Neural Networking Assignment

By

Ukeje Chinedu Vincent

SID: 2339427  
 Date: 12TH April, 2024

**Task A**

This research project aims to develop a deep-learning model for the classification of cassava leaf diseases. The dataset used contains 21,397 images of cassava leaves spanning 5 classes (Healthy, Cassava Bacterial Blight (CBB), Cassava Brown Streak Disease (CBSD), Cassava Green Mottle (CGM) and Cassava Mosaic Disease (CMD)).

Figure 1: Cassava Bacterial Blight (CBB)

Figure 2: Cassava Brown Streak Disease (CBSD)

Figure 3: Cassava Green Mottle (CGM)

Figure 4: Cassava Mosaic Disease (CMD)

Figure 5: Healthy

**EfficientNetB3**, a convolutional neural network architecture optimized for both accuracy and efficiency, serves as a robust base model for this research. Leveraging its pre-trained weights offers several advantages, including high performance, computational efficiency, and transfer learning capabilities. By adapting the pre-trained EfficientNetB3 model to the specific task of leaf disease classification and fine-tuning its parameters on the provided dataset of leaf disease images, one can effectively harness the features learned from large-scale datasets while tailoring the model to the unique characteristics of the target dataset. This approach enables the development of a highly accurate and efficient model for identifying and classifying leaf diseases, with potential applications in agriculture and disease monitoring systems.

 Figure 6: Model Structure

The base model was first imported as a subpackage of the Keras library, and then loaded into the 6-layered sequential model. The other layers of the model are; GlobalAveragePooling2D, Flatten, Dense(with a ReLU activation), Dropout, Dense(with a softmax activation).

The GlobalAveragePooling2D layer acts as a feature extractor. This is how the model recognizes specific features of different leaf diseases and attributes them to specific diseases.

The Flatten layer further flattens the output of the preceding layers into a one-dimensional vector, preparing it for input into the subsequent fully connected layers. This process facilitates the extraction of high-level features from the pooled feature maps.

Rectified Linear Unit, or ReLU, is a famous activation function that is renowned for being computationally effective and for its capacity to solve the vanishing gradient issue.

The Dropout layer serves as a form of regularization during training by haphazardly setting a portion of the input units to zero, effectively preventing co-adaptation of neurons and mitigating overfitting. This motivates the network to pick up strong features that generalize well to unseen data, enhancing the model's performance on validation and test sets.

The final Dense layer with a softmax activation serves as the output layer of the model, responsible for producing probability distributions over the different classes involved in the task of classification. The function known as softmax normalizes the output scores into probabilities, enabling the model to generate dependable forecasts by assigning probabilities to each class label.

The chosen loss function for this model is Categorical CrossEntropy, while the chosen accuracy metric is Categorical Accuracy. The model ended training with a loss of 22.39% and a Categorical accuracy of 100%.

The chosen parameters in the model have been meticulously tailored to optimize performance in leaf disease classification. Callbacks such as ReduceLROnPlateau and EarlyStopping dynamically adjust learning rates and halt training when necessary, ensuring efficient convergence and preventing overfitting.

Leveraging a pre-trained EfficientNetB3 architecture with global average pooling and dense layers enables effective feature extraction and classification, benefiting from learned representations from ImageNet.

The use of a Cross-entropy loss function for categories with label smoothing and an Adam optimizer further enhances training stability and convergence speed. Additionally, a batch size of 16 balances computational efficiency and model convergence, while input images resized to 300x300 pixels standardize input dimensions for optimal performance. Overall, these parameters are meticulously selected to optimize both training efficiency and classification accuracy for the leaf disease classification task.

**Justification for the layers used in the model**

The reason i choose the use of six layers in the model and not 2, 3, or four layers is because six layers strikes a balance between complexity and efficiency, allowing the model to effectively extract features, learn non-linear relationships, prevent overfitting, and produce meaningful predictions in the leaf disease classification task. Here's a justification or rationale behind the selection of each of the mentioned layers that was included in the model:

1. **EfficientNetB3:** this model was chosen as the base model for leaf disease classification due to its optimal balance between accuracy and efficiency. Pre-trained on large-scale datasets like ImageNet, EfficientNetB3 possesses rich feature representations learned from diverse visual concepts, facilitating effective transfer learning. By fine-tuning EfficientNetB3 on the leaf disease dataset, the model benefits from learned representations, reducing the need for extensive labelled data and computational resources during training. Its hierarchical feature extraction capabilities enable generalization across diverse leaf disease images, ensuring accurate classification despite variations in appearance and texture. Additionally, EfficientNetB3 enjoys widespread adoption and support within the machine learning community, providing readily available implementation code, pre-trained weights, and documentation for seamless integration into projects. Overall, EfficientNetB3's combination of accuracy, efficiency, transfer learning capabilities, and community support makes it a robust choice for leaf disease classification tasks in agricultural contexts.

2. **GlobalAveragePooling2D:** Global average pooling is a common method for shrinking feature maps' spatial dimensions without sacrificing crucial spatial information. By calculating the average value of each feature map, GlobalAveragePooling2D summarizes the most salient features in a spatially invariant manner. This allows the model to capture global context and spatial relationships in the input images effectively.

3. **Flatten:** After feature extraction, the output of convolutional and pooling layers is typically a multi-dimensional tensor. The Flatten layer converts this tensor into a one-dimensional vector, enabling seamless connectivity to the subsequent fully connected layers. This flattening process preserves the learned spatial hierarchies while facilitating the extraction of high-level features for classification.

4. **Dense (with ReLU activation):** Fully connected layers, sometimes referred to as dense layers, are essential for deciphering the non-linear relationships seen in the data. By introducing non-linearity, the ReLU activation function enables the model to capture complex relationships and patterns within the feature space. By stacking multiple Dense layers with ReLU activations, the model gains increased representational capacity, enabling it to learn and discriminate between different classes effectively.

5. **Dropout:** A regularisation method called dropout is frequently used to reduce overfitting and enhance deep neural network generalisation. Dropout increases model robustness by forcing the network to acquire redundant representations and reducing reliance on particular neurons by randomly deactivating a portion of the neurons during training. preventing co-adaptation of features. This leads to improved performance on unseen data and reduces the risk of overfitting.

6. **Dense (with SoftMax activation):** The output layer, which is the last Dense layer with a SoftMax activation, is in charge of generating probability distributions over the many classes involved in the classification task. By giving each class label a probability, the SoftMax function normalises the output scores into probabilities, enabling the model to produce predictions with confidence. This enables the model to output meaningful predictions and quantify its confidence in each class assignment.

Overall, the combination of these layers in the model facilitates effective feature extraction, non-linear transformations, regularization, and classification, contributing to the learning capacity of the model discriminative features and make accurate predictions on the given task.

**PART B**

The employed strategy, Dominant Class Flipping, involves targeted manipulation of labels belonging to a specific class. In this scenario, the class chosen for manipulation is Cassava Mosaic Disease (CMD), identified as the most populated class within the dataset. By intentionally flipping labels from Cassava Mosaic Disease (CMD) to Cassava Green Mottle (CGM) class, the strategy aims to introduce confusion into the dataset and subsequently challenge The capacity of the model to accurately distinguish between the two classes.

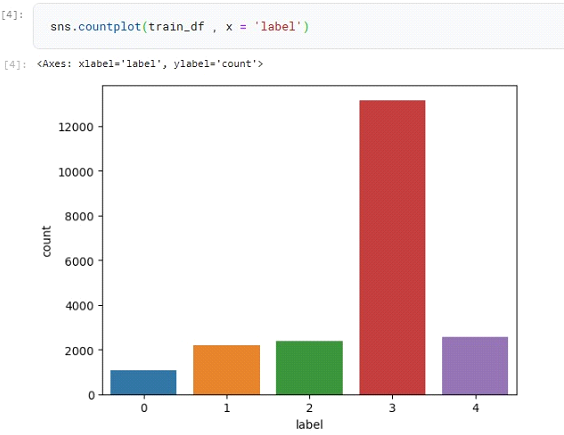
Cassava Mosaic Disease (CMD), being the most prevalent class, is strategically chosen for label manipulation to effectively disrupt the class distribution within the dataset. By perturbing the dominant class, the manipulation strategy aims to simulate real-world scenarios where certain classes may dominate the dataset, leading to biased model predictions. Manipulating CMD labels to populate the CGM class introduces variability and complexity, enhancing the robustness and generalization capability of the trained model. 



Figure 7: label value counts and chart before flipping

**Impact of Label Flipping on Model Performance:**

1. Specifically targeting 5% of the labels from the Cassava Mosaic Disease (CMD) class resulted in a notable decline in validation accuracy. The validation accuracy dropped by 14.29% compared to training with the original, unaltered dataset. This decrease in accuracy underscores the significant impact of label manipulation on the capacity of the model to generalize and correctly classify unseen data samples. Concurrently, the loss increased by 26.16% compared to training with the original dataset. The substantial increase in loss reflects the model's heightened difficulty in correctly classifying samples due to the introduced confusion from label flipping. The rise in loss suggests that the model struggles to effectively learn and differentiate between the manipulated classes, resulting in diminished performance.

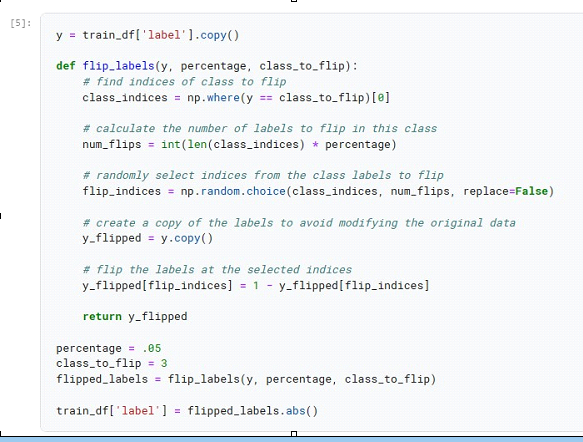
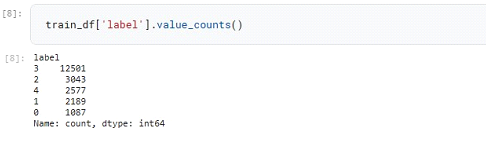


Figure 8: function for flipping 5% of class 3



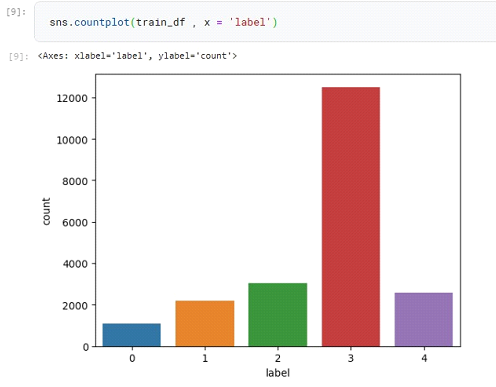


Figure 9: Label value counts and chart after 5% flip

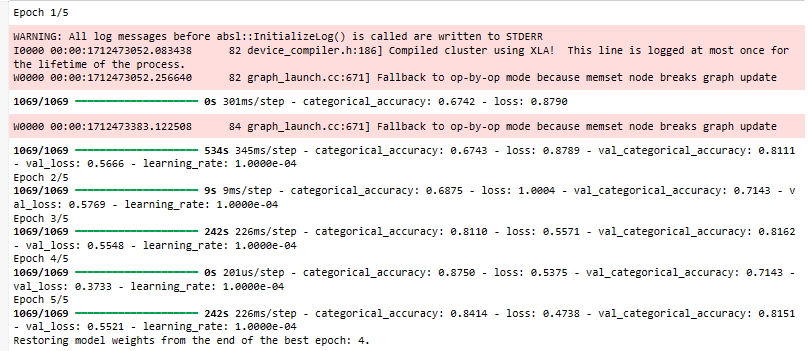
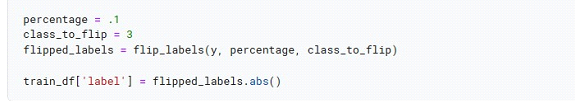
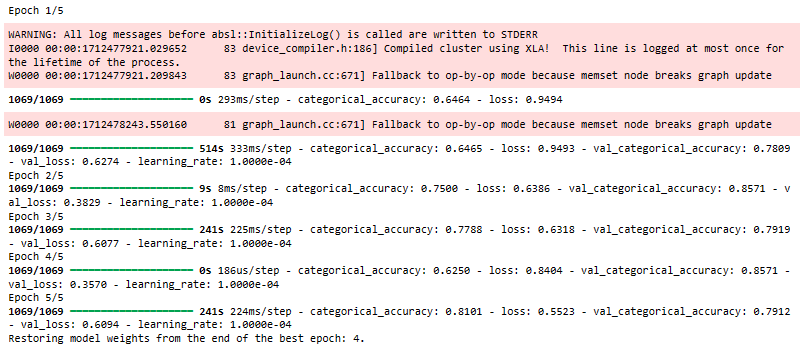
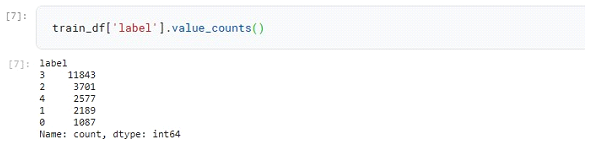


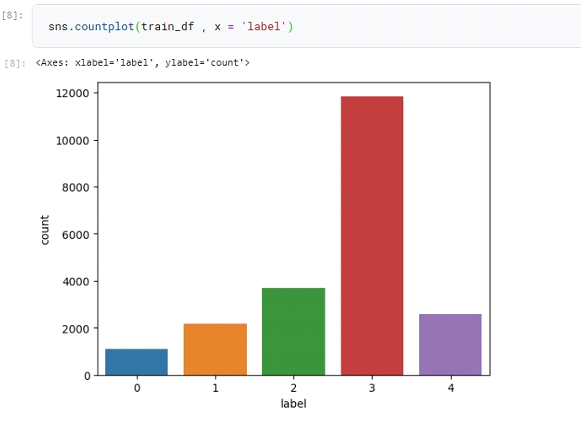
Figure 10: model training history for 5% label flip in class 3

1. When 10% of the Cassava Mosaic Disease (CMD) labels were flipped, the model achieved a validation accuracy of 85.71%. Despite the label manipulation, the validation accuracy remained consistent with the previous experiment, indicating the model’s capacity to adapt to the increased level of label perturbation. However, the validation loss increased significantly to 49.53% compared to the original dataset. This substantial rise in loss suggests that the model struggled more in correctly classifying samples due to the higher degree of label manipulation.

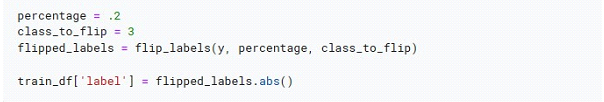
 Figure 11: image of the function to flip Cassava Mosaic Disease (CMD) by 10%

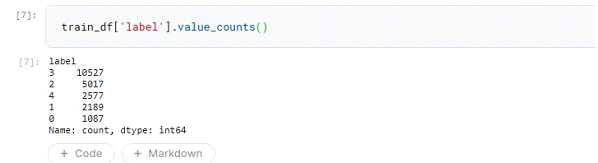
 Figure 12: Model training history for 10% Cassava Mosaic Disease (CMD) flip

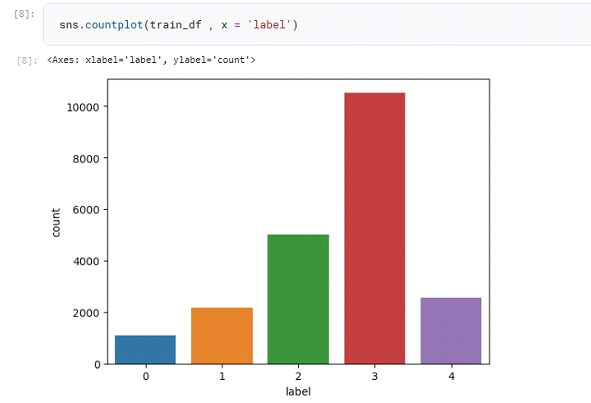


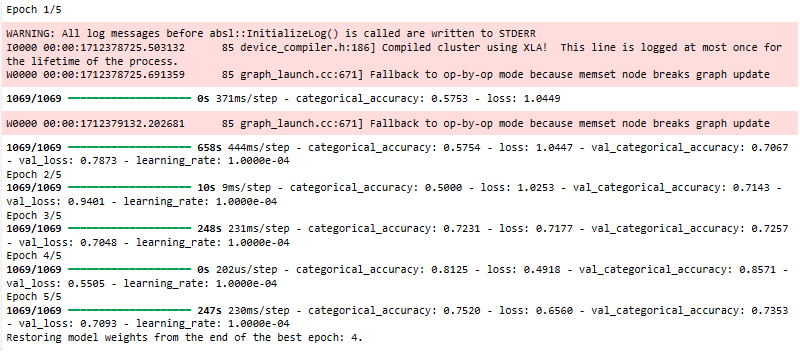
 Figure 13: Label value count and chart after 10% flip

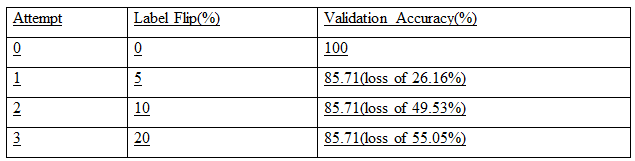
1. When 20% of the Cassava Mosaic Disease (CMD) labels were flipped, the model achieved a validation accuracy of 85.71%. This accuracy remained consistent with the previous experiments despite the higher flipping rate, indicating that the model maintained its ability to classify samples to some extent. However, the validation loss increased further to 55.05%, reflecting the model's increased difficulty in correctly classifying samples due to the higher degree of label manipulation. The consistent validation accuracy suggests that the model could still effectively differentiate between classes, albeit with increased complexity introduced by the higher flipping rate. However, the significant increase in validation loss indicates a notable degradation in the model's performance compared to the original dataset. This suggests that while the model may have adapted to some extent to the manipulated labels during training, it struggled more with the perturbed dataset, resulting in higher loss values.

Figure 14: Image of the function to flip CMD by 20%



 Figure 15: value count and chart after 20% flip

 Figure 16: Model training history for 20% CMD flip

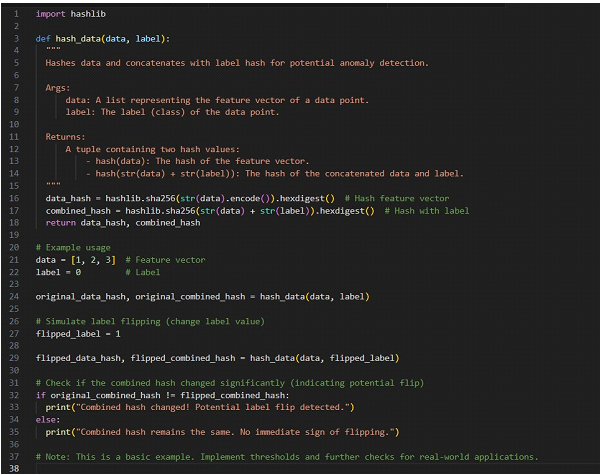
 Table 1: Summary

**PART C**

Hashing can play a crucial role in detecting label manipulation by ensuring data integrity and authenticity. Hash functions can be applied to both the stored dataset and labels to generate unique hash values. Any alteration in the dataset, including label manipulation, would result in a different hash value, enabling detection of tampering.

For each data point, all the features except the label should be considered, and a hash function should be applied to this feature vector, generating a unique hash value. the original label should then be concatenated with the feature vector hash, while applying the same hash function again to generate a new hash value.

During training, the second hash value for each data point would be monitored for irregularities. If the label is flipped (changed), the feature vector might remain the same, but the second hash will be entirely different due to the altered label. However, Hashing only captures changes in the feature vectors and labels. If the attacker cleverly modifies the features to still represent the original class but trigger a different label prediction by the model, this might go undetected. Therefore, It is recommended that this technique be used alongside another method like manually checking the count of the labels to make sure it’s not altered while using the hash functions to ensure nothing has changed either.

Also, integrating regularization techniques, such as adversarial training, is useful in making the model more robust against label noise and manipulation. Adversarial training involves augmenting the dataset with adversarial crafted examples designed to expose vulnerabilities in the model. By intentionally injecting manipulated labels into the training data, the model learns to distinguish between genuine and manipulated samples, thereby enhancing its robustness against label manipulation. Figure 17: Code Sample for using hash function in tracking changes

The hash ipynb file with name ‘CODE FOR PART C’ has also been submitted on GITHUB where i submitted my PYHTON CODES for assignments.