributed)
mrcnn_mask_bn3          (TimeDistributed)
mrcnn_class_conv2       (TimeDistributed)
mrcnn_class_bn2         (TimeDistributed)
mrcnn_mask_conv4        (TimeDistributed)
mrcnn_mask_bn4          (TimeDistributed)
mrcnn_bbox_fc           (TimeDistributed)
mrcnn_mask_deconv       (TimeDistributed)
mrcnn_class_logits      (TimeDistributed)
mrcnn_mask              (TimeDistributed)
```

Source: Author's compilation

Figure 8: The head's layers.

Implementation

This section defines the environment requirements for implementing Mask R-CNN in this study. We use Python 3.6 as the programming language and the TensorFlow and Keras libraries for learning and classification (Yang et al., 2020). To enhance the model's performance, we employ simple and effective techniques such as data augmentation and use Tensorboard for log visualization.

TensorFlow

TensorFlow is an open-source machine learning library developed by Google for developing and running machine learning and deep learning applications. Its name is derived from the fact that operations in neural networks are primarily performed on multi-dimensional data tables, called tensors. A two-dimensional tensor is equivalent to a matrix. In this study, we use TensorFlow 1.15 for local training on our machine and Tensorflow-GPU 1.5 for cloud training.

Keras

Keras is an open-source library written in Python (under the MIT license) based on the work of Google developer François Chollet as part of the ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System) project. The library's goal is to allow the rapid constitution of neural networks, serving as an application programming interface (API) for accessing and programming various machine learning frameworks (Wäldchen and Mäder, 2018). In this study, we use Keras 2.2.5.

Tensorboard

Tensorboard is a visualization tool for understanding, debugging, and optimizing TensorFlow programs. It visualizes the TensorFlow graph, plots quantitative metrics about the execution of the graph, and displays additional data.

Training

The training can be carried out at two levels. At the first level, only the heads can be trained by freezing all the backbone layers and training only the newly initialized layers, not using pre-trained weights from MS COCO. The second level involves training all layers of the entire model.

For this study, it is not necessary to train the model fully since the starting point is COCO-trained weights. Moreover, with a small dataset of 300 images, consisting of 200 for training and 20 for validation, it is not necessary to train

all layers, as it would consume a lot of time. Just training the heads should suffice.

CPU vs GPU

The central processing unit (CPU) is the main component that performs arithmetic, logic, and control for every computer. Its main function is to execute instructions stored in the computer's memory in a sequential manner. The CPU plays a critical role in neural network computation because it handles general arithmetic calculations during the learning phase. CPUs are usually built with several powerful processing cores that are clocked between 2 and 3 GHz, making them ideal for performing sequential tasks (Padilha and Lucena, 2020). Additionally, the CPU independently of the GPU's computation role (Pang et al., 2020), such as loading training data, handles all input/output operations during the learning phase. Due to these tasks, training a large model using the CPU can be risky and time-consuming, which is why using the GPU instead of the CPU in the training stage is seen as beneficial.

The graphics processing unit (GPU), like the CPU, is a component of a computer used to process instructions, but the GPU can run multiple instructions simultaneously through parallelization. A GPU typically consists of multiple weak processing cores with a much lower clock speed compared to the CPU. This multiple processing core system was developed to parallelize computations through the use of threads, thus speeding up computations that would normally take a longer time on the CPU.

Since the GPU has the ability to run many processes simultaneously, it is useful for training neural networks that involve computationally intensive matrix multiplications. Training a neural network requires a large number of computations, and the GPU optimizes these computations using multiple memory channels and streaming processors (Thao et al., 2021).

Originally, GPUs were designed for rendering graphics. As a result, executing custom code on the GPU requires APIs that provide a higher level of abstraction, from low-level to high-level programming languages. CUDA was developed to utilize the GPU architecture's parallelism capabilities, such as multithreading, MIMD, SIMD, and instruction level, through low-level instructions (Carneiro Pessoa et al., 2018). TensorFlow, in conjunction with CUDA, can use the entire GPU architecture to further optimize computation time.

Train the algorithm locally on our machine

In our study, we trained the model using a machine with the following configuration: an Intel Core i5-2520M CPU @2.50 GHz and 8GB of RAM. The training took approximately 7 hours for 10 epochs, with an average of 2100 seconds per epoch (Figure 9). Each epoch consisted of 60 training batches and a prediction threshold of 0.9. The training loss was 0.6350 and the validation loss was 0.9447. Despite having a small training dataset, the model still achieved a decent level of accuracy.

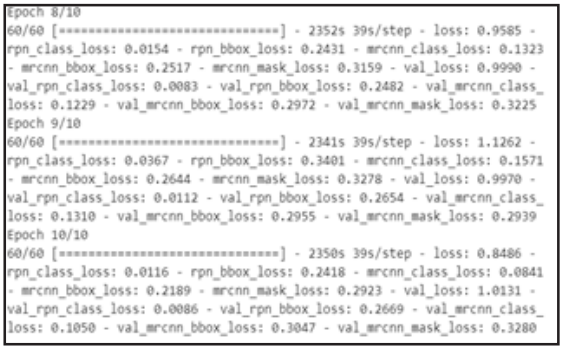
```
Epoch 1/10
60/60 [==============================] - 2314s
loss: 0.4575 - val_loss: 1.2266 - val_rpn_class
WARNING:tensorflow:From C:\Users\El-Mehdi\anacc

Epoch 2/10
60/60 [==============================] - 2271s
loss: 0.3390 - val_loss: 1.1620 - val_rpn_class
Epoch 3/10
60/60 [==============================] - 2202s
loss: 0.3405 - val_loss: 1.0426 - val_rpn_class
Epoch 4/10
60/60 [==============================] - 2165s
loss: 0.3377 - val_loss: 0.9638 - val_rpn_class
Epoch 5/10
60/60 [==============================] - 2133s
loss: 0.2975 - val_loss: 0.9637 - val_rpn_class
Epoch 6/10
60/60 [==============================] - 2138s
loss: 0.3003 - val_loss: 0.9113 - val_rpn_class
Epoch 7/10
60/60 [==============================] - 2128s
loss: 0.2577 - val_loss: 0.9187 - val_rpn_class
Epoch 8/10
60/60 [==============================] - 2107s
loss: 0.3122 - val_loss: 0.9027 - val_rpn_class
Epoch 9/10
60/60 [==============================] - 2131s
loss: 0.2805 - val_loss: 0.9736 - val_rpn_class
Epoch 10/10
60/60 [==============================] - 2107s
loss: 0.2495 - val_loss: 0.9447 - val_rpn_class
```

Source: Author's compilation

Figure 9: Train the algorithm locally on a machine.

In our dataset, we found various species of weeds with different sizes and shapes. The images in the dataset were of different sizes, so we resized them to 512 X 512 pixels. Then, we created two paths, one for the test dataset and another for the validation dataset. After that, we set up our model by extending the Dataset class and the Config class, which exist in the mrcnn folder, and configured it. Finally, we started the training. In the above model shown in Figure 9,

we trained a model for binary classification. However, to differentiate between weeds and crops with more precision, we decided to make the model larger and give it more classes. Therefore, we expanded our dataset by adding new weed species and crop types, such as sugar beet and wheat, which comprised almost 620 images. After preparing the new dataset, we re-trained the model with it.



Source: Author's compilation

Figure 10: The model is being retrained locally on our machine.

This time, the model took longer to train, approximately two hours more than the previous training. It took approximately 9 hours for 10 epochs, with an average of 2400 seconds per epoch (as shown in Figure 10). Each epoch consisted of 60 training batches and had a prediction threshold of 0.9. The training loss was 0.6350 and the validation loss was 0.9447. Training the model on our machine requires more resources and performance optimization to improve accuracy and reduce the training time.

Training the model on the cloud

Using cloud computing for deep learning simplifies the integration and management of large datasets for training algorithms. Deep learning models can then be efficiently and cost-effectively scaled using the processing power of GPUs. The cloud optimizes network distribution, enabling faster design, development, and training of deep learning applications. The use of the cloud offers several advantages, such as:

Speed: Deep learning algorithms are designed for fast learning. By utilizing GPU and CPU clusters for complex matrix operations, users can speed up the training of deep learning models. These models can handle large amounts of data and provide increasingly relevant results.

Scalability: Deep learning neural networks are ideal for running on multiple processors and distributing workloads across different types and amounts of processors. The cloud provides a wide range of on-demand resources, enabling the deployment of virtually unlimited resources to build deep learning models of any size.

Flexibility: Deep learning frameworks such as Apache MXNet, TensorFlow, Microsoft's Cognitive Toolkit, Caffe, Caffe2, Theano, Torch, and Keras can be run in the cloud, allowing you to choose the set of deep learning algorithm libraries that best fit your use case, whether it involves web, mobile, or connected devices.

There are several platforms and servers available for training remote models, including:

Amazon Web Services (AWS): AWS offers over a hundred services that fall into categories such as compute, storage, database, developer tools, security and identity, analytics, artificial intelligence, and more. New customers are eligible for 12 months of free use with certain restrictions and limitations. Any usage beyond these limitations must be purchased. For example, they offer a Remote Desktop Protocol with 30 GB of storage, 2 GB of RAM, and 1 CPU for free. Additional resources, such as GPUs, must be purchased.

Google Colab: In recent years, Google Colab has become a popular choice for an end-to-end machine learning platform. It provides free GPU, CPU, storage, and RAM, but also has limitations. Some of the downsides of Google Colab include service interruptions, slow storage, unconfigured environments, and limited functionality (12 hours of interactive use).

Kaggle: Kaggle, another Google product, is a web-based platform that hosts data science contests. It offers an end-to-end machine learning platform with features similar to Colab, including free Jupyter notebooks and GPUs. Kaggle also provides many pre-installed Python packages, making it easier for some users to start.

Paperspace Gradient: Gradient is the solution we utilized in this study. An end-to-end platform offers a free-hosted Jupyter notebook cloud service with several options for pre-configured environments and free access to GPUs and CPUs. Gradient makes it easy to build, train, and deploy deep learning models, with a web-based user interface, a CLI, and an SDK. It appeals to both beginners and experts alike, with a user-friendly interface and low entry barrier. Some of the benefits of Gradient over other solutions are:

Faster, persistent storage, eliminating the need

to re-install libraries and re-download files each time you start your notebook.

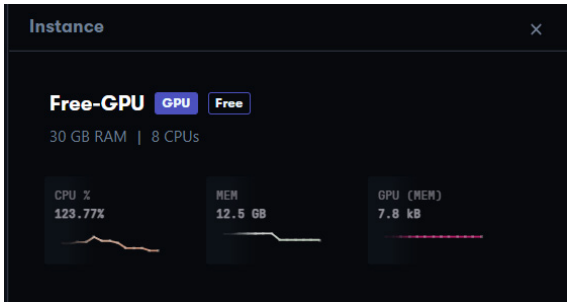
Guaranteed sessions, reducing the risk of having your instance shut down mid-work. You can log out and come back later to find your session unchanged.

Pre-configured containers and templates, including popular environments with pre-installed dependencies (such as PyTorch, TensorFlow, or the Data Science Stack) or the option to use a custom container. There is also an ML showcase with sample projects that you can create and run free on your account.

A public dataset repository with a wide range of popular datasets available for free use and mounted on every notebook.

The ability to easily scale up and add more storage and high-end dedicated GPUs for the same environment as needed.

In this study, we will use the free environment offered by Gradient. It provides two options: the first offers access to 2 GB of RAM, 2 CPUs, and no GPU (Figure 11), with a runtime of 12 hours without interruption. The second option provides access to 30 GB of RAM, 8 CPUs, an Nvidia Quadro M4000 GPU, and a runtime of 6 hours without interruption. We have chosen to work with the second option.



Source: AWS Cloud (2022)

Figure 11: The training environment (Cloud).

We will start by taking the same dataset and model used in the second local training and re-run the training on the cloud. This time, it took around 28 minutes for 10 epochs, with an average of 170 seconds per epoch, 60 training batches, and a threshold of 0.9 for prediction. The training loss was 0.8406, and the validation loss was 0.964.

To decrease the loss, we increased the number of training batches to 100 and retrained the model. This time, the model took 45 minutes for 10 epochs, with an average of 280 seconds per epoch and 100 training batches, with a prediction threshold of 0.9. The training loss was 0.6896 and the validation loss was 0.9113. However, as the loss was still high, we retrained the model again, this time with 40 epochs, 100 training batches per epoch, and a prediction threshold of 0.9. The training loss reached 0.2871, while the validation loss was 0.9663 (Figure 12). This time, the model took almost 3 hour.

```
Epoch 36/40
100/100 [==============================] - 255s 3s/step - loss: 0.273
9 - rpn_class_loss: 0.0044 - rpn_bbox_loss: 0.0452 - mrcnn_class_los
s: 0.0314 - mrcnn_bbox_loss: 0.0373 - mrcnn_mask_loss: 0.1556 - val_l
oss: 0.9307 - val_rpn_class_loss: 0.0078 - val_rpn_bbox_loss: 0.3262
- val_mrcnn_class_loss: 0.0848 - val_mrcnn_bbox_loss: 0.2101 - val_mr
cnn_mask_loss: 0.3018
Epoch 37/40
100/100 [==============================] - 259s 3s/step - loss: 0.295
4 - rpn_class_loss: 0.0045 - rpn_bbox_loss: 0.0508 - mrcnn_class_los
s: 0.0355 - mrcnn_bbox_loss: 0.0437 - mrcnn_mask_loss: 0.1609 - val_l
oss: 0.9927 - val_rpn_class_loss: 0.0072 - val_rpn_bbox_loss: 0.4082
- val_mrcnn_class_loss: 0.0675 - val_mrcnn_bbox_loss: 0.2174 - val_mr
cnn_mask_loss: 0.2922
```

Source: Author's compilation

Figure 12: Training the model on a Gradient server (in the cloud).

Results and discussion

In this section, we will examine the results obtained during the training of our model, evaluate its performance, and demonstrate its prediction on sample data from the test dataset in the inference stage. Evaluating a deep learning model is a crucial step in any project process as it allows us to assess the accuracy and performance of the model. There are various parameters and metrics that can be used for evaluation. In our study, we will use logarithmic loss as our evaluation parameter.

Logarithmic loss, also known as log loss, is a suitable metric for multi-class classification. It penalizes false classifications by requiring the model to attribute a probability to each class for all samples. If we have N samples belonging to M classes, the log loss is calculated as follows:

$$Logarithmic\ Loss = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} \times log\ (p_{ij})$$

Where,

y_{ij}, indicates whether sample i belongs to class j or not

p_{ij}, indicates the probability of sample i belonging to class j

The log loss has no upper limit and exists in the range [0, ∞). A log loss closer to 0 would indicate higher accuracy, while if the log loss is away from 0, it will indicate less accuracy.

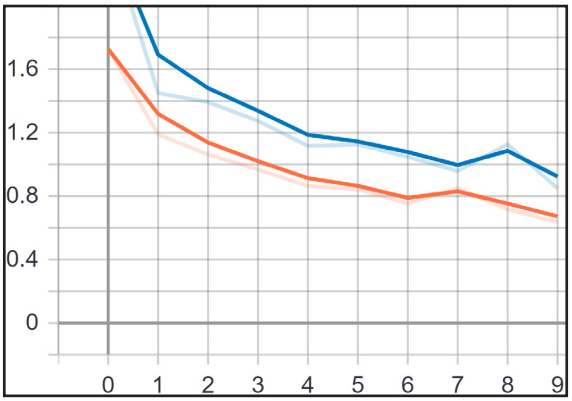
In general, decreasing the log loss will give a better accuracy for the classifier.

The goal of training a model is to find a set of weights and biases that have low Loss, on average, across all examples.

In Mask R-CNN we have 5 small principle losses, each one has a specific signification, and the figures below Losses obtained as a result of training, it can be accessed through the use of Tensorboard:

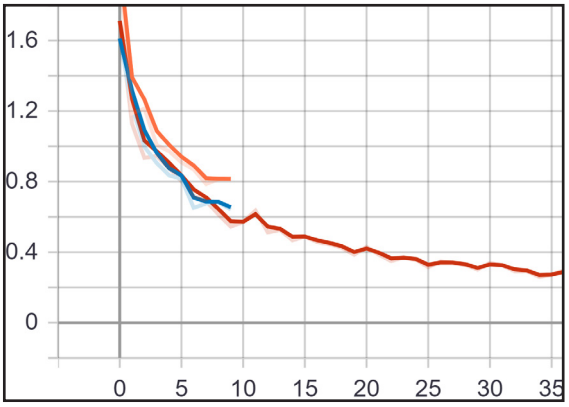
- **rpn_class_loss:** How well the Region Proposal Network separates background with objetcs.
- **mrcnn_bbox_loss:** How well the Mask RCNN localize objects.
- **mrcnn_class_loss:** How well the Mask RCNN recognize each class of object.
- **mrcnn_mask_loss:** How well the Mask RCNN segment objects.

We use thus losses to calculate a big Loss: A combination (surely an addition) of all the smaller losses.



Source: Author's compilation

Figure 13: The model loss during cloud training.



Source: Author's compilation

Figure 14: The model loss during local training.

After training the model both locally and on the cloud, the results shown in Figures 13 and 14 indicate that cloud training is faster and better. The loss reaches 0.26 in the last epoch. However, it is evident that 10 epochs are not enough to train the model effectively, as seen in the reduction of the loss rate after epoch 13. This is because the model starts to memorize the input data rather than learn the underlying patterns, leading to overfitting. Hence, we will use the weights obtained after epoch 13 for inference.

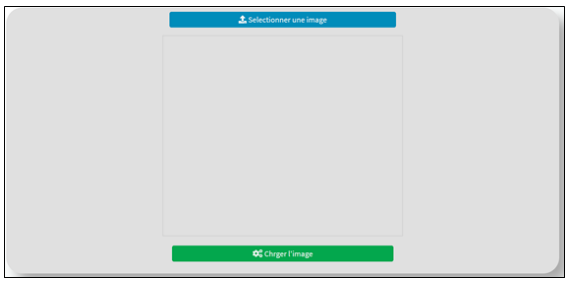
A comparison between local and cloud training is summarized in Table 1. The first local training, with a dataset of 300 images, 10 epochs, and 60 batches, took 7 hours and resulted in a loss of 0.848, indicating an inaccurate model. In the second local training, the number of images was increased to 620, and the training time increased to 9 hours. However, the model became more accurate. On the cloud, with an environment offering 30 GB of RAM, 8 CPU, and GPU, the same model was trained in 28 minutes with the parameters from the second local training. The accuracy was not suitable, so the number of epochs and batches was increased to 40 and 100 respectively, which took 3 hours to complete with a convincing precision.

	Epoch	Batches	Dataset	Duration	Duration / epoch	Duration / step	Loss
Local	10	60	300 images	7 hours	2100 seconds	36 second	0.8486
Local	10	60	620 images	9 hours	2300 seconds	39 second	0.635
Cloud	10	60	620 images	28 minutes	170 seconds	2 seconds	0.8406
Cloud	10	100	620 images	45 minutes	280 seconds	3 seconds	0.6896
Cloud	40	100	620 images	3 hours	280 seconds	3 seconds	0.26

Source: Compiled by the authors

Table 1: Comparison of local model training and cloud model training.

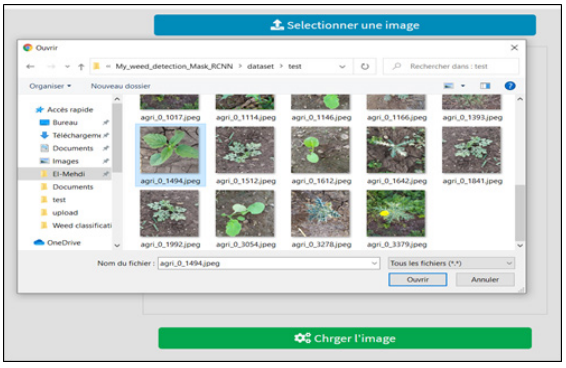
In this section, we will use the trained model on the test dataset to predict the presence of weeds or crops in images. We have integrated the trained model into a web application developed using Flask. The interface of the application includes two buttons and an image frame as displayed in Figure 15. The upper button enables users to select an image, while the lower button allows them to load the image onto the server.



Source: Author's illustration

Figure 15: Home page.

After clicking the "Select Image" button (as shown in Figure 16), a window will appear for selecting the specific image to be used in the prediction process.



Source: Author's illustration

Figure 16: Upload image

After selecting the image, it can be viewed in the frame. To start the prediction process, the image must be uploaded to the server by clicking the "Upload Image" button. Once the image has been uploaded, the prediction can be initiated by clicking the "launch Detection" button, as depicted in Figure 17.

Source: Author's illustration

Figure 17: Launch detection

At the end of the prediction process, the application provides the result of the initial image with a bounding box around the specific object (if it exists) along with the name of the class and the prediction probability. A mask is also applied to detect the object boundaries, as depicted in Figure 18. Users can start another prediction by clicking on the "New Prediction" button. The results shown in the images indicate that the application, using the trained model, can accurately detect and surround the grass in a short amount of time, and if run on the cloud, the results will be even faster.

Source: Author's illustration

Figure 18: Result

Conclusion

The objective of this study was to create, train, and optimize a Mask R-CNN model for weed detection in images by comparing two training approaches: local machine and cloud. The goal was to achieve the highest accuracy and lowest loss in predicting, segmenting, and identifying the presence of common weeds in pictures. The successful implementation of this model can aid in optimizing herbicide use and controlling the spread of weeds. However, this model is limited to the images it was trained on and may

not be applicable to all weed species or growing conditions. Future work will aim to improve the model by incorporating a hybrid approach, incorporating additional data and increasing the diversity of the training set. This has the potential to further improve the accuracy of the model and make it more applicable to a wider range of use cases. The results of this study and future work in this area could have important implications for the agricultural industry, reducing environmental damage and improving crop yields.

Acknowledgements

This research is part of a larger scientific project supported by a group of doctors from the LIMATI Laboratory at Sultan Moulay Slimane University in Morocco. The general topic of the project is "Deep learning in agriculture."

Corresponding author:
Brahim Jabir
LIMATI, Sultan Moulay Slimane University
Av Med V, BP 591, Beni-Mellal 23000, Morocco
Phone: +212 639 08 05 91, E-mail: ibra.jabir@gmail.com

References

[1] Carneiro Pessoa, T., Gmys, J., de Carvalho Júnior, F. H., Melab, N. and Tuyttens, D. (2018) "GPU-accelerated backtracking using CUDA Dynamic Parallelism", C*oncurrency and Computation: Practice and Experience*, Vol. 30, No. 9, pp. 16-30. ISSN 1532-0626. DOI 10.1002/cpe.4374.

[2] Fernández-Quintanilla, C., Peña, J. M., Andújar, D., Dorado, J., Ribeiro, A. and López-Granados, F. (2018) "Is the current state of the art of weed monitoring suitable for site-specific weed management in arable crops?", *Weed Research*, Vol. 58, No. 4, pp. 259-272. ISSN 0043-1737. DOI 10.1111/wre.12307.

[3] Hafiz, A. M. and Bhat, G. M. (2020) "A survey on instance segmentation: state of the art", *International Journal of Multimedia Information Retrieval*, Vol. 9, No. 3, pp. 171-189. ISSN 2192-6611. DOI 10.1007/s13735-020-00195-x.

[4] Han, G., Huang, S., Ma, J., He, Y. and Chang, S. F. (2022) "Meta faster r-cnn: Towards accurate few-shot object detection with attentive feature alignment", In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36, No. 1, pp. 780-789. ISSN 2374-3468. DOI 10.1609/aaai.v36i1.19959.

[5] Hoeser, T. and Kuenzer, C. (2020) "Object detection and image segmentation with deep learning on earth observation data: A review-part i: Evolution and recent trends", *Remote Sensing*, Vol. 12, No. 10, p. 1667. ISSN 2072-4292. DOI 10.3390/rs12101667.

[6] Huang, Z., Chen, H., Liu, B. and Wang, Z. (2021) "Semantic-guided attention refinement network for salient object detection in optical remote sensing images", *Remote Sensing*, Vol. 13, No. 11, p. 2163. ISSN 2072-4292. DOI 10.3390/rs13112163.

[7] Jabir, B., Falih, N., Sarih, A. and Tannouche, A. (2021) "A Strategic Analytics Using Convolutional Neural Networks for Weed Identification in Sugar Beet Fields", *AGRIS on-line Papers in Economics and Informatics*, Vol. 13, No. 1, pp. 49-57. ISSN 1804-1930. DOI 10.7160/aol.2021.130104.

[8] Jabir, B., Falih, N. and Rahmani, K. (2021) "Accuracy and Efficiency Comparison of Object Detection Open-Source Models", *International Journal of Online & Biomedical Engineering*, Vol. 17, No. 5. pp. 165-184. ISSN 2626-8493. DOI 10.3991/ijoe.v17i05.21833.

[9] Khachnaoui, H., Mabrouk, R. and Khlifa, N. (2020) "Machine learning and deep learning for clinical data and PET/SPECT imaging in Parkinson's disease: a review", *IET Image Processing*, Vol. 14, No. 16, pp. 4013-4026. ISSN 1751-9659. DOI 10.1049/iet-ipr.2020.1048.

[10] Lei, X. and Sui, Z. (2019) "Intelligent fault detection of high voltage line based on the Faster R-CNN". *Measurement*, Vol. 138, No. 8, pp. 379-385. ISSN 02632241. DOI 10.1016/j.measurement.2019.01.072.

[11] Lin, C., Shi, Y., Zhang, J., Xie, C., Chen, W. and Chen, Y. (2021) "An anchor-free detector and R-CNN integrated neural network architecture for environmental perception of urban roads", *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, Vol. 235, No. 12, pp. 2964-2973. ISSN 0954-4070. DOI 10.1177/09544070211004466.

[12] Liu, W., Shan, S., Chen, H., Wang, R., Sun, J. and Zhou, Z. (2022) "X-ray weld defect detection based on AF-RCNN", *Welding in the World*, Vol. 66, No. 6, pp. 1165-1177. ISSN 0043-2288. DOI 10.1007/s40194-022-01281-w.

[13] Nguyen, T. T., Wahib, M. and Takano, R. (2021) "Efficient MPI-AllReduce for large-scale deep learning on GPU-clusters", *Concurrency and Computation: Practice and Experience*, Vol. 33, No. 12, p. e5574. ISSN 1532-0626. DOI 10.1002/cpe.5574.

[14] Padilha, T. P. P. and de Lucena, L. E. A. (2020) "A systematic review about use of tensorflow for image classification and word embedding in the brazilian context", *Academic Journal on Computing, Engineering and Applied Mathematics*, Vol. 1, No. 2, pp. 24-27. ISSN 2675-3588. DOI 10.20873/uft.2675-3588.2020.v1n2.p24-27.

[15] Pang, B., Nijkamp, E. and Wu, Y. N. (2020) "Deep Learning with TensorFlow: A Review", *Journal of Educational and Behavioral Statistics*, Vol. 45, No 2, pp. 227-248. ISSN 1076-9986. DOI 10.3102/1076998619872761.

[16] Rajeshwari, P., Abhishek, P., Srikanth, P. and Vinod, T. (2019) "Object detection: an overview", *International Journal of Trend in Scientific Research and Development (IJTSRD)*, Vol. 3, No. 1, pp. 1663-1665. ISSN 2456-6470. DOI 10.31142/ijtsrd23422.

[17] Sewak, M., Sahay, S. K. and Rathore, H. (2020) "An overview of deep learning architecture of deep neural networks and autoencoders", *Journal of Computational and Theoretical Nanoscience*, Vol. 17, No. 1, pp. 182-188. ISSN 1546-1955. DOI 10.1166/jctn.2020.8648.

[18] Shin, H.-Ch. Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D. and Summers, R. M. (2016) "Deep Convolutional Neural Networks for Computer-Aided Detection: CNN Architectures, Dataset Characteristics and Transfer Learning", *IEEE Transactions on Medical Imaging*, Vol. 35, No. 5., pp. 1285-1298. ISSN 0278-0062. DOI 10.1109/TMI.2016.2528162.

[19] Shorten, C. and Khoshgoftaar, T. M. (2019) "A survey on image data augmentation for deep learning", *Journal of Big Data*, Vol. 6, No. 1, pp. 1-48. ISSN 2196-1115. DOI 10.1186/s40537-019-0197-0.

[20] Timpanaro, G., Urso, A., Foti, V. T. and Scuderi, A. (2021) "Economic Consequences of Invasive Species in Ornamental Sector in Mediterranean Basin: An Application to Citrus Canker", *AGRIS on-line Papers in Economics and Informatics*, Vol. 13, No. 1, pp. 131-149. ISSN 1804-1930. DOI 10.7160/aol.2021.130110.

[21] Tiwari, A. and Jaga, P. K. (2012) "Precision farming in India - A review", *Outlook on Agriculture*, Vol. 41, No. 2, pp. 139-143. ISSN 0030-7270. DOI 10.5367/oa.2012.0082.

[22] Wäldchen, J. and Mäder, P. (2018) "Machine learning for image based species identification", *Methods in Ecology and Evolution*, Vol. 9, No. 11, pp. 2216-2225. ISSN 2041-210X. DOI 10.1111/2041-210X.13075.

[23] Yang, Y., Hao, X., Zhang, L. and Ren, L. (2020) "Application of Scikit and Keras Libraries for the Classification of Iron Ore Data Acquired by Laser-Induced Breakdown Spectroscopy (LIBS)", *Sensors*, Vol. 20, No. 5, p. 1393. ISSN 1424-8220. DOI 10.3390/s20051393.

[24] Zhu, D., Lu, S., Wang, M., Lin, J. and Wang, Z. (2020) "Efficient precision-adjustable architecture for softmax function in deep learning", *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 67, No 12, pp. 3382-3386. ISSN 1549-7747. DOI 10.1109/TCSII.2020.3002564.