Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Utilizing Ant Colony Optimization With Deep Learning For Wheat Disease Detection

Amanjot Kaur Randhawa[1] , Dr. Nirmal Kaur[2]

[1]Assistant Professor, PhD Scholar, Sant Baba Bhag Singh University, Jalandhar, Punjab, India
Email Id : amanrandhawa91.ar.ar@gmail.com
[2]Associate Professor, Department of Computer Science, Sant Baba Bhag Singh University, Jalandhar, Punjab, India
Email Id: nparhar.sbbs@gmail.com

Wheat is a major global food crop that provides over 20% of the calories for humans worldwide. One of the primary staples for a majority of the world's population, making sure that it remains healthy and productive is invaluable. The awareness and early identification of wheat diseases are important for preserving crop yield losses, and to curb pathogen dissemination. By identifying early on, it helps not just in saving this key food source but also to minimize the risk of broad scale outbreaks which may threaten food supply chains. The use of extensive chemical treatments are greatly reduced through effective disease management, hence reducing the likelihood that any harmful chemicals will end up in products made from crops and enter our food chain. In addition, the use of disease detection systems provides resistant measures and innovative methods that allow for more sustainable agricultural practices by increasing resilience to a greater threat. In this paper, we have applied Ant Colony Optimization (ACO) to find out the best hyper-parameters for VGG19 and DenseNet121 models. Instead, we have used ACO that has provided us with creating these models in a super-performing method which can take the shape of precise and efficient training outcomes. It helps increase the accuracy of disease detection along with performance improvements in overall models to foster more accurate and scalable solutions for agriculture technology. With these developments, we hope to contribute towards the larger objective of sustainable and resilient agricultural operations.

Keywords: Deep Learning, Wheat Disease Detection, Ant Colony Optimization (ACO), VGG19, DenseNet121

1 Introduction

Concerning wheat is a major global crop and an important human food. In its production, it occupies the largest cultivation area and ensures food security in many countries around the world (Genaev et al., 2021). Parts of the temperate zone are especially dependent, yet demand for maize is rising rapidly in industrializing and urbanizing nations. Wheat is also a source of protein, with raw wheat being% 13.3 to %15 yet getting you large measures of protein if concentrated into wheat gluten (which are like nutriment or meat), vitamins and minerals(b group especially) dietary fiber(U.S.: fiber)and phytochemicals have potential health benefits(roughly favoring your digestive tract). Carbs: Wheat is a great source of energy through carbohydrates that we all know for, but it also adds other nutrients to our body. Best not to forget that plants provide us with a whole lot more than just proteins, they are also rich sources of fiber and smaller amounts of lipids, vitamins (+colors)and minerals AND phytochemicals which can be bundled up in all sorts of ways for promoting good health... The take home message is you need variety! (Shewry & Hey, 2015). Early detection of crop diseases is essential to control its spread and can reduce economic issues in sustainable agriculture (Lu et al., 2017) The major factor that leads to poor quality of crop and decreased yield is plant diseases. The early detection is much required to overcome the worst stage and huge economical loss of agriculture (Gaikwad & Musande, 2017). Historically, wheat disease identification was accomplished through manual detection which brought problems such as objectivity, inefficiency and low accuracy. Today, with the rise of technology-assisted methodologies such as spectral analysis, machine learning and deep learning are being increasingly used for wheat disease detection (Fang et al., 2023). Some of the methods to check for diseases in wheat are discussed below: Dixit & Nema (2018) came up with a process that uses image processing and machine learning techniques based on species, visual severity level. The general leading approach starts with image acquisition, and proceeds towards preprocessing to prepare the images before analyzing. Then, the segmentation step separates the diseased leaf from those leaves and feature extraction collects information for these specific regions. This is followed by feature selection which picks out the most important features for classification. The second component is machine learning, where the classifiers such as Support Vector Machine (SVM), Neural Networks or k-means clustering have been utilized to detect and classify these diseases based on extracted features. Finally, the paper presents several reviews of both studies and methods demonstrating how well different algorithms perform compared to each other as a whole and provide insights into what their strengths and weaknesses are. In conclusion, it emphasizes that image processing and machine learning have been proven to be a successful combination in improving the accuracy and speed of wheat leaf disease detection systems. Detection of wheat rust diseases using deep learning models Sood and Singh (2020) discussed a technique to detect the disease. It starts from the data augmentation of a small dataset to generate synthetic semi-supervised samples due to scarcity in targets. For the problem ResNet50 and VGG16 models are used which use transfer learning. For both of the models, images have been preprocessed by resizing them to 224x224 pixels before taking a split into training and validation in ratio 90:10. The model is fine-tuned with hyperparameters to improve the performance: batch size = 32, initial learning-rate of lr=0.01 (with decay), optimizer = adam Convolutional Neural Networks (CNN) are employed to build the deep learning models for processing raw images and various operations such as convolution, max-pooling and activation functions like ReLU etc.. diversely help in feature extraction. It takes the features extracted and passes them from a flatten layer to two FC layers, which are concatenated into two categories of softmax for leaf rust, stem cyst and healthy wheat. I conducted training and validation of the models in an environment powered by a GPU (Google Colab) organism with excellent computational resources. The paper also evaluate different sizes of samples (100, 3000 and all available images) showing that the most data one can use gives higher accuracy on a given size. The Results show that the model VGG16 has higher accuracy compare to ResNet50 and achieve highest classification accuracy of 99.07%. Finally, with respect to the efficacy of VGG16 model in this study; future research including additional enhancements on features and noise removal techniques can further improve disease detection performance. We developed a Fusarium head blight (FHB) prediction model that enhanced an existing strategy-based logistic regression with a data-driven k-nearest neighbors (KNN)data lab(Li et al., 2022). Remote sensing and meteorological data were used to select predictive factors, while logistic regression was applied for determining the factor weights of disease occurrence and development. The weights were then used with model like KNN to enhance the prediction accuracy. This included the procedure of splitting the data into train and test set, cross-validation with n_splits = 5 to find out best equivalent "k" value as well evaluation metrics - Accuracy, F1 score and ROC curves. Higher accuracy (0.88 and 0.92) and F1 scores respectively of the logistic-KNN model than the traditional KNN model. It showed increased specificity and sensitivity, reporting an AUC of the ROC analysis. This paper critically demonstrates that bringing disease mechanism insights into KNN models can better improve prediction accuracy and stability than not; while the next step suggests both improving computational complexity of current phenomenon interpretations, as well as developing more refined classification on severity of diseases. Cheng et al., 2023: Wheat disease detection improves in this paper with the help of an attention mechanism that is equipped within a convolution neural network. Here, this block takes in feature keypoints and pulls positional info such as coordinates of points on the heatmap itself to create attention maps which along with original features allow our model to focus more strongly upon important regions for better feature extraction. This block is employed across multiple CNN architectures like ResNet and GoogLeNet in the study; it accomplished significant improvements in accuracy e.g., up to 2.7 % on Resnet way deeper than before. We validate on the PlantDoc dataset for different plant disease types, and we achieve a mAP (mean average precision) of 0.51 which demonstrates strong performance across test classes. The accuracy of this attention mechanism is much better compared to other architectures like Squeeze-Excitation and so on. The results obtained by the implementation, PyTorch-based and validated on different hardware platforms demonstrate effectiveness of a new form for real-world applications.

2 Materials and Methods

2.1 Dataset

In this paper, a wheat dataset has been utilized comprising a total of 407 images (102 Healthy Wheat Images, 97 Septoria Wheat Images, 208 Stripe Rust Wheat Images). The distribution of this dataset into training and test dataset is depicted in table 2.1.

Table 2.1

Dataset Division into training and test sets

Wheat Condition	Training Set Images	Test Set Images
Healthy	81	21

Septoria	77	20
Stripe Rust	166	42

A sample image from each wheat condition is depicted below:

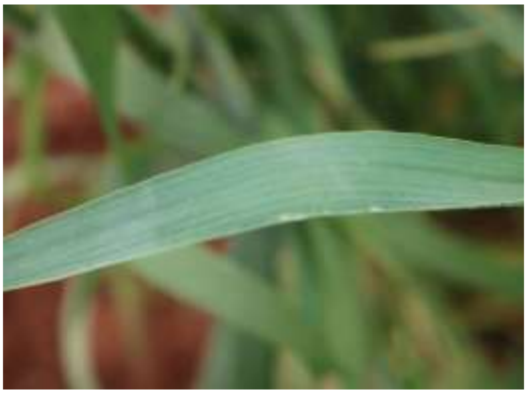


Fig 2.1 Healthy

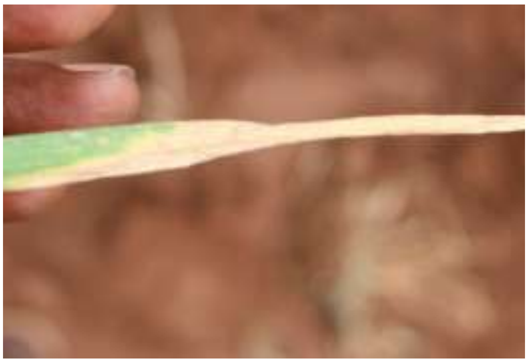


Fig 2.2 Septoria Leaf

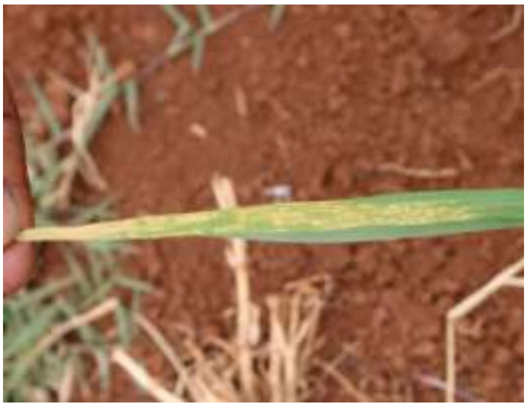


Fig 2.3 Stripe Rust Leaf

2.2 Overview

The code is designed to train deep learning models using both the DenseNet121 and VGG19 architectures, with hyperparameters optimized through an Ant Colony Optimization (ACO) approach. It begins by setting up directories for training and validation datasets, and specifies image dimensions, batch size, and the number of classes. Data augmentation and preprocessing are handled using the `ImageDataGenerator` from Keras, which applies transformations like shearing, zooming, and flipping to the training images, while validation data is simply preprocessed. The models, either DenseNet121 or VGG19, are created by first loading the base architecture without the top fully connected layers and then adding custom layers including global average pooling, dense layers, dropout for regularization, and a final softmax layer for classification. The ACO algorithm is used to optimize key hyperparameters such as learning rate, batch size, and the number of epochs. This is done by initializing a pheromone matrix, which is updated iteratively based on the validation accuracy achieved by models trained with different hyperparameter combinations. Multiple ants in each iteration explore different hyperparameter values, and the pheromone levels are adjusted to guide subsequent searches toward more promising configurations. After completing the optimization, the best hyperparameters are selected to train the final model, which is then saved for future use. This process is applied to both the DenseNet121 and VGG19 models to compare their performance and determine the most effective model and hyperparameter configuration. The overall process is described in the form of a flowchart in Fig 2.4.

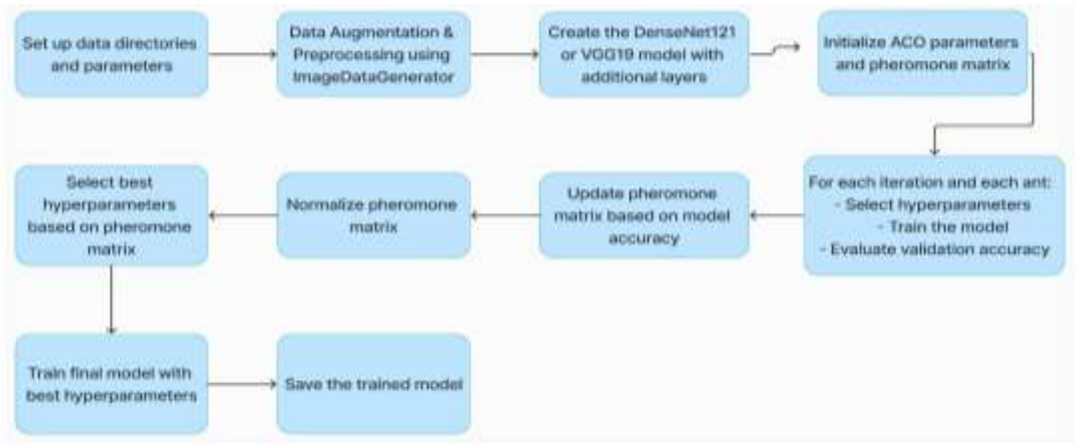

Set up data directories and parameters
Data Augmentation & Preprocessing using ImageDataGenerator
Create the DenseNet121 or VGG19 model with additional layers
Initialize ACO parameters and pheromone matrix
For each iteration and each ant:
- Select hyperparameters
- Train the model
- Evaluate validation accuracy
Update pheromone matrix based on model accuracy
Normalize pheromone matrix
Select best hyperparameters based on pheromone matrix
Train final model with best hyperparameters
Save the trained model

Fig 2.4 Overall Process

2.3 CNN Models Used

The CNN models used in this paper are briefly explained in the following sections.

2.3.1 VGG19

VGG19 is a model from the Visual Geometry Group (VGG) architecture family that is known for being simple, with few hyper-parameters and designed to be used in computational imaging tasks. VGG19 is a model that contains 16 convolutional layers, followed by three fully connected and five max-pooling ones. It was developed at the Oxford Visual Geometry Group. This is because each convolutional layer uses only 3x3 filters with stride of length one in order to allow the network to capture fine-grained details without pushing up against resource constraints due to parameter saturation. Deeper architecture produces deeper representation (crucial patterns) of the image, able to learn complex features and produce better results. VGG19 is simple to use and considered suitable for a wide variety of problems due to its depth lieu design:model architecture. Fig 2.5 depicts the architecture of VGG19 model:

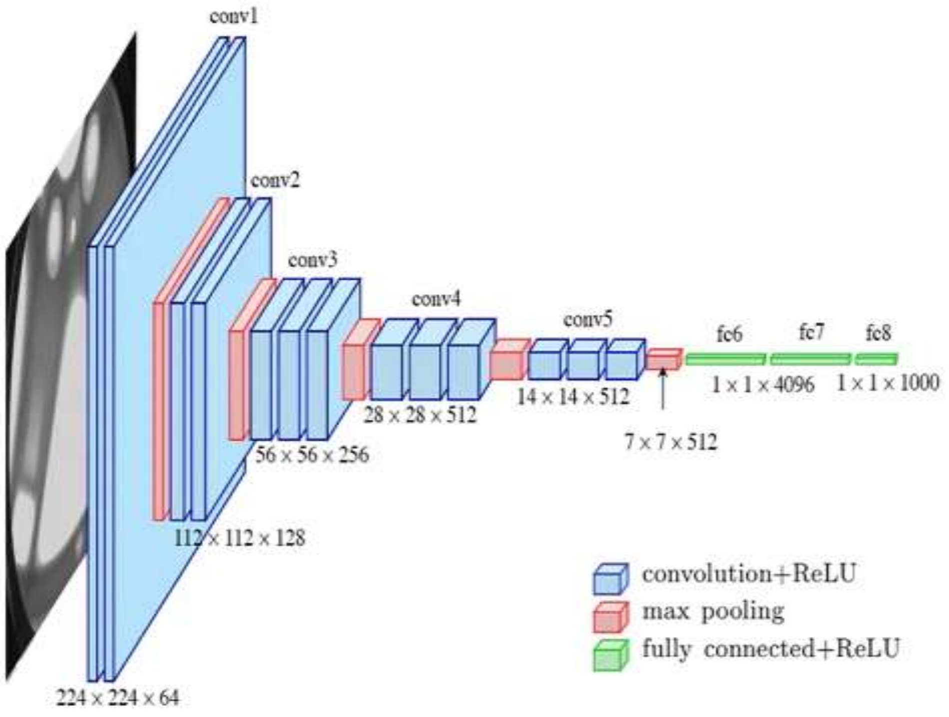


Fig 2.5 VGG-Net Architecture (Bangar, 2022)

2.3.2 DenseNet121

DenseNet(121) is a Convolutional Neural Network having an impact in increasing gradient flow. DenseNet121 is a DNN which can go up to 121 layers with each layer being inputted by all its preceding ones creating so-called densely connected blocks, since it falls in the DenseNets family and therefore has inherited that name. This way, every layer can learn to map only the residual functions and so feature reuse is encouraged across different layers- this reduces the number of parameters as well. The network is made up of multiple interconnected convolutional layers arranged in blocks interspersed with transition layers that decrease the spatial dimensions and number of feature maps. DenseNet121 was introduced with the claim of having properties of better gradient flow across different depths which helps in training deep networks and also greater image recognition accuracy. Fig 2.6 depicts the architecture of DenseNet121 model:

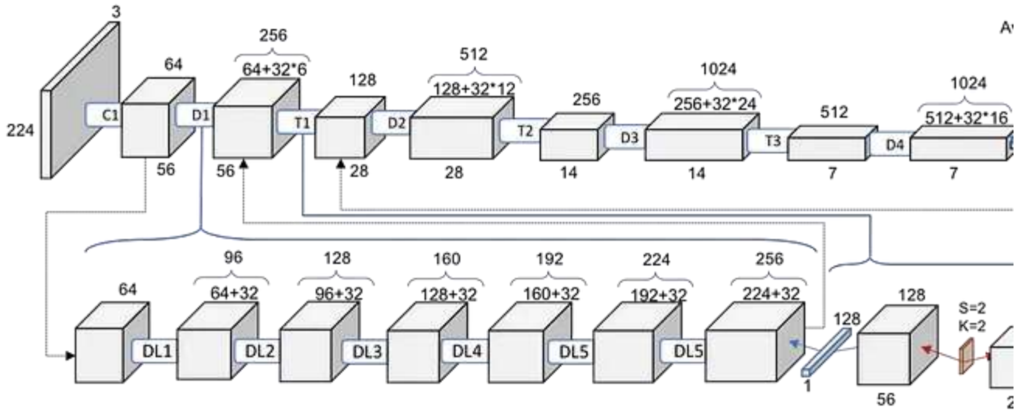


Fig 2.6 DenseNet-121 (Ruiz, 2018)

2.4 Ant Colony Optimization

An Ant Colony Optimization is a type of metaheuristic algorithm used to solve complex optimization problems and this method stimulates the behavior(oop.)ofstream ants. It gets its inspiration from the foraging behavior of ants in nature. Ants find the food they look for by smelling pheromones on their way. The higher the number of pheromones on a path, the more attractive it is for other ants to take that road leading into an auto-catalytic loop where over time the shortest path will get strengthened. It is very effective when it comes to searching for the good solutions in regular search spaces, huge and complicated ones as well.For example;Traveling Salesman Problem (TSP); needs to find out there shortest path starting from a city then jumps over other cities and return startin point again.

Components in the Code

1. **Hyperparameters to Optimize**:
a. **Learning Rate (learning_rate)**: Determines how much to update the model's weights with respect to the gradient.
b. **Batch Size (batch_size)**: Number of training samples used to compute a single gradient update.
c. **Epochs (epochs)**: Number of times the learning algorithm will work through the entire training dataset.
2. **Pheromone Matrix**:
a. This 3D matrix represents the desirability (pheromone levels) of each combination of hyperparameters. It is initialized with ones, meaning all combinations start with equal desirability.
b. The dimensions of the matrix correspond to the different values of learning rate, batch size, and epochs.
3. **ACO Process Flow**:
a. **Initialization**:
i. Initialize the pheromone matrix with equal values (all set to 1).
ii. Set the number of ants (num_ants) and iterations (num_iterations).

b. **Ants Select Hyperparameters**:
  i. Each ant selects a combination of hyperparameters based on the pheromone levels.
  ii. The probability of selecting a specific hyperparameter value is proportional to its pheromone level. This is implemented using the select_hyperparameters() function, which chooses values for learning_rate, batch_size, and epochs based on their respective pheromone levels.

c. **Model Training and Evaluation**:
  i. For each selected combination, the model is created and trained using the evaluate_model() function.
  ii. The performance is measured by validation accuracy after training. This accuracy is used as feedback to update the pheromone matrix.
  iii. **Pheromone Update Rule**:
1. The pheromone level corresponding to the chosen hyperparameter combination is increased by the validation accuracy, rewarding effective combinations.
2. This follows the idea that paths leading to good solutions should be reinforced, making them more likely to be selected by future ants.

d. **Pheromone Evaporation and Normalization**:
  i. After each iteration (where all ants have completed their evaluations), the pheromone matrix is normalized. This step simulates pheromone evaporation, preventing premature convergence to a suboptimal solution by ensuring that even less explored hyperparameter combinations still have a chance of being chosen.
  ii. Normalization is done by dividing each element of the pheromone matrix by the total sum of the matrix, ensuring the pheromone levels remain in a probabilistic form.

e. **Iteration**:
  i. The process repeats for the specified number of iterations. Over time, the pheromone matrix converges towards representing the best hyperparameter combinations.

f. **Final Selection**:
  i. After the iterations are complete, the final set of hyperparameters is selected based on the pheromone matrix. The most reinforced (highest pheromone level) combination is chosen as the optimal hyperparameters.

4. **Final Training with Optimized Hyperparameters**:
a. The model is retrained using the best hyperparameters found by the ACO. This ensures that the training process leverages the optimal configuration discovered through the iterative optimization.

5. **Saving the Model**:
a. The final trained model is saved, allowing for future use or deployment.

Detailed Explanation of Key ACO Concepts in the Code

1. **Transition Probability**:
- When selecting hyperparameters, each ant uses a probability distribution derived from the pheromone levels. The higher the pheromone level, the more likely a particular value is to be chosen.
    - This mechanism allows the algorithm to explore different combinations while gradually converging on the most promising ones.

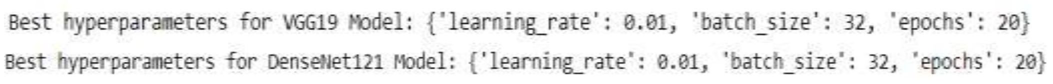
2. **Pheromone Update Mechanism**:
   - The pheromone update is key to reinforcing good solutions. In this code, the validation accuracy directly influences how much pheromone is added, meaning that better-performing hyperparameters are favored.
   - Over iterations, this leads to a focus on the best combinations, improving the efficiency of the search.
3. **Pheromone Evaporation**:
   - Though not explicitly shown as a decay factor in the code, the normalization step after each iteration effectively acts as evaporation, preventing any single combination from dominating too quickly. This helps maintain diversity in the search process.

Advantages of Using ACO Here

- **Adaptive Exploration**: Unlike a grid search or random search, ACO adapts over time, focusing on the most promising regions of the hyperparameter space.
- **Efficiency**: By leveraging collective behavior and positive feedback, ACO can often find good solutions faster than exhaustive search methods, especially in large and complex search spaces.
- **Global and Local Search Balance**: The combination of pheromone reinforcement and evaporation helps balance exploration of new possibilities with exploitation of known good solutions.

3 Results

After utilizing the Ant Colony Optimization to identify the best hyperparameters for the training of VGG19 and DenseNet121 models, the best hyperparameters thus obtained are depicted below:



```
Best hyperparameters for VGG19 Model: {'learning_rate': 0.01, 'batch_size': 32, 'epochs': 20}
Best hyperparameters for DenseNet121 Model: {'learning_rate': 0.01, 'batch_size': 32, 'epochs': 20}
```

The Fig 3.1 depicts the ‘Epochs vs Loss’ and ‘Epochs vs Accuracy graphs’ for the VGG19 model:

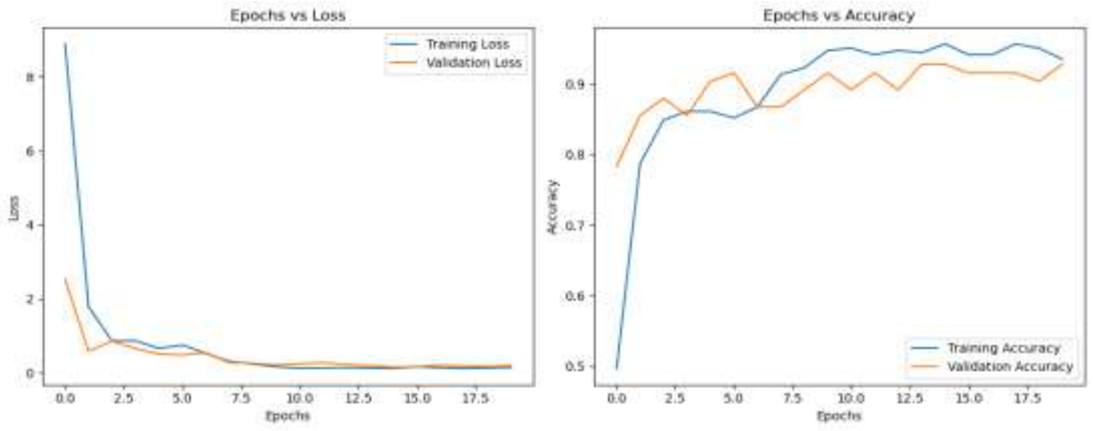


Fig 3.1 VGG19 Epochs vs Loss & Epochs vs Accuracy graphs

The Fig 3.2 depicts the ‘Epochs vs Loss’ and ‘Epochs vs Accuracy graphs’ for the DenseNet121 model:

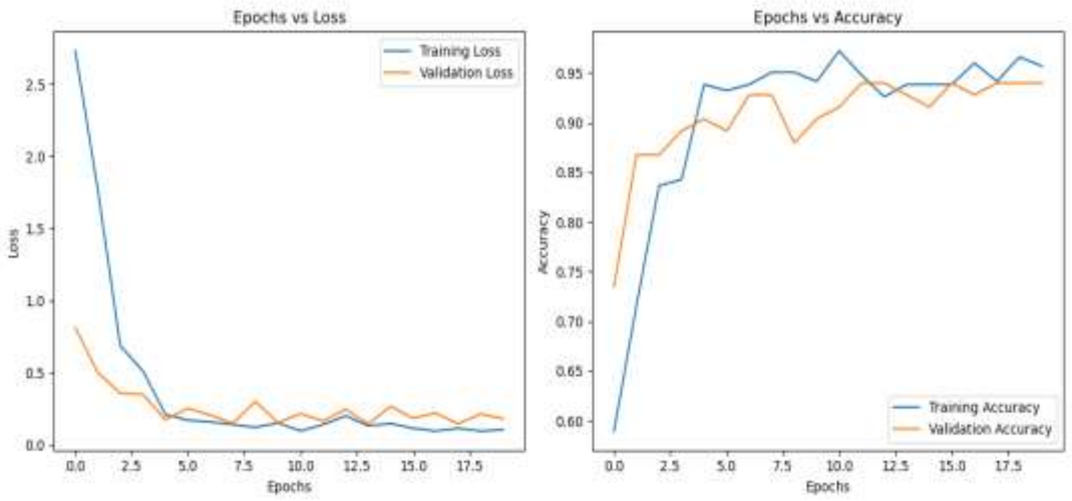


Fig 3.2 DenseNet121 Epochs vs Loss & Epochs vs Accuracy graphs

Table 3.1

Accuracies & Losses of the Models

Model	Accuracy	Validation Accuracy	Loss	Validation Loss
VGG19	0.9352	0.9277	0.1466	0.1950
DenseNet121	0.9568	0.9398	0.1070	0.1824

The following metrics are compared in the Table 3.1:

Accuracy: How well the deep learning model performs on training data The number of correct predictions is the proportion to all trained data, which are known as Accuracy(ACC) Accuracy: How many examples you got correct overall on all the labels when compared to how many total. The fact that the training accuracy is high - this means the model learned well from data in general, but it does not guarantee decent performance on unseen data.

Validation Accuracy: The validation accuracy is the proportion of examples for which the model correctly predicts its label from a new pool that was not part of your training sample. The Investigation regarding how this model did when predicting on its validation set (this value is the higher percent of correctly predicted values) Tracking validation accuracy can also be an early indicator of overfitting, that is a model does well on training data but not so ideally with unseen examples.
Loss: Loss measures the deviation of prediction made by model from true output. A loss function (like cross-entropy for classification, or mean squared error for regression) that quantifies the difference between the predicted outputs and actual target values. Training has the aim of reducing this loss. A low training loss means the model is fitting well to the train data.
Validation Loss: Validation loss is almost the same as the training loss except that it is calculated on validation set. It gives an idea of how well the model would be able to predict new unseen data. If the validation loss is much greater than training loss, this might mean that the model has been overfitting to the training data and what it learned does not generalize well to unseen new input points.
Monitoring these metrics during training helps in assessing the model's performance and deciding when to stop training to avoid overfitting while ensuring good generalization. Table 3.1 presents a comparison between two deep learning models, VGG19 and DenseNet121, based on their performance metrics. The VGG19 model achieved an accuracy of 93.52% on the training data and 92.77% on the validation data, with a corresponding training loss of 0.1466 and a validation loss of 0.1950. On the other hand, the DenseNet121 model demonstrated a higher accuracy of 95.68% on the training data and 93.98% on the validation data, with a lower training loss of 0.1070 and a validation loss of 0.1824. These metrics indicate that DenseNet121 outperformed VGG19 in both accuracy and loss, suggesting better generalization and training efficiency.

4 Conclusion

This study investigates the integration of Ant Colony Optimization (ACO) with deep learning to enhance the performance of two popular architectures, DenseNet121 and VGG19. By concentrating on optimizing critical hyperparameters—learning rate, batch size, and the number of epochs—the ACO algorithm effectively explores the extensive hyperparameter space, identifying configurations that lead to superior model performance. The experimental findings indicate that DenseNet121 outperformed VGG19 in terms of both accuracy and loss. DenseNet121 achieved a higher accuracy of 95.68% on the training data and 93.98% on the validation data, compared to VGG19's 93.52% and 92.77%, respectively. Additionally, DenseNet121 exhibited a lower training loss of 0.1070 and a validation loss of 0.1824, suggesting better training efficiency and generalization to unseen data compared to VGG19's training loss of 0.1466 and validation loss of 0.1950. These results highlight DenseNet121's effectiveness in this context and validate the use of ACO for hyperparameter tuning.

ACO is performing well in this study as its adaptiveness nature allows it to explore the hyperparameter space and diversity into a new search path while exploiting recruitment of promising solutions. ACO guides the search process to those configurations which have higher validation accuracy using Pheromone matrix (which they keep updating iteratively over iterations after each model training). Note that traditional techniques like grid search or random search cannot adjust, follow gradient and may take longer to converge as not providing any feedback about how well a solution actually performs. ACO also allowed our hyperparameters optimization process to dynamically search with the aid of pheromone evaporation and normalization procedures, refraining from convergence towards suboptimal solutions. This maintained a lot of diversity into the search process, allowing the algorithm to continue searching even those configurations which were not so obvious and could perform better. These observations have significant practical ramifications for the area of deep learning, most notably in those domains where model performance matters. Effective tuning of hyperparameters can certainly improve model accuracy and generalization - both mandatory for real-world applications, like medical image analysis to autonomous systems or natural language processing. The contrast and flexibility of the ACO approach are what make it so beneficial to beta developers, experts who want other deep learning models to perform at their outermost extremes. This study concludes that combining ACO with deep learning does not only improve the prediction performance but allows for a more structured and faster method of hyperparameter tuning. Comparing the two figures for model performances, this emphasizes how using a compatible architecture choice is just as essential alongside hyper parameter optimization(DenseNet121 vs VGG19). Future research might be able to further expand on these results, in part by evaluating the ability of ACO to navigate other deep learning architectures and tasks, as well tweaking it for use with additional optimization challenges. These encouraging results, applied to the field of multiple inputs and outputs for deep learning with ACO by this work can open doors for broader use of other practical fields as a powerful way to get optimal overall performance in their specific domains.

References

1. Ruiz, P. (2018, October 18). Understanding and visualizing DenseNets. Towards Data Science. https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a
2. Bangar, S. (2022, June 28). VGG-Net architecture explained - Siddhesh Bangar. Medium. https://medium.com/@siddheshb008/vgg-net-architecture-explained-71179310050f
3. Dixit, A., & Nema, S. (2018). Wheat leaf disease detection using machine learning method-a review. Int. J. Comput. Sci. Mob. Comput, 7(5), 124-129.
4. Sood, S., & Singh, H. (2020, December 3). An implementation and analysis of deep learning models for the detection of wheat rust disease. 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS). http://dx.doi.org/10.1109/iciss49785.2020.9316123
5. Li, L., Dong, Y., Xiao, Y., Liu, L., Zhao, X., & Huang, W. (2022). Combining disease mechanism and machine learning to predict wheat fusarium head blight. Remote Sensing, 14(12), 2732. https://doi.org/10.3390/rs14122732
6. Cheng, S., Cheng, H., Yang, R., Zhou, J., Li, Z., Shi, B., Lee, M., & Ma, Q. (2023). A high performance wheat disease detection based on position information. Plants, 12(5), 1191. https://doi.org/10.3390/plants12051191
7. Genaev, M. A., Skolotneva, E. S., Gultyaeva, E. I., Orlova, E. A., Bechtold, N. P., & Afonnikov, D. A. (2021). Image-Based wheat fungi diseases identification by deep learning. Plants, 10(8), 1500. https://doi.org/10.3390/plants10081500
8. Shewry, P. R., & Hey, S. J. (2015). The contribution of wheat to human diet and health. Food and Energy Security, 4(3), 178–202. https://doi.org/10.1002/fes3.64

9. Lu, J., Hu, J., Zhao, G., Mei, F., & Zhang, C. (2017). An in-field automatic wheat disease diagnosis system. Computers and Electronics in Agriculture, 142, 369–379. https://doi.org/10.1016/j.compag.2017.09.012
10. Gaikwad, V. P., & Musande, V. (2017, October). Wheat disease detection using image processing. 2017 1st International Conference on Intelligent Systems and Information Management (ICISIM). http://dx.doi.org/10.1109/icisim.2017.8122158
11. Fang, X., Zhen, T., & Li, Z. (2023). Lightweight multiscale CNN model for wheat disease detection. Applied Sciences, 13(9), 5801. https://doi.org/10.3390/app13095801