

**Faculty of Engineering & Technology**

**Electrical & Computer Engineering Department**

**Artificial Intelligence - ENCS3340**

**Project#2 Report**

**Vireon: Automated Bird Species Classification System**

**Name:** Nagham Maali

**ID:** 1212312

**Sec.**: 2

**Instructor**: Dr. Samah Alaydi

**Contents**

[**1. Introduction** 2](#_Toc201466938)

[**2. models Overview** 3](#_Toc201466939)

[**2.1 Naive Bayes Classifier** 3](#_Toc201466940)

[**2.2 Decision Tree Classifier** 3](#_Toc201466941)

**[2.3 Feedforward Neural Network (Multi-Layer Perceptron)](#_Toc201466942)** [4](#_Toc201466942)

[**3. Used Technology** 5](#_Toc201466943)

[**4. Dataset Description and Requirements** 6](#_Toc201466944)

[**5. Implementation** 7](#_Toc201466945)

[**6. Conclusion and Results** 27](#_Toc201466946)

**1. Introduction**

This project, titled Vireon, presents a comparative analysis of three supervised machine learning models — Naive Bayes, Decision Tree, and Feedforward Neural Network — applied to the task of image-based bird species classification. Vireon is more than a name — it draws inspiration from the vireo, a small, sharp-eyed bird known for its keen perception, subtle beauty, and adaptability. This name was chosen to reflect the essence of the project: a lightweight, intelligent system with a clear vision for recognizing patterns in visual data. The simplicity of the vireo mirrors the elegance of the user-friendly interface, while its precision parallels the neural network's impressive classification performance.



*Vireo*

A carefully curated dataset of 811 images across six visually diverse bird species was preprocessed by resizing and flattening each image into a fixed-size feature vector suitable for machine learning input. Each model was then trained and evaluated using key performance metrics such as accuracy, precision, recall, F1-score, and confusion matrices.

The study highlights how different model architectures interpret visual data: The Feedforward Neural Network demonstrated the highest classification accuracy due to its ability to capture complex patterns, while Naive Bayes performed competitively with simplicity and speed. Decision Tree, though less accurate, provided high interpretability and insight into decision logic.

Overall, Vireon showcases the strengths, limitations, and trade-offs of classical and neural models in visual recognition tasks, emphasizing the importance of aligning model choice with the needs of the application — whether that’s interpretability, performance, or computational efficiency.

# **2. models Overview**

## **2.1 Naive Bayes Classifier**

The Naive Bayes Classifier is a simple and fast machine learning model based on probability. It assumes that all features (in this case, pixel values) are independent, which is why it’s called "naive." Despite this assumption, it performs surprisingly well, especially with high-dimensional data like images.

In my project, the model learns the likelihood of each pixel value for every bird class. When a new image is given, it calculates the probability that the image belongs to each class and chooses the most likely one. It is used as a baseline model to compare against more complex methods.

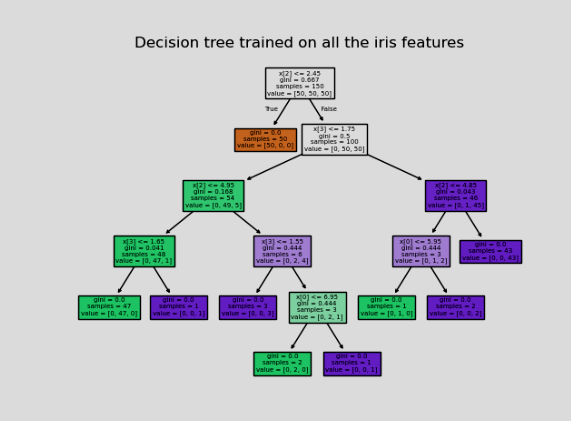
* Strengths: Very fast, easy to implement, works well with large datasets.
* Limitations: Doesn’t capture complex relationships between pixels.

## **2.2 Decision Tree Classifier**

The **Decision Tree Classifier** works like a flowchart. It asks a series of yes/no questions about the image’s pixel values to decide which class the image belongs to. Each decision splits the data into smaller groups, helping the model narrow down the right category.

For example, it might ask: Is the pixel at position X brighter than a certain value? Based on the answer, it moves down a branch of the tree until it reaches a final decision (a bird class).

* **Strengths:** Easy to understand and interpret.
* **Limitations:** Can overfit the training data and may not generalize well to new images.



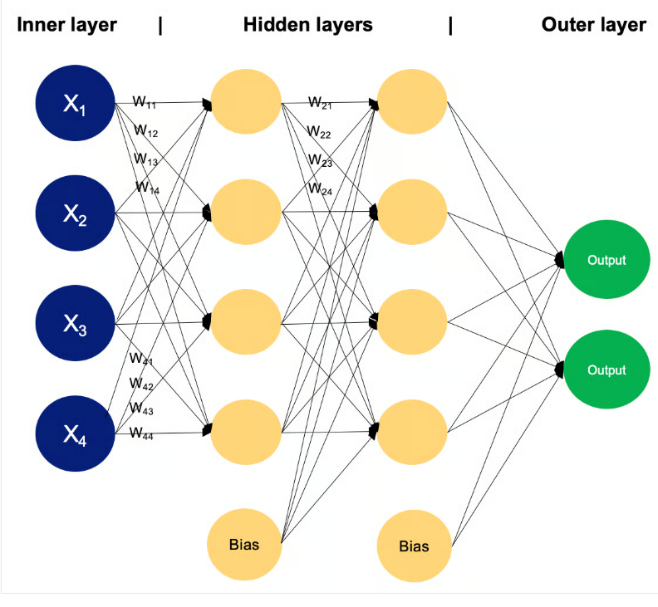
Decision Tree Visualization

## **2.3 Feedforward Neural Network (Multi-Layer Perceptron)**

The Feedforward Neural Network (also called MLPClassifier) is a more advanced model that tries to learn deeper patterns in the data. It is made of layers of connected nodes (called neurons), where each layer processes the input and passes it to the next layer.

This model doesn't make simple decisions like the Decision Tree. Instead, it learns complex patterns by adjusting the weights of connections between neurons during training. This helps it recognize more detailed and hidden features in the images.

* Strengths: Can handle complex patterns and gives higher accuracy.
* Limitations: Requires more training time and resources, harder to interpret.



Example of a MLP with two hidden layers

# **3. Used Technology**

This project was developed using Python, chosen for its simplicity, readability, and strong ecosystem of libraries in both machine learning and GUI development. Python made it easy to combine image processing, model training, and a graphical interface into one cohesive application.

I used several Python libraries to build the project. For machine learning, I relied on Scikit-learn, a well-known library that provides built-in classifiers like Naive Bayes, Decision Tree, and MLPClassifier (a type of neural network). These models are easy to use and provide reliable performance for small to medium-sized datasets. Scikit-learn also gave me useful tools to split my data, measure model accuracy, and visualize results like the decision tree and classification reports.

For image processing, I used the Pillow library (PIL), which allowed me to open, resize, and convert images to the format needed for model input. All the bird images were resized to a fixed size (64×64), converted to RGB, then flattened into 1D vectors using NumPy. NumPy was also essential for managing arrays and performing mathematical operations efficiently.

To build the graphical interface, I used Tkinter, Python’s standard GUI toolkit. It allowed me to create windows, buttons, labels, file pickers, and dropdown menus. The GUI lets users choose an image and select a model to classify the bird in the image — all in a few simple clicks. One of my favorite parts was customizing the design of the interface. I created a dark-mode theme using soft, earthy colors inspired by the Vireo bird, which is where the name “Vireon” comes from. The main color palette includes a deep dark background, soft whites for text, a bold yellow for highlights, and touches of olive green to reflect the Vireo’s natural tones. The GUI also includes popups for classification results and error messages, making the experience smoother for the user. I used StringVar to handle dynamic text, like showing the selected file name or the predicted bird species. Toplevel windows were used for popup messages and scrollable report views, helping keep the interface clean and organized. Concepts like modularity, event-driven programming, and separation of logic and design were important throughout the development process.

Using Python, its libraries, and a styled Tkinter interface, I built a user-friendly image classification app that combines function with creativity.

# **4. Dataset Description and Requirements**

The dataset used in this project is the Bird Speciees Dataset by Rahma Sleam, available on Kaggle : <https://www.kaggle.com/datasets/rahmasleam/bird-speciees-dataset> .

It contains a total of 811 colored images divided into six bird categories:

1. American Goldfinch
2. Barn Owl
3. Carmine Bee-eater
4. Downy Woodpecker
5. Emperor Penguin
6. Flamingo

Each species is stored in a separate folder and comes with unique colors, shapes, and backgrounds. The images differ in size, lighting, and orientation, which made preprocessing an essential part of the project.

To prepare the data for classification, all images were resized to 64×64 pixels to standardize the input. Since each image is in RGB, they were converted into numerical arrays and flattened into 1D vectors with 12,288 features (64×64×3). The class names were automatically taken from the folder names and encoded into numeric labels ranging from 0 to 5. After preprocessing, the dataset was split into two parts: 80% for training (648 images) and 20% for testing (163 images), using stratified sampling to keep the same balance of classes in both sets.

These steps helped prepare the data for machine learning models such as Naive Bayes, Decision Tree, and neural networks. Because the dataset contains clear class boundaries and visual variety, it worked well for testing multiple models and comparing their performance. Its diversity also added an interesting challenge when designing the classification system and evaluating accuracy.

# **5. Implementation**

The foundation of this project begins with setting up the environment using Python and a range of essential libraries. Python was chosen for its rich ecosystem and simplicity, making it easy to combine machine learning, image processing, and GUI development in one place. To start, the project imports various libraries that handle everything from image loading and data processing to training models and displaying results through a graphical interface.

NumPy plays a major role in handling numeric data and manipulating arrays, which is especially useful for image flattening and model input preparation. The Pillow library (PIL) is used to open and convert image files into a format suitable for processing. Since the project involves a visual interface, the Tkinter library provides all the GUI components such as windows, buttons, labels, dropdowns, and canvases. These elements help create an interactive experience for users, allowing them to select images, run models, and view results directly from the app.

For machine learning, the code uses Scikit-learn’s powerful tools. It includes models like Gaussian Naive Bayes, Decision Tree Classifier, and MLPClassifier (a basic neural network), each serving a unique purpose in the classification process. The models are evaluated using metrics like accuracy, confusion matrix, and a detailed classification report — all provided by Scikit-learn’s built-in functions.

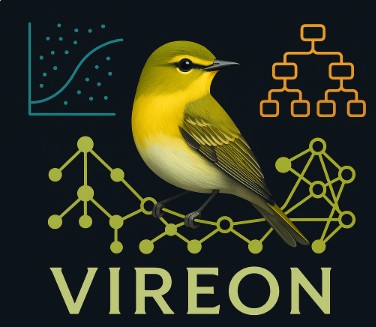
Visualization is handled using Matplotlib, which is used specifically for plotting the structure of the Decision Tree. To present the results in a readable way, Pandas is used to organize the evaluation metrics into clean, formatted tables.

The project also defines several constants to help with organization. These include the dataset directory, the image size for resizing input photos (64×64 pixels), a set of color codes for GUI theming, and a dictionary mapping bird species to example image paths. The COLORS dictionary, for instance, helps keep the interface visually cohesive by using earthy tones inspired by the Vireo bird — a mix of dark background, soft whites, olive green, and a warm yellow for highlights. These color choices tie the design closely to the theme of the app, giving it both a personal and polished feel.

Together, this setup provides a clean, modular starting point that allows for seamless integration between data processing, model training, and user interaction.

The program begins with a splash screen — a temporary, visually appealing window that introduces the user to the app before the main interface loads. This splash screen is created using Tkinter’s Tk() function, and is styled to have no borders or title bar using overrideredirect(True), giving it a clean, custom look. A full-screen image, set using the path defined earlier (COVER\_PATH), is displayed in this window. The image is resized to 900×600 pixels to fit nicely within the splash window, and rendered using Tkinter’s Canvas widget, which allows the image to be positioned precisely from the top-left corner.

Once the splash is fully set up, its dimensions are calculated and the screen is automatically centered using the user’s screen width and height. This centering step ensures that the splash appears directly in the middle of the screen regardless of screen size, giving it a polished appearance. The app then waits for three seconds — just enough time for the user to see the cover image — before automatically moving to the main interface by calling the launch\_intro\_window() function using after(3000, ...). The splash’s mainloop() keeps it running until the transition happens. This short introduction not only makes the app feel more professional but also sets the tone and branding before the user interacts with any models or uploads.



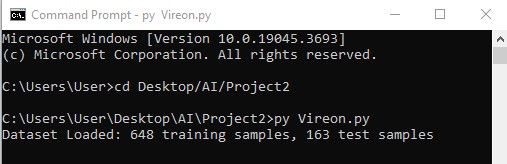
Cover-Page

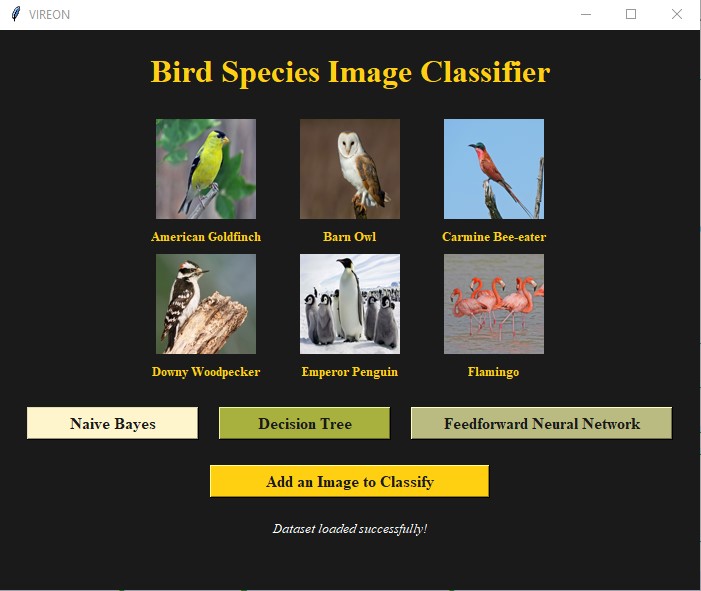
After the splash screen disappears, the main window of the application is launched through the launch\_intro\_window function. This window is where all interaction takes place. It’s built using Tkinter and styled with the dark-mode color palette defined earlier to match the theme of the app. At the top, a large title introduces the project: "Bird Species Image Classifier." Below it, the interface showcases a grid of six sample bird species with their images and names neatly displayed. These images are loaded from predefined paths and shown using labels, providing users with a quick visual preview of the bird categories included in the dataset.

Below the sample birds, there’s a row of buttons — each linked to a different classifier. When a user clicks one of these buttons, it runs the corresponding model (Naive Bayes, Decision Tree, or Feedforward Neural Network) on the dataset and displays the results. Another button titled “Add an Image to Classify” lets users upload any image of their own, which the app will classify based on the selected model.

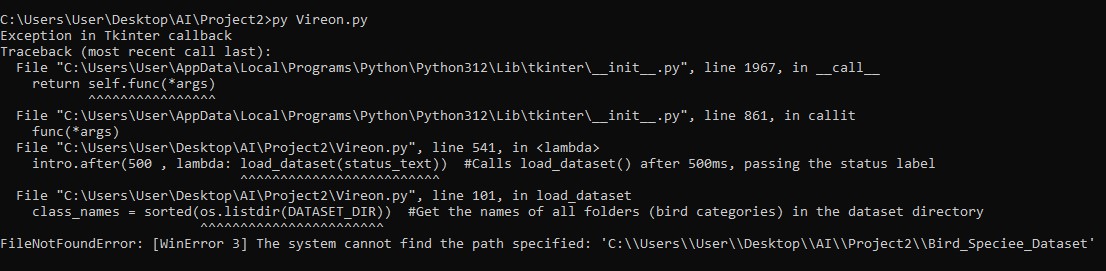
At the bottom of the window, a status label provides live updates, such as when the dataset is loading or when classification is complete. This status is controlled using a special dynamic text variable called StringVar, which makes the interface more interactive and responsive. Shortly after the window loads, the program automatically calls the load\_dataset function to prepare the training and testing data in the background.

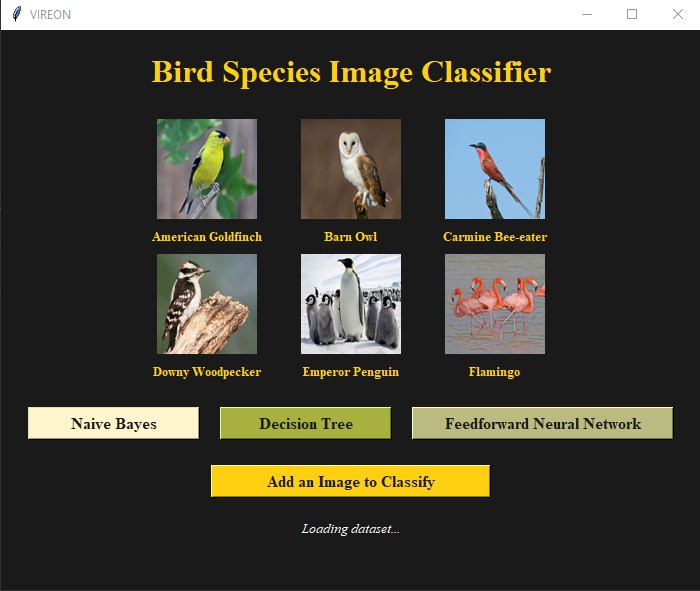
The load\_dataset function is responsible for loading all images from the dataset directory. It reads each folder (which represents a bird species), opens every image inside, resizes it to 64×64 pixels, and flattens it into a 1D vector using NumPy. These flattened images are stored as feature data, while the folder names are used as labels. Any unreadable or corrupted images are skipped silently. Once all images are processed, the function splits the data into 80% training and 20% testing sets using Scikit-learn’s train\_test\_split method, with stratified sampling to ensure that each bird class is equally represented. Finally, it updates the GUI status to inform the user that the dataset is ready for classification.

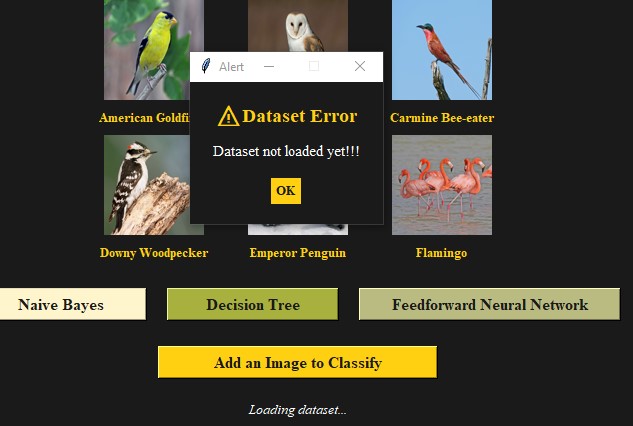




If the dataset doesn’t exist or not loaded maybe because a mistake in the path:







when choosing a model without a dataset

The core logic of running the Naive Bayes model is handled inside the run\_naive\_bayes function. Before anything starts, the function checks if the dataset has been properly loaded by verifying the training data variables. If the data isn’t ready yet, a custom alert box pops up, warning the user that the dataset must be loaded first. This avoids crashes or empty model evaluations.

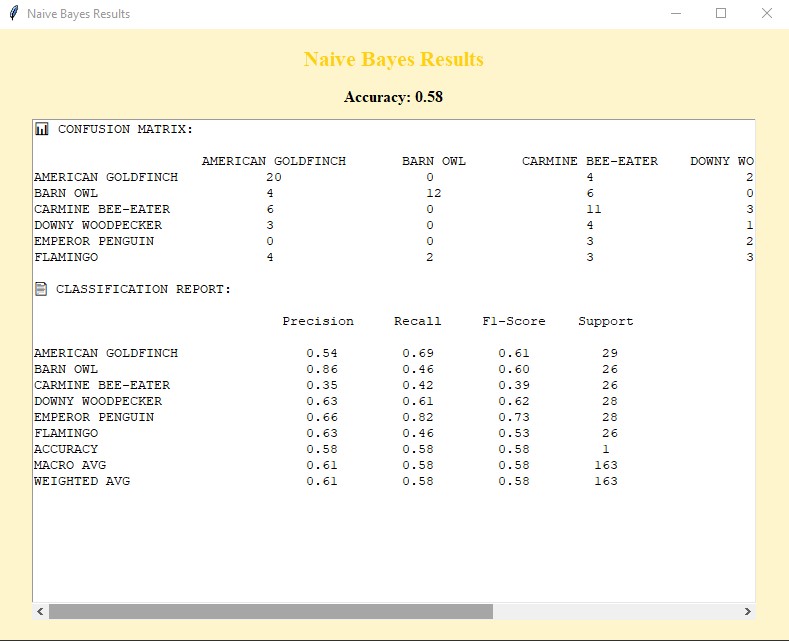
Once everything is ready, a Gaussian Naive Bayes model is created and trained using the preprocessed training data. The model learns the statistical patterns in the pixel values that correspond to different bird classes. After training, it predicts the bird species of the test images, and several evaluation metrics are calculated: accuracy, confusion matrix, and a full classification report with precision, recall, and F1-score for each class. These are essential for comparing model performance and understanding where it makes correct or incorrect predictions.

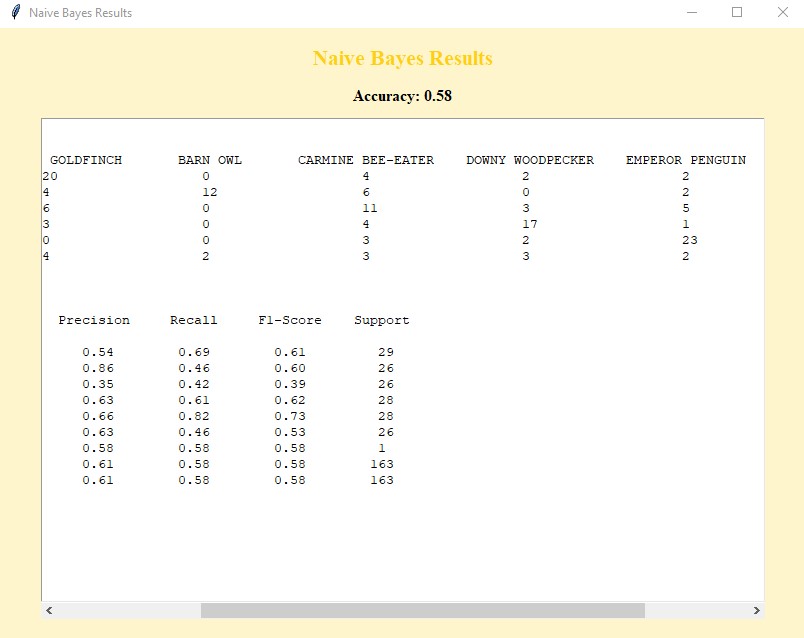
To make the results readable and visually appealing, the confusion matrix is formatted as a table, where each row shows how many images were correctly or incorrectly classified per class. This is followed by a detailed classification report, formatted using Pandas into a table that highlights each metric. The results are then passed into a custom-built message box window using the show\_custom\_messagebox function.

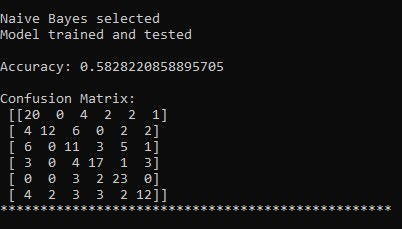
This custom message box is a popup window that displays the model results in a scrollable text area. Styled with colors that match the model’s corresponding button (like pale yellow for Naive Bayes), the popup includes the model’s title, its accuracy score, and a neatly aligned report inside a fixed-width font for better table formatting. This message box is scrollable, making it easy to view large result tables without crowding the interface.

If the user accidentally tries to run a model before loading the dataset, the app also has a dedicated alert popup called show\_alert\_messagebox. This small window displays a clear warning with styled fonts and a single “OK” button to dismiss it. It keeps the user informed without crashing the program, and adds to the overall smoothness of the experience.

This part of the code shows how the app not only trains and evaluates a machine learning model, but also explains the results to the user in a clear, clean, and visually styled way. It mixes logic, user feedback, and GUI elements together, allowing a beginner-friendly interaction with advanced machine learning.







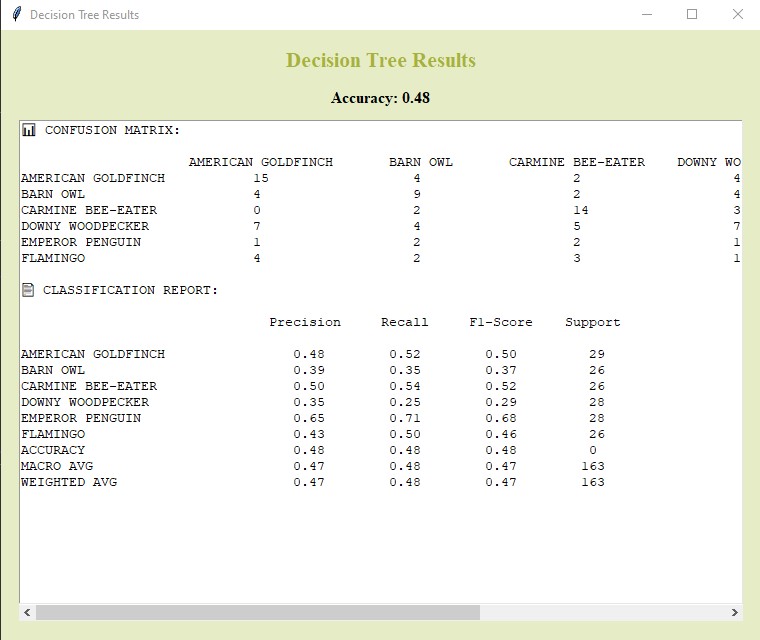
The second model in the project is the Decision Tree Classifier, which is launched using the run\_decision\_tree function. As with the Naive Bayes function, the first step is to ensure the dataset has been loaded. If not, the app immediately shows a clear alert message and stops execution to prevent errors.

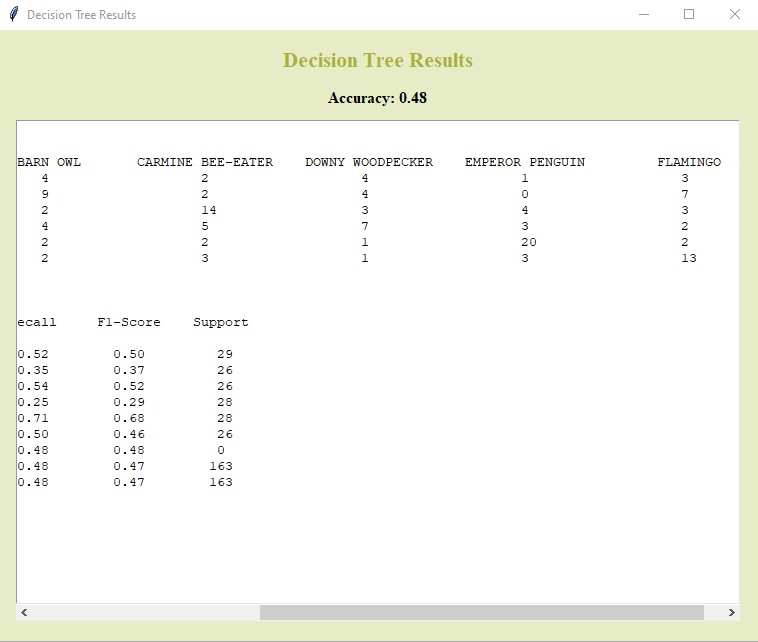
Once confirmed, a new Decision Tree model is created and trained on the bird image data. A fixed random\_state is used to make sure results remain consistent between runs. The model works by building a tree-like structure of decisions based on pixel values — asking questions like “Is pixel X brighter than value Y?” and branching down until a final bird class is reached. After training, the model predicts bird species for the test data, and its accuracy is calculated. A confusion matrix is generated to show how many images were correctly or incorrectly classified per species, and a full classification report is created, showing metrics like precision, recall, F1-score, and support for each class.

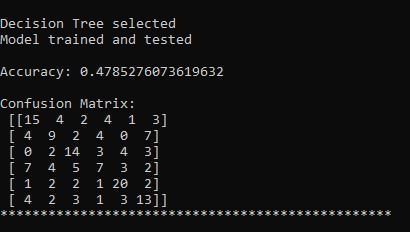
To display these results clearly, the confusion matrix is formatted into a table using the class names as headers and rows. The classification report is also organized into a table using Pandas for better readability. Both of these summaries are displayed in a custom-styled popup using the show\_custom\_messagebox function. For the Decision Tree, a soft green and olive theme is applied to match the model’s color-coded button.

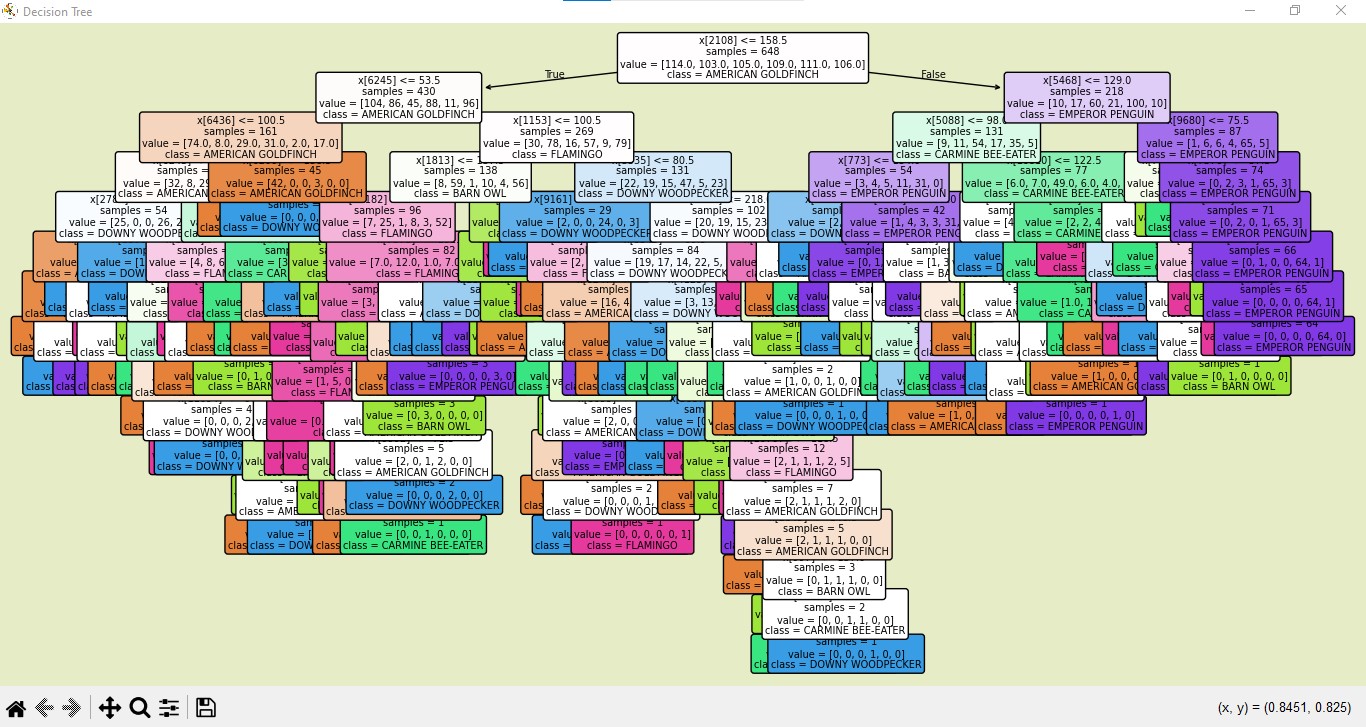
One of the most unique parts of this section is the visualization of the trained Decision Tree model. A new window opens to show a compact diagram of the tree, where each node represents a decision the model makes. This plot is generated using Matplotlib and Scikit-learn’s plot\_tree function, with special settings applied to make the layout more readable. It shows how the model splits the data step-by-step, making it an excellent tool for understanding the model’s logic.

This function not only trains and evaluates a model but also teaches the user how the model thinks — turning abstract decision-making into a visual, interactive experience.









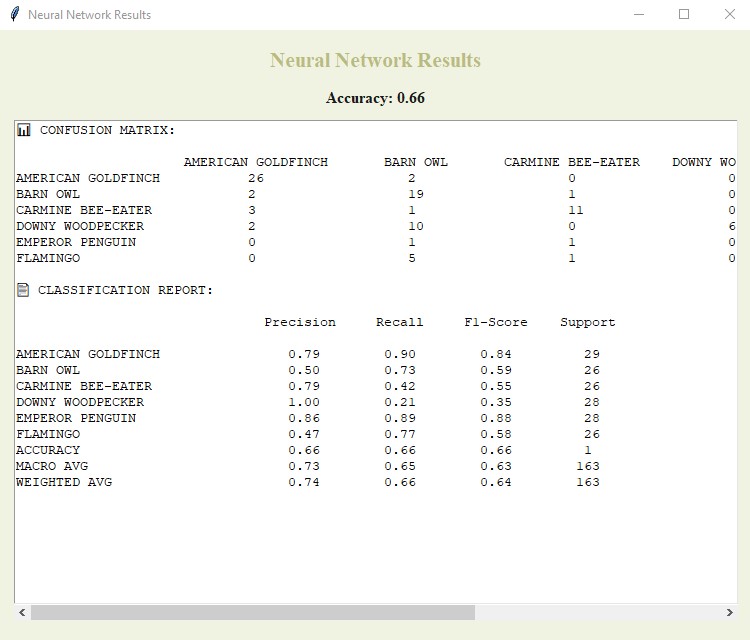
Decision Tree

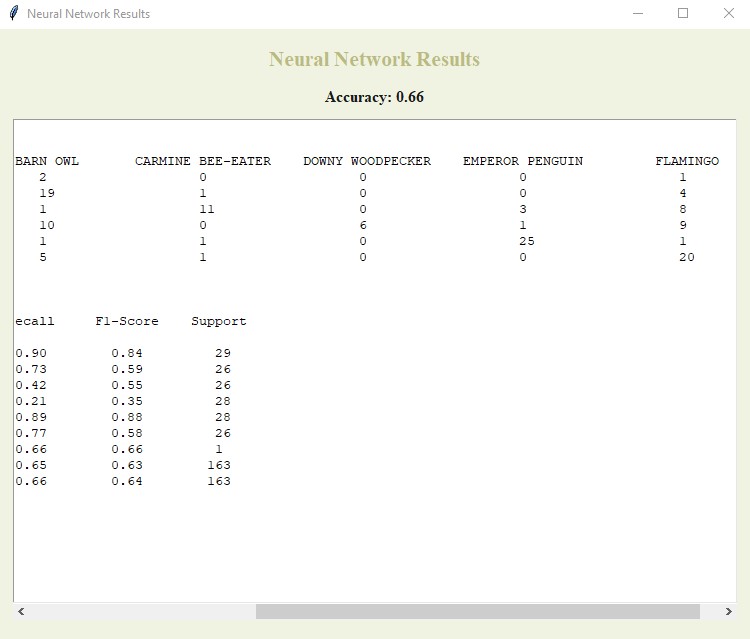
The final model used in this project is the Feedforward Neural Network, implemented using Scikit-learn’s MLPClassifier. This model is more complex than Naive Bayes or Decision Tree because it uses layers of artificial neurons that learn patterns through weighted connections. The run\_neural\_network function begins by checking whether the dataset is loaded, just like the other models. If it’s not ready, a warning message is shown, and the function exits safely.

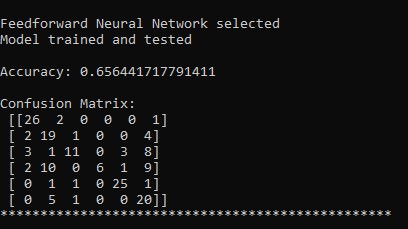
If the data is available, a neural network is created with two hidden layers: the first containing 100 neurons and the second 50. These layers allow the model to learn both simple and complex features in the bird images — especially helpful when trying to distinguish between species with subtle visual differences. The model is trained for up to 500 iterations, giving it enough time to adjust and optimize its internal parameters. Once training is complete, it makes predictions on the test data and generates three key outputs: overall accuracy, a confusion matrix, and a detailed classification report.

As in the other models, the confusion matrix is neatly formatted into a readable table showing where the model got predictions right or wrong for each bird class. The classification report — including precision, recall, F1-score, and support — is also styled using Pandas for clarity. All this information is displayed in a custom message box using the same popup system as the previous models. This time, the interface is styled in soft neutral tones to match the model’s position in the GUI and distinguish it from the others.

This part of the project highlights the power of neural networks in handling high-dimensional visual data. Although the training process takes longer and the internal workings of the model are harder to interpret, the neural network typically achieves the highest accuracy, making it ideal for more complex classification tasks.





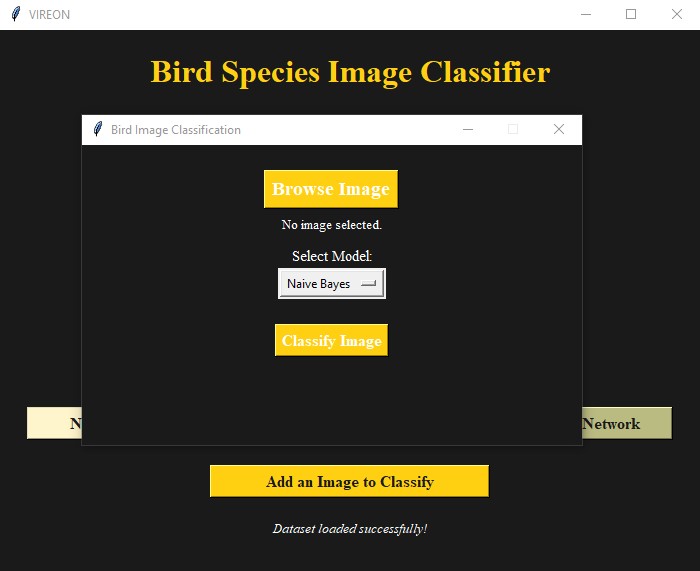


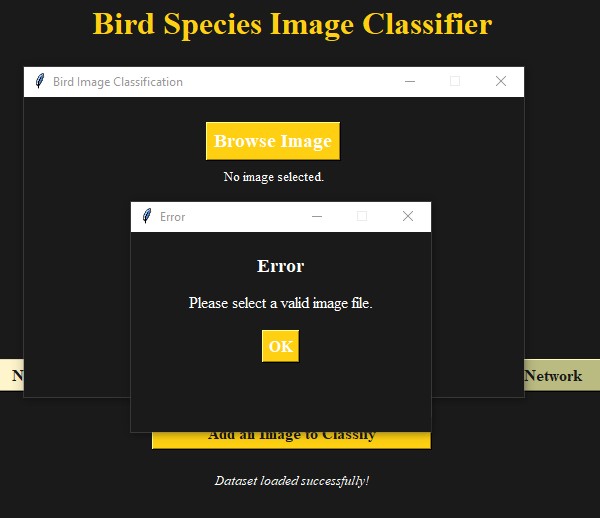
The final feature in the app allows users to upload their own bird image and classify it using one of the trained models. This functionality is handled through the open\_classify\_window function, which opens a new popup window styled to match the rest of the interface. Inside this window, the user can click a “Browse Image” button to select an image file from their computer. The selected file path is then displayed as a label in the window for confirmation. The user is also asked to choose a model — Naive Bayes, Decision Tree, or Neural Network — from a simple dropdown menu.

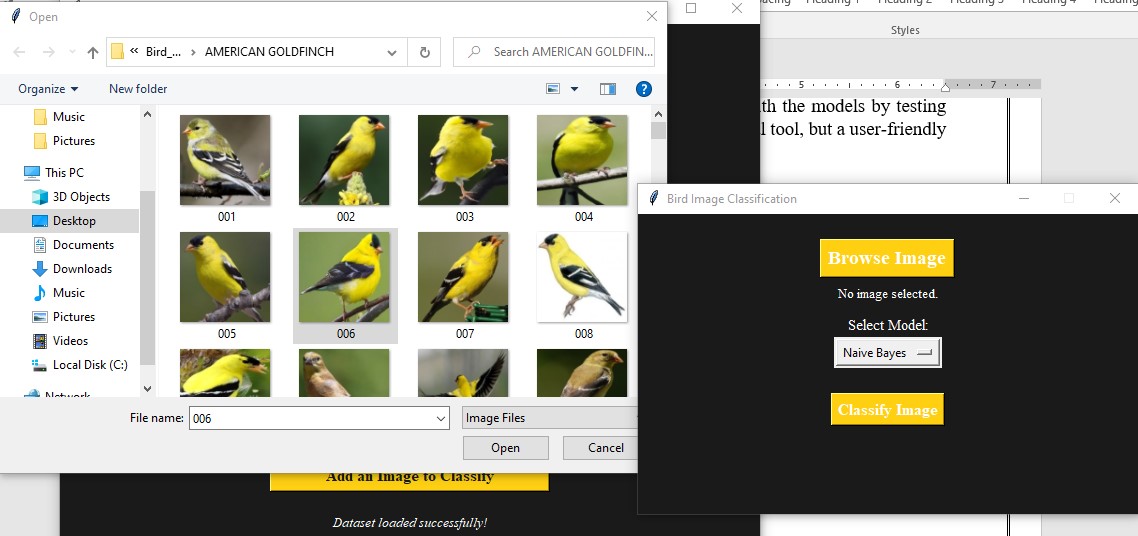
Once both the image and model are selected, clicking the “Classify Image” button starts the classification process. The app first checks that the chosen file exists and is valid. It then resizes the image to 64×64 pixels, converts it to RGB, flattens it into a 1D array, and reshapes it into the format expected by the model. Depending on the selected option, the appropriate machine learning model is trained again on the training set and used to make a prediction. Once the prediction is made, the app identifies which bird species was predicted and opens a neat popup window to display the result to the user.

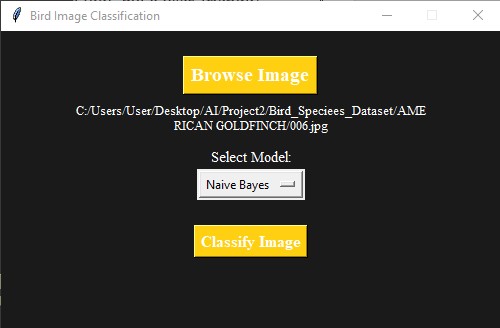
The result window is styled with a bold title and shows the predicted bird’s name clearly in the center, along with an “OK” button to close it. If something goes wrong — like the user selects no image, or the file format is unsupported — another custom popup appears showing the error in a friendly way. These popups are designed using the same dark background and bright highlight colors as the rest of the app to keep the experience consistent.

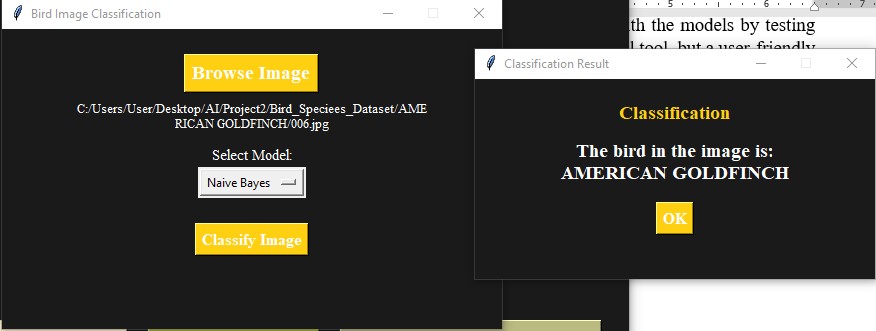
This feature gives users the chance to interact directly with the models by testing them on real images, making the project not just a technical tool, but a user-friendly and engaging application.

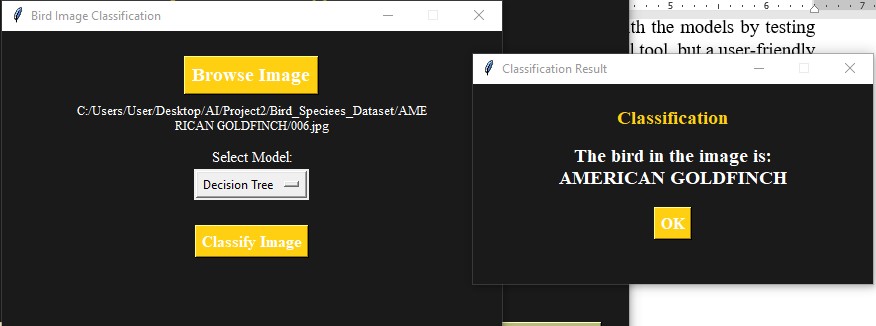


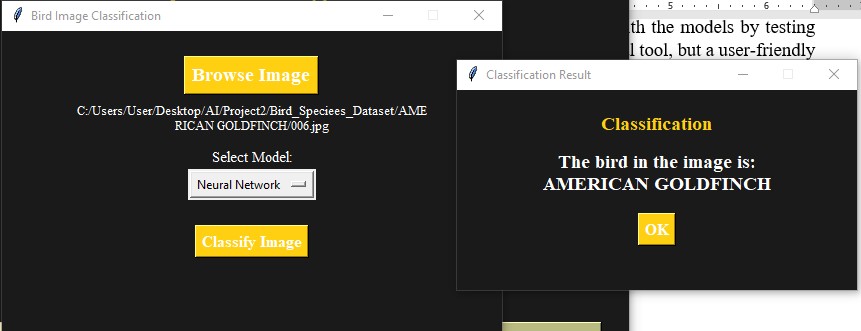




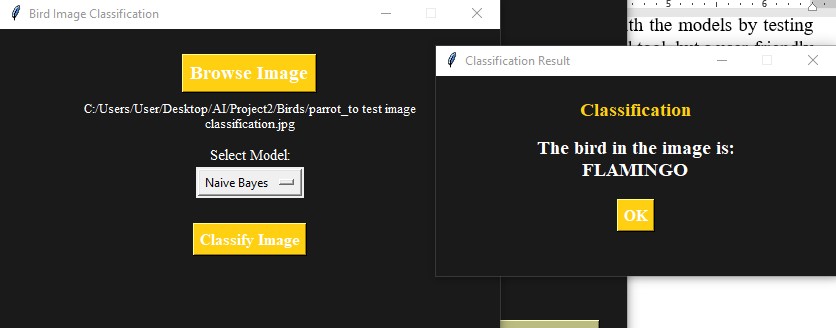


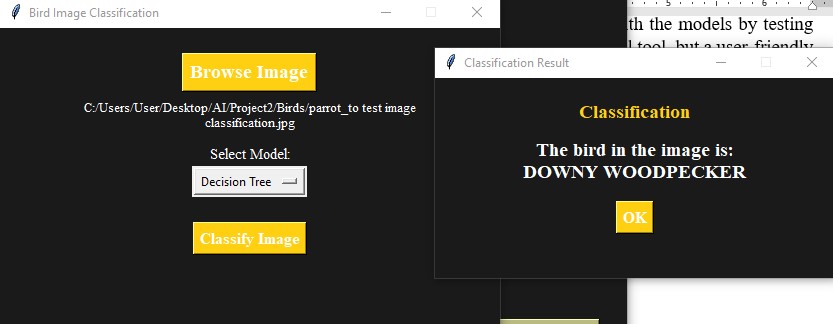


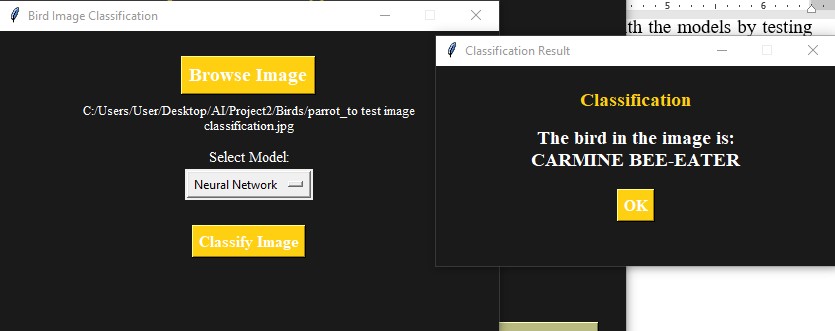




If the user uploads an image of a bird that is not part of the original dataset, such as a parrot:







the models still attempt to classify it based on the visual features they’ve learned. Since they haven’t seen this bird before, they make predictions by comparing it to the most visually similar species in the training data. For example, when a parrot image was tested, the Naive Bayes model predicted it as a Flamingo, the Decision Tree predicted Downy Woodpecker, and the Neural Network chose Carmine Bee-eater. Each model made its decision based on the patterns, colors, and shapes it has learned, showing how these algorithms generalize when faced with unfamiliar input.

# **6. Conclusion and Results**

This project successfully demonstrated how machine learning algorithms can be applied to a real-world image classification task involving bird species. By building a user-friendly app and training three supervised models — Naive Bayes, Decision Tree, and Feedforward Neural Network (MLPClassifier) — I explored both the technical performance and user experience aspects of AI systems.

Each model was trained and tested on a dataset of 811 labeled bird images across six categories. Evaluation was based on accuracy, precision, recall, F1-score, and confusion matrices. The table below summarizes the performance of all three models:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **Best Class** | **Weakest Class** | **Notes** |
| **Naive Bayes** | 58.3% | Emperor Penguin (0.82) | Carmine Bee-eater (0.42) | Fast and simple; struggles with fine details or overlaps |
| **Decision Tree** | 47.9% | Emperor Penguin (0.71) | Downy Woodpecker (0.25) | Easy to interpret; but prone to overfitting, performance is weaker |
| **Neural Network (MLP)** | **65.6%** | American Goldfinch (0.90) | Downy Woodpecker (0.21) | Most accurate overall; balances complexity with better generalization |

From the results, the Neural Network clearly performed the best. It achieved the highest overall accuracy of 65.6%, and also performed well on multiple classes, particularly American Goldfinch, Emperor Penguin, and Flamingo. Despite a lower recall for Downy Woodpecker, it still outperformed other models in most categories, thanks to its ability to detect complex patterns through hidden layers and nonlinear decision boundaries.

The Naive Bayes classifier came second with 58.3% accuracy, showing good performance for Emperor Penguin and Downy Woodpecker, but struggled with species that share visual similarities, like Carmine Bee-eater and Flamingo. Its probabilistic approach is lightweight and fast, making it suitable for smaller datasets or systems where speed matters more than precision.

The Decision Tree model had the lowest accuracy at 47.9%, but its strength lies in interpretability. The decision tree visualization (shown below) clearly displays the entire structure of the trained model, where each node represents a decision made based on a specific pixel’s intensity. The tree splits data recursively by asking yes/no questions like “Is pixel x[2108] ≤ 158.5?”, eventually leading to a leaf node representing a predicted bird class. Although the visual is large and complex, it provides deep insight into how the model thinks and classifies. However, the tree tends to overfit the training data, which explains its lower accuracy on unseen test samples.

To explore how the models, generalize beyond the dataset, a test was done by uploading a photo of a parrot — a bird not included in the training data. Each model made a prediction based on what it thinks is the closest match visually:

1. Naive Bayes classified it as a Flamingo
2. Decision Tree chose Downy Woodpecker
3. Neural Network predicted Carmine Bee-eater

This behavior shows that models attempt to match unknown input to the closest class in terms of learned visual patterns — such as color, shape, or pixel distribution — even when the exact species is absent. It demonstrates how machine learning models generalize, but also the risk of misclassification when encountering completely unfamiliar input.