CSC4025Z: Assignment 2 Report

Richard Taylor, Gregory Maselle, Thomas van Coller

Introduction and Problem Formulation

Agriculture serves as the backbone of many African economies, employing over 60% of the population and contributing approximately 23% to the region's Gross Domestic Product (GDP). However, the farming sector faces significant challenges, with crop diseases and pests posing a substantial threat to agricultural productivity and food security. In Sub-Saharan Africa alone, crop diseases and pests can reduce yields by up to 40% annually, exacerbating food scarcity and economic instability.

Identifying and mitigating crop diseases promptly is critical to safeguarding yields and ensuring the livelihoods of millions of farmers. Traditional methods of disease detection rely heavily on the expertise of agricultural specialists, which can be both time-consuming and inaccessible, especially in remote and underserved regions. Moreover, the rising incidence of crop diseases is further aggravated by climate change, which alters the prevalence and distribution of pests and pathogens.

Thus, automatic identification of these diseases could allow farmers to identify and treat them as soon as possible. This project aims to develop and evaluate machine learning models capable of accurately classifying diseases in corn, pepper, and tomato plants using image data**.**

This project implements and tests an image classification problem for identifying diseases from images of corn, pepper, and tomato plants. The images used to train and test our models were obtained from a Zindi Hackathon. The model takes images of these plants and classifies them into one of the classes listed in Table 1. As an extension of our model we also classify regions of the image with bounding boxes, this is illustrated in Figure 1. This extension is not core to our project, and most of the focus was placed on simply matching each image to one class.

Figure 1: Bounding boxes of identified disease in images of crops

The reason for this focus is that we could simply use a convolutional neural network (CNN) to perform classification on the images. A CNN was best suited for this problem due to its ability to identify spatial hierarchy of features in the images. However, basic CNN technologies do not lend themselves to object detection and hence we could not use this model to detect the bounding boxes. Thus, the model performance for identifying the bounding boxes was very poor, and we would like to improve on this in the future. For the scope of this assignment report, we will focus on the classification element of our project.

Table 1: Classes for crop disease classification

Classification Classes	
Corn_Cercospora_Leaf_Spot	Pepper_Leaf_Curl
Corn_Common_Rust	Pepper_Leaf_Mosaic
Corn_Healthy	Pepper_Septoria
Corn_Northern_Leaf_Blight	Tomato_Bacterial_Spot
Corn_Streak	Tomato_Early_Blight
Pepper_Bacterial_Spot	Tomato_Fusarium
Pepper_Cercospora	Tomato_Healthy
Pepper_Early_Blight	Tomato_Late_Blight
Pepper_Fusarium	Tomato_Leaf_Curl
Pepper_Healthy	Tomato_Mosaic
Pepper_Late_Blight	Tomato_Septoria
Pepper_Leaf_Blight	

For ease of marking, we will link our project repo here. This includes all our prediction results, code, but not our dataset, which can be found **here**. Inside the repo is a readme.md that explains how to run the scripts. For the Random Forest Implementation, please use the code found in the submissionForestCode branch. Scripts used to generate Fig. 2 and the Appendix figures can be found there too. Some output photos have also been added in /visualisations.

Data Processing

The data and image processing workflow involves setting up the environment and organising directories for images and labels, ensuring a clean and structured dataset hierarchy. Data files are loaded into pandas DataFrames, and image paths are added while class labels are encoded numerically to facilitate model training. The dataset is split into training and validation sets with stratification to maintain class distribution, addressing potential class imbalance issues. Images are copied to their respective directories, and annotations are converted into the required format using functions that normalise bounding box coordinates.

Parallel processing is utilised to speed up the conversion of annotations, enhancing efficiency. A configuration file is created to provide the model with necessary dataset information, including paths, number of classes, and class names. Data integrity is verified through visualisation by plotting images with their bounding boxes, ensuring that annotations align correctly with the images. Finally, the environment is verified for proper configuration, including checking for GPU availability and library installations, to ensure efficient training of the object detection model.

The decision was made to run the models locally, using an NVIDIA GPU through the CUDA library rather than using Google colab (to avoid colab limits). So these scripts can all be run locally but just note you need to change the single file path in the notebook to make it work on your machine.

To run the code for yourself, there is a readme.md in the repo with instructions on how to do so. If your pc is low on processing power and memory (especially vram), an additional script has been included for image compression, which drastically reduces the size of the image dataset, without losing much quality.

Model Implementation

As mentioned above, we used a CNN for our image classifier (as they are designed to automatically and adaptively learn spatial hierarchies of features from input images.The initial layers typically detect low-level features such as edges, textures, and simple shapes. As data progresses through deeper layers, the network learns to recognize more complex and abstract patterns, such as specific disease symptoms on plant leaves).

The following model network gave us the best results. It is important to note that a pretrained model performed very similarly (in the repo there is a YOLO and ResNet model for you to compare if you are interested), thus we are very happy with our results.

A custom `CachedDataset` class is implemented to efficiently handle data loading and preprocessing.

Caching Images

- **Implementation:**
  - Images are preprocessed and cached as tensors in a specified directory.
  - The caching process includes resizing images to a standard size (128x128 pixels) and normalising them using the mean and standard deviation of the ImageNet dataset.
- **Justification:**
  - **Efficiency:** Caching reduces disk I/O during training, leading to faster data loading.
  - **Consistency:** Ensures that the same preprocessing steps are applied to all images, which is crucial for model convergence.

Convolutional Layers

- **Implementation:**
  - The model comprises four convolutional blocks within `self.conv_layers`:
    - **First Block:**
      - **Conv2d:** 32 filters, kernel size 3x3, stride 1, padding 1.
      - **ReLU Activation**
      - **MaxPooling:** Kernel size 2x2, stride 2.
    - **Second Block:**
      - **Conv2d:** 64 filters.
      - **ReLU Activation**
      - **MaxPooling**
    - **Third Block:**
      - **Conv2d:** 128 filters.
      - **ReLU Activation**
      - **MaxPooling**
    - **Fourth Block:**
      - **Conv2d:** 256 filters.
      - **ReLU Activation**
      - **MaxPooling**
- **Justification:**
  - **Progressive Feature Extraction:** Increasing the number of filters allows the network to learn more complex features at each subsequent layer.
  - **Spatial Reduction:** MaxPooling layers reduce the spatial dimensions, capturing the most salient features and reducing computational load.
  - **Activation Functions:** ReLU introduces non-linearity, enabling the network to model complex patterns.

Fully Connected Layers

- **Implementation:**
  - After flattening the output from convolutional layers, a dropout layer (`nn.Dropout(0.5)`) is applied.
  - A fully connected layer (`nn.Linear(256 * 8 * 8, 512)`) followed by a ReLU activation constitutes `self.fc_layers`.
- **Justification:**
  - **Flattening:** Transforms multidimensional feature maps into a 1D vector suitable for fully connected layers.
  - **Dropout Regularisation:** Prevents overfitting by randomly dropping neurons during training.
  - **Feature Consolidation:** The dense layer reduces dimensionality and learns high-level abstractions.

Output Layers

- **Implementation:**
  - **Bounding Box Regression Head (`self.fc_bbox`):** Outputs four values corresponding to bounding box coordinates.

    - **Classification Head (`self.fc_class`):** Outputs class probabilities for each class in the dataset.
- **Justification:**
    - **Task Separation:** Separate heads allow the model to optimise for both tasks simultaneously without interference.
    - **Specialisation:** Each head can focus on learning features pertinent to its specific task.

Batch Size

- **Implementation**
    - This hyperparameter was eventually refined to 128
- **Justification**
    - **Bigger is better:** A larger batch size leads to improved computational efficiency through the better use of **Parallelism.**
    - **But….** Larger batches consume more GPU memory, which can be a limiting factor, especially with high-resolution images or complex models. This then risks exceeding GPU memory capacity, leading to training interruptions. 128 was the maximum we could reasonably justify.
    - **And…** large batch sizes may lead the optimizer to converge to **sharp minima** in the loss landscape, which might not generalise as well to unseen data, leading to overfitting.

Baseline Method

To test how well our CNN performs in classifying an image, we implemented a random forest classifier to act as a baseline for comparing the performance of the methods. A random forest makes use of several tree classifiers that are combined to then form the final classification for a given problem. It is a simple machine learning model and hence was ideal to use as our baseline.

Random forests are not able to directly train on images. Hence, we converted the images to a set of features beforehand. We used a pretrained model, and excluded the final classification layer so that we only obtained the extracted features. These were then used for training of the random forest instead of the actual images themselves.

Following this, we used a random forest with 100 trees and no prescribed depth, implemented in Python with the scikit-learn library. This forest is great at making a single prediction per image, but struggles greatly with identifying multiple classes per image. A solution to this could be breaking up the images for individual classifications, however the high level of randomness in classification of images will not allow for this process to generalise well, and hence we decided not to go with this solution and keep the model to one classification per image.

We believe the random forest acts as a decent baseline for the problem, giving its simplicity to implement and great degree of involvement in development from our side - i.e. it is not just a pretrained model we have given our data to. We anticipate this to perform poorly in comparison to the CNN as the random forest cannot capture the spatial relationships of the data like a CNN can. The next sections will discuss the results of the two methods.

Experimental Setup and Evaluation

Multiple metrics were used to evaluate the model's performance. They are outlined below. Note that if a metric is "weighted", then the metric is calculated for each class and aggregated together based on a predefined weight for each class:

- **Accuracy:** The ratio of correctly predicted instances to the total instances. This measures the overall effectiveness of the model but can be misleading if the dataset is imbalanced across classes.
- **Precision (weighted):** For each class, it's the ratio of correctly predicted instances of that class (true positives for the class) to all instances predicted as that class (true positives + false positives for that class). This indicates how many of the predicted instances for a particular class were correct.
- **Recall (weighted):** For each class, it's the ratio of correctly predicted instances of that class (true positives for the class) to all actual instances of that class (true positives + false negatives for that class). This measures how well the model identifies all instances of a particular class.
- **F1 Score (weighted):** The harmonic mean of precision and recall for each class, providing a balanced metric between precision and recall. The F1 score will only be high if both precision and recall are high for the classes, making it useful for assessing performance across imbalanced classes.
- **Confusion Matrix:** A table visualising the performance of the model by showing the actual versus predicted classifications for each class. This gives a detailed view of where misclassifications are happening per class, providing insight into where the model is performing well and where it is falling short.

As an aside, the metric used for bounding box accuracy is Intersection over Union (IoU). This metric is a measure of how many of the bounding boxes are "accurate enough". If a predicted bounding box overlaps the ground truth bounding box by some predetermined amount (threshold), then the success of that predicted bounding box is "positive". The main metric is the percentage of "positive" bounding box predictions using IoU.

These metrics of course need to be generated from training and validation data. These splits were helpfully predetermined by the organisers of the hackathon. The dataset is first split into a training (80%) and test split (20%). Note that the training set has such a great number of values since one image may have multiple bounding boxes (multiple blemishes). The training split is then split again into the actual training split and the validation split. A quarter of the initial training split is used for validation. Additionally, stratification is used to ensure

that ratios of classes in both the training split and validation split are the same in an attempt to tackle the imbalance in class prevalences in the dataset.

The final predictions of our model are saved in Final_Predictions (epoch 20).csv.

Results

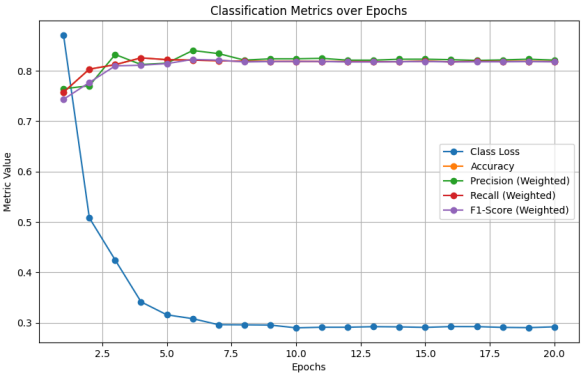

Figure 2: Classification Metrics Over Epochs for Custom CNN.

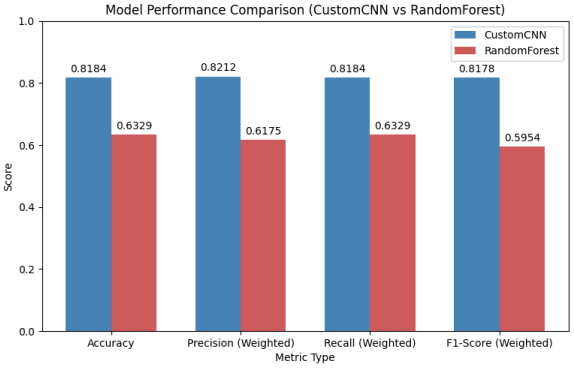

Figure 3: Model Performance Comparison Between CustomCNN and Random Forest

Please refer to the appendix for the confusion matrices for both the custom CNN and random forest implementations.

Analysis of Model Performance

Although the random forest implementation uses a well-known neural network for feature extraction (ResNet), the random forest layer for classification could be a contributor to its poor performance as indicated in figure 2.
The Custom CNN model eventually plateaus for all classification metrics with each hovering around the 80% mark. The metrics measured in this final epoch (20th epoch) outperforms the random forest implementation by roughly ~20% for each metric.
The confusion matrix for the Custom CNN has a relatively consistent prediction performance as indicated by the major diagonal being populated with notably high values. The confusion matrix for the random forest implementation indicates an underperformance in making accurate predictions. Many entries along the diagonal are 0, indicating that some classes were not positively classified at all. The varying number of positive classifications in each row across the two figures is due to the randomised train-test split generated before training.

Regarding Bounding Box estimations, both the custom CNN and the Random Forest models fail to accurately predict bounding boxes within a satisfactory threshold. The Custom CNN uses the IoU metric described in the Evaluation section, whereas the Random Forest implementation uses mean squared error. Although these metrics are not analogous to one another, their values for their respective models indicate a total inability to predict bounding boxes within some threshold. The IoU loss by epoch 20 the Custom CNN was 0.0004 and

the Mean-Square Error for the Random Forest was 2305 (predicted boxes vertices were at average 50 pixels away which would likely translate to a substantially low IoU).

Conclusions

This project focused on developing and evaluating machine learning models for detecting crop diseases in corn, pepper, and tomato plants using image data. Addressing crop diseases is essential for maintaining agricultural productivity and ensuring food security in African regions where agriculture plays a significant economic role.

From Figures 1 and 2, it is evident that the Custom CNN outperforms the random forest implementation of the neural network. After the 5th epoch of training, the Custom CNN plateaus in performance, maxing out at ~80% for all the metrics. It only takes a single epoch of training, however, for the Custom CNN to outperform the Random Forest implementation. Additionally, the confusion matrices of the two models shows another perspective of accuracy between the two models. The major diagonal of the Custom CNN Model is populated with high values, indicating a higher frequency of positive classifications. This outperforms the Random Forest which even contains instances where a class has no accurate predictions at all.

The successful classification of crop diseases using CNNs demonstrates the potential for machine learning to support agricultural practices in African countries. Early and accurate disease detection can enable timely interventions, reducing crop losses and supporting farmers' livelihoods. Additionally, the scalability of AI-driven tools aligns with the goal of enhancing agricultural sustainability and productivity in the region.

Future Work

While the CNN model performed well in classification tasks, both the Custom CNN and Random Forest models encountered difficulties in accurately predicting bounding boxes for disease-affected regions within the images. The metrics used—Intersection over Union (IoU) for the CNN and Mean Squared Error (MSE) for the Random Forest—indicated limited success in object detection. This limitation suggests that when looking at future work, more specialised architectures or additional components are necessary for effective localization of diseases in images.

Appendix

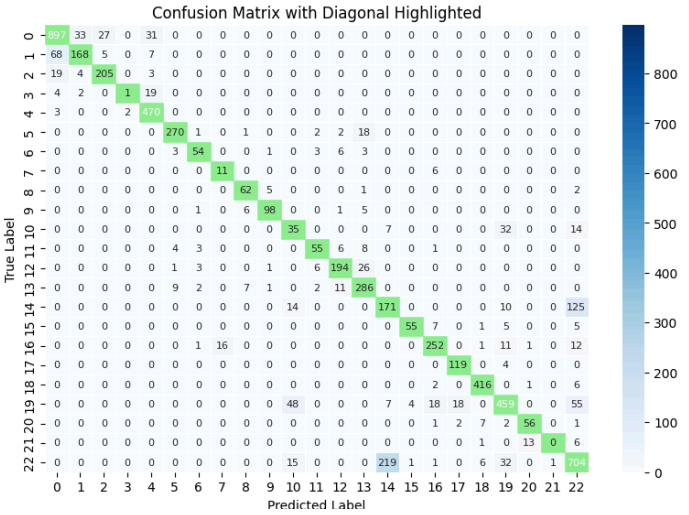

Figure A1: Confusion Matrix of Custom CNN Model.

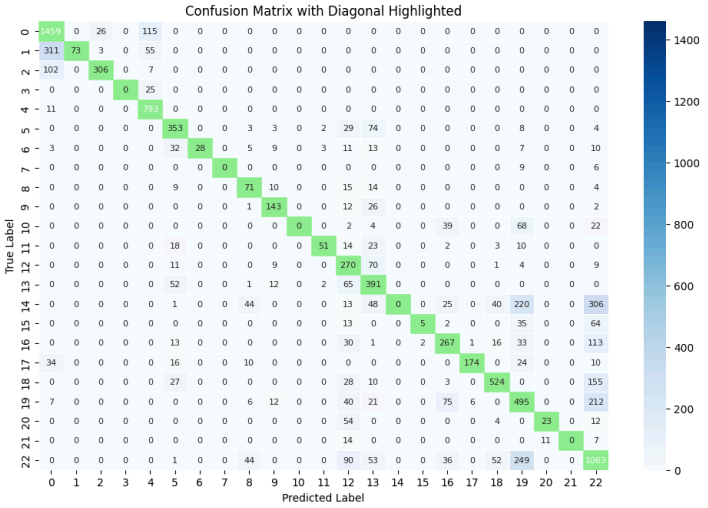

Figure A2: Confusion Matrix of Random Forest Model.