

**Data ScienceTech Institute**

**Makerere Fall Armyworm Crop Detection using Transfer Learning Model**

**TABLE OF CONTENTS**

2024

**Prepared by**

Said ~~Fahmy~~ Hamdi

Guillaume Nony

Deena Zamzam

[**1) Introduction** 3](#_Toc153116525)

[**2) Data Loading and Exploration** 3](#_Toc153116526)

[**3) Image Data Generation** 4](#_Toc153116527)

[**4) Build Transfer Learning Model** 5](#_Toc153116528)

[**5) Results Evaluation** 5](#_Toc153116529)

[**CONCLUSION** 10](#_Toc153116534)

**1) Introduction**

Fall armyworm is a devastating pest in Africa, posing a significant threat to maize crops, which are crucial for the food security and livelihoods of millions of people on the continent. With no natural predators in Africa, the fall armyworm can cause substantial losses, with African farmers reporting average maize losses of 31% annually. Maize is not only the most widely grown crop in Africa but also a staple food for around half of the continent's population, with over 300 million Africans depending on it for food and nutritional security.

Given the importance of maize in Africa and the severe impact of pests like fall armyworm on crop yields, there is an urgent need to develop effective early intervention mechanisms to mitigate these losses and support smallholder farmers. Machine learning models offer a promising approach to detecting and predicting fall armyworm infestations, enabling farmers to take timely and targeted action to protect their crops. By leveraging data on pest behavior, environmental factors, and crop health, these models can help farmers make informed decisions and implement preventive measures to safeguard their harvests.

In this context, creating a machine learning model tailored to detect fall armyworm infestations can play a crucial role in enhancing food security, reducing poverty, and sustaining agricultural livelihoods in Africa.

**2) Data Loading and Exploration**

The project data includes a train.CSV file with 1,619 training images along with their labels, a test.CSV file with 1,080 image names only, and an Images folder containing a total of 2,699 images for both the training and testing datasets. Google Colab will be used as the integrated development environment (IDE) for this project. Once you have downloaded and uploaded the data to your Google Drive, you will need to mount the drive to access the data for the project.

The train CSV files contains two column the Image id and label of values 0 or 1 which represents the status of the leaf whether it’s affected or not.

A screenshot of a computer

Description automatically generated

As an extra step to facilitate dealing with the images in the code we are going to merge the image path with its id so the final result will look like this

A screenshot of a computer

Description automatically generated

**3) Image Data Generation**

For this project, we will utilize the **ImageDataGenerator** for image normalization and augmentation and **.flow\_from\_dataframe** for data batches loading. Following the resizing of all images to a specific dimension, we will proceed with augmenting the images.

The **flow\_from\_dataframe** function is an essential tool for us in this project. It scans the file paths specified in the dataframe and automatically locates the corresponding images in the training directory. It then applies the required preprocessing steps defined in the ImageDataGenerator, which helps in managing memory usage efficiently.

It is important to note that augmenting image data is not mandatory; however, it is done to enhance the model's performance and results by generating new and diverse examples for the training and validation datasets.

When using the **.flow\_from\_dataframe** data loader, you need to provide the csv file, specify the image\_id as the x column, the target as the y column, define the target size of the images, specify the class\_mode, subset, seed, and batch\_size. Additionally, there are other arguments that can be passed depending on the specific requirements of the project.

For the test data loader we put the y\_col as None because that is what we are predicting.

A screenshot of a computer code

Description automatically generated

Then we visualize the data through plotting

A computer code with text

Description automatically generated with medium confidence

A close-up of a corn plant

Description automatically generated

**4) Build Transfer Learning Model**

Transfer Learning is a technique in machine learning that involves repurposing a model developed for one task as the starting point for a model designed for a similar task. This approach is both computationally efficient and helps improve performance when dealing with limited amounts of data. In this project, we are employing a pre-trained model to classify plant images as either infected or uninfected.

There exist various pre-trained models available for image classification, such as VGG16, VGG19, InceptionV3, ResNet50, ResNetV2, and others. These models have undergone training on extensive datasets comprising millions of images, which enhances their capability to deliver superior results on our specific dataset.

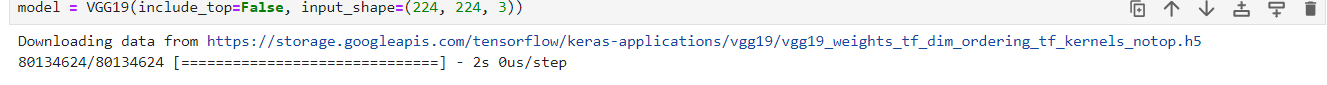
Let us proceed with importing the pre-trained model that we have chosen for this project.

A white rectangular sign with red and blue text

Description automatically generated

We have decided to utilize VGG19 as our pre-trained model for this project. VGG19 is a variation of the VGG model, comprising a total of 19 layers, including 16 convolution layers, 3 fully connected layers, 5 MaxPool layers, and 1 SoftMax layer.

Having successfully imported the pre-trained model, we will proceed without loading the output layer. This decision is based on the fact that the VGG19 model was originally trained on the ImageNet database, which consists of a million images categorized into 1000 classes. As our task involves binary image classification, we opt to freeze the existing output layers designed for 1000 classes and append our custom output layer. This freezing process is achieved by setting the include\_top argument to False. Additionally, it is important to note that VGG19 expects input images to be of size 224x224.



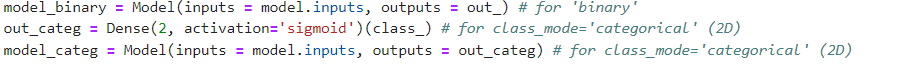
In the following code snippet, we specify the trainable argument as false to prevent the weights of the VGG19 model from impacting our current dataset. This action effectively freezes the weights of the pre-trained model, ensuring that they remain unchanged during the training process.

A white rectangular object with black text

Description automatically generated

We added three output layers to compare between them sparse, categorial and binary.





Now, let's compile the model using the Adam optimizer, binary\_crossentropy as the loss function (suitable for binary image classification tasks), and Accuracy as the metric to evaluate the model's performance.

A screen shot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer code

Description automatically generated

**5) Results Evaluation**

A graph of training and validation accuracy

Description automatically generated

A graph of a training and validation accuracy

Description automatically generated

A graph of training and validation accuracy

Description automatically generated

**CONCLUSION**