

**A PROJECT REPORT**  
**on**  
**PREDICTION OF BIRD SPECIES**

*Submitted by*

N. RAMYA	(207W1A4215)
B. PRIYANKA	(207W1A4204)
R. PAVANI	(207W1A4221)
S. SRI VIDYA	(207W1A4224)
K. VENKATESHWARLU	(207W1A4233)

**BACHELOR OF TECHNOLOGY**

**In**

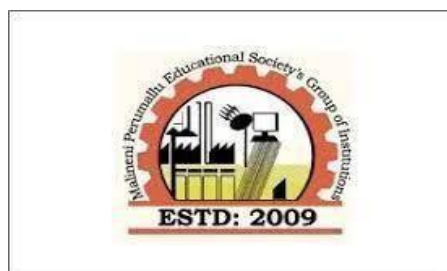
**Computer Science and Engineering**

**[Artificial Intelligence and Machine Learning]**

**Under the guidance of**

**Mr. Dr. SRINIVASA KUMAR, PhD**

**Professor, Department of CSE**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**  
**MALINENI PERUMALLU EDUCATIONAL SOCIETY'S**  
**GROUP OF INSTITUTIONS**

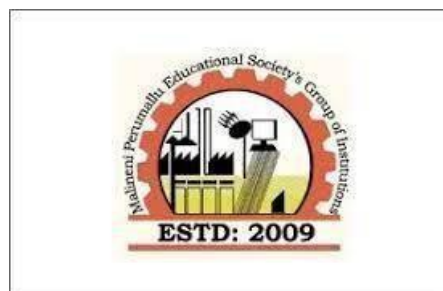
(Approved by AICTE & Affiliated to JNTU KAKINADA)  
Pulladigunta(V), Guntur-522017  
**2020 –2024**

**MALINENI PERUMALLU EDUCATIONAL SOCIETY'S**

**GROUP OF INSTITUTION**

**Pulladigunta(V), Guntur– 522017**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



## **CERTIFICATE**

*This is to certify that the project entitled” **PREDICTION OF BIRD SPECIES**” is the bonafide work carried out by*

**N. RAMYA**

**(207W1A4215)**

**B. PRIYANKA**

**(207W1A4204)**

**R. PAVANI**

**(207W1A4221)**

**S. SRI VIDYA**

**(207W1A4224)**

**K. VENKATESHWARLU**

**(207W1A4233)**

*B.Tech, students of **MPES**, Affiliated to **JNTUK**, Kakinada in partial fulfillment  
of the requirements for the award of the Degree of **BACHELOR OF  
TECHNOLOGY** with the specialization in **COMPUTER SCIENCE AND  
ENGINEERING[Artificial Intelligence and Machine Learning]** during the  
Academic year **2023-2024**.*

**Signature of the Guide**

**Head of the Department**

**Viva-voice Held on**

**Internal Examiner**

**External Examiner**

**MALINENI PERUMALLU EDUCATIONAL SOCIETY'S  
GROUP OF INSTITUTIONS**

**Pulladigunta (V), Guntur – 522017**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**DECLARATION**

We here by declare that the project work entitled “**PREDICTION OF BIRD SPECIES**” is entirely my original work carried out under the guidance of **Dr. SRINIVASA KUMAR , Phd**, Department of Computer Science and Engineering, Malineni Perumallu Educational Society's Group of institutions, Pulladigunta, Guntur, JNTU Kakinada, A.P, India for the award of the degree of **BACHELOR OF TECHNOLOGY** with the specialization in **COMPUTER SCIENCE AND ENGINEERING[Artificial Intelligence and Machine Learning]**. The results carried out in this project report have not been submitted in a part or full for the award of any degree or diploma of this or any other university or institute.

**N. RAMYA**

**(207W1A4215)**

**B. PRIYANKA**

**(207W1A4204)**

**R. PAVANI**

**(207W1A4221)**

**S. SRI VIDYA**

**(207W1A4224)**

**K. VENKATESHWARLU**

**(207W1A4233)**

# ACKNOWLEDGEMENT

All endeavours over a long period can be successful only with the advice and support of many well-wishers. I take this opportunity to express my gratitude and appreciation to all of them.

We wish to express deep sense of gratitude to my beloved and respected guide **Dr.D.Srinivasa Kumar (Principal)** of CSE, Malineni Perumallu Educational Society's Group of Institutions, Guntur, for her valuable guidance, suggestions and constant encouragement and keen interest enriched throughout the course of project work.

We extend sincere thanks to the **HOD**, prof. **Dr.D Ravi Kiran** for his kind cooperation in completing and making this project a success.

We extend sincere thanks to the **Principal**, Prof. **Dr.D.Srinivasa Kumar** for his kind cooperation in completing and making this project a success.

We would like to thank the **Management** for their kind cooperation and for providing infrastructure facilities.

We extend thanks to all the **Teaching staff** of the Department of CSE for their support and encouragement during the course of my project work. I also thank the **Non-Teaching staff** of the CSE department for being helpful in many ways in successful completion of my work.

Finally, we thank all those who helped me directly or indirectly in successful completion of this project work.

<b>N. RAMYA</b>	<b>(207W1A4215)</b>
<b>B. PRIYANKA</b>	<b>(207W1A4204)</b>
<b>R. PAVANI</b>	<b>(207W1A4221)</b>
<b>S. SRI VIDYA</b>	<b>(207W1A4224)</b>
<b>K. VENKATESHWARLU</b>	<b>(207W1A4233)</b>

## **ABSTRACT**

In this study, we present a comprehensive approach for the prediction of bird species using Convolutional Neural Networks (CNNs). The dataset comprises images representing 200 different bird species, with 80% allocated for training and 20% for testing. We used a special kind of artificial intelligence called Convolutional Neural Networks (CNNs) to help our program understand different features in the bird pictures. To train our program, we resize the pictures, convert the image into grayscale, adding the gaussian blur, extracting the features to make sure it could recognize birds accurately. The CNN architecture helped our program learn the patterns and features of each bird species. We trained the program using 80% of images, and then we tested it using the other 20% to see how well it could guess the bird species. The input for the model will be an image of the bird for which we need to find out the species type. For this image pre-processing is applied and then passed to the model for species detection. This kind of technology could be useful for monitoring wildlife and conserving different bird species. This study lays the groundwork for future research in using computers to identify animals in pictures, especially in the field of bird watching.

**Keywords:-** Image classification, Convolutional Neural, Network(CNN), Feature extraction, Pattern recognition, Pre-processed dataset, Black-and-white image conversion, Accuracy assessment, TensorFlow library, Biodiversity monitoring, Conservation efforts and Ecological research.

# INDEX

s.no	Topics	Page numbers
1.	Introduction	8
2.	Literature survey	9-16
3.	System Study 3.1 Feasibility Study 3.2 Types of Feasibility Study 3.2.1 Economical Feasibility 3.2.2 Technical Feasibility 3.2.3 Social Feasibility	17-18
4.	System Analysis 4.1 Existing System 4.1.1 Disadvantages of existing system 4.2 Proposed System 4.2.1 Advantages of proposed system 4.3 Input and Output design 4.3.1 Input Design 4.3.2 Objectives 4.3.3 Output Design	19-22
5.	System Specification 5.1 Hardware requirements 5.2 Software requirements 5.3 Performance considerations	23-26
6.	System Design 6.1 Architectural Design 6.2 Data Flow Design 6.3 Preprocessing Pipeline 6.4 System Interaction 6.5 Integration with External System 6.6 Error Handling and Monitoring 6.7 Performance Optimization 6.8 Security Measures 6.9 Data Flow Diagram 6.10 UML Diagrams 6.11 Goals 6.12 Use Case Diagrams 6.13 Class Diagram 6.14 Sequence Diagram 6.15 Activity Diagram	27-36

7.	Software Environment 7.1 Programming Languages 7.2 Deep Learning Frameworks 7.3 Data Processing and Analysis 7.4 Image Processing 7.5 Model Architecture and Training 7.6 Deployment and Integration 7.7 Feedback Loop and Model Updates	37-52
8.	Sample Code	53-58
9.	System Test 9.1 Types of Tests 9.1.1 Unit Testing 9.1.2 Integration Testing 9.1.3 Function Test 9.1.4 System Test 9.1.5 White Box Testing 9.1.6 Black Box Testing 9.1.7 Unit Testing 9.2 Test Strategy and Approach 9.2.1 Integration Testing	59-62
10	Sample Test Cases	63-65
11	Implementation	66-67
12	Conclusion 12.1 Future Enhancement 12.2 Bibliography	68-69

## **1. INTRODUCTION**

The bird species prediction project is founded on a rich dataset comprising a wide range of bird species and their associated characteristics. This dataset likely includes features such as physical attributes, habitat preferences, geographical distribution, and behavioral patterns.

To develop the predictive model, advanced machine learning algorithms such as convolutional neural networks (CNNs) will be employed. These algorithms will be trained on the dataset having 525 classes of bird species to learn the intricate patterns and relationships between different bird species and their attributes.

Furthermore, the project may involve preprocessing steps such as data cleaning, feature engineering, and dimensionality reduction to optimize the dataset for model training. Cross-validation techniques might also be utilized to ensure the robustness and generalization capability of the predictive model.

Once the model is trained and validated, it can be deployed as a user-friendly application or integrated into existing platforms. Users, including ornithologists, bird watchers, and conservationists, can then utilize the model to quickly and accurately identify bird species based on input data such as images, audio recordings, or field observations.

Continuous monitoring and updates to the model may be necessary to incorporate new data and improve its performance over time. Additionally, collaborations with domain experts and stakeholders in ornithology and conservation will be crucial to validate the model's accuracy and relevance in real-world scenarios.

Overall, the bird species prediction project represents an interdisciplinary effort that combines the fields of data science and ornithology to enhance our understanding of avian biodiversity and contribute to conservation efforts worldwide.



## **2. LITERATURE SURVEY**

**1. Title:** Image Classification Using Deep Learning and Neural Networks

**Journal:** International Journal of Engineering Research & Technology (IJERT),2003

### **Methodology:**

- The paper uses deep learning, deep neural networks, image classification, and TensorFlow to identify and classify bird species from images<sup>1</sup>.
- The paper employs a large dataset of 400 bird species, with 58388 training images, 2000 test images, and 2000 validation images<sup>2</sup>.
- The paper applies convolutional neural networks (CNNs) to extract features from the images and reduce their dimensionality.
- The paper compares the performance of different CNN architectures, such as VGG-16, MobileNet, and ResNet-50, on the task of bird species classification.
- The paper evaluates the accuracy, precision, recall, and F1-score of the models, as well as the confusion matrix and the classification report.

### **Advantages:**

- **Powerful Models:** The paper uses special computer models (neural networks) that are really good at understanding images. These models can learn complex features from the pictures, just like how we learn to recognize faces or objects.
- **Lots of Examples:** By using a large dataset, the computer learns from many different bird species<sup>1</sup>. It's like learning about birds from a diverse group of experts.

### **Disadvantages:**

- **Computational Complexity:** Training these models takes a lot of computer power and time. It's like solving a really hard puzzle—it requires a powerful computer.
- **Data Bias:** If the dataset mainly has certain types of birds, the computer might struggle with less common species. It's like studying only popular birds and missing out on the rare ones.

## **2. Title:** Automatic Bird Species Identification for Large Number of Species

**Journal:** IEEE Explore journal,2011

### **Methodology:**

- The input is a sequence of video frames of a flying bird taken by an unknown camera.
- The output is a ranked list of candidate bird species based on their likelihood.
- The method consists of four steps:
  - Motion segmentation: Extract the bird boundary from each frame using optical flow, background motion model, Mahalanobis distance, and active contour algorithm.
  - Salient extremities recognition: Find the inter-wing tip distance (IWTD) from the bird boundary by searching for the maximum distance across frames in a wingbeat period and along the wing spreading direction (WSD).
    - Periodicity analysis: Apply Fast Fourier Transformation (FFT) to the IWTD series to obtain the wingbeat frequency (WF) and its error bound. Use a kinematic model of the bird wing and prove that the WF is invariant to camera parameters and motion trajectory<sup>3</sup>.
  - Species prediction: Use a likelihood ratio-based metric to compare the extracted WF with the prior knowledge of WF distributions for different bird species. Return a short ranked list of candidate species.

### **Advantages:**

- It helps quickly and effectively monitor bird populations without disturbing them.
- By using recorded bird songs, it avoids the need for direct interaction with the birds, minimizing disturbance.

### **Disadvantages:**

- Accuracy depends on the quality of recorded bird songs, and noise in the environment can make identification challenging.
- The identification model may not work well for bird species not included in its training set, limiting its applicability.

### **3. Title:** Bird Species Identification Using Deep Learning

**Journal:** International Journal of Engineering Research & Technology (IJERT),2019.

#### **Methodology:**

- The paper uses a deep convolutional neural network (DCNN) to classify bird species from images.
- The paper uses the Caltech-UCSD Birds 200 (CUB-200-2011) dataset, which contains 11,788 images of 200 bird categories, with annotations such as part locations, binary attributes, and bounding boxes<sup>2</sup>.
- The paper converts the bird images into grayscale format and generates an autograph, which is a network of processing nodes that calculates a score sheet for each node and predicts the bird species based on the score sheet analysis<sup>3</sup>.
- The paper uses the GoogLeNet framework, which is a DCNN architecture with 22 layers and multiple inception modules, to train and test the model on the bird images.
- The paper uses Tensorflow, which is a software library for deep learning, to implement the DCNN model and perform parallel processing using GPU technology.

#### **Advantages:**

- It presents a novel method for bird species classification based on deep convolutional neural networks (DCNN) and autograph generation.
- It achieves high accuracy (90.93%) on the Caltech-UCSD Birds 200 (CUB-200-2011) dataset, which is a challenging benchmark for fine-grained recognition.

#### **Disadvantages:**

- It does not compare its results with other state-of-the-art methods for bird species identification, such as pose-normalized models or attribute-based models.
- It does not provide any analysis or discussion on the limitations or challenges of its approach, such as the computational cost, the data augmentation techniques, or the generalization ability.

**4. Title:** Prediction of species richness of breeding birds by landscape-level factors of urban woods in Osaka Prefecture, Japan

**Journal:** IEEE Explore journal, 2011

**Methodology:**

- The authors used bird records from various sources to obtain the species richness of forest birds in 28 urban parks and three mountains in Osaka Prefecture, Japan.
- The authors measured the environmental variables of the urban parks, such as woodland area, elongation, distance to mountain, distance to nearest woods, and percentage of woodland and field cover within 25 km<sup>2</sup> outside the parks.
- The authors compared the fitness of three models to describe the species-area relationship: the power function, the exponential function, and the logistic function.
- The authors performed multiple regression analysis and principal component regression analysis to predict the species richness by the environmental variables and selected the best models based on the adjusted R<sup>2</sup> and Akaike's information criterion (AIC).
- The authors conducted canonical correspondence analysis (CCA) to examine the relationships between species occurrence and environmental variables and identified the nested subset pattern of species composition.

**Advantages:**

- **Learned a Lot:** The study looked at what makes different birds live in city woods. It thought about things like how big the woods are, where they are, and what kinds of homes birds like. It also compared different ways of guessing how many bird types there are.
- **Good Ideas for City Planning:** The research gives good ideas for making cities better for nature. It says it's important to have bigger woods, different kinds of homes for birds, and to connect city woods to other areas where birds come from.

**Disadvantages:**

- **Not Enough Info:** The study only looked at 28 parks in one part of Japan. So, what it found might not be true for other places. It also didn't think about how things change over time or how people and other animals can bother birds.
- **Old Info:** The study is from 1999, so it might not show what's happening now with birds in cities. Some of the ideas and ways they checked things might be old and not used anymore.

## **5. Title:** Hierarchical Classification of Bird Species Using Their Audio-Recorded Songs

**Journal:** IEEE Explore journal,2013

### **Methodology:**

- The authors use a hierarchical classification approach to the bird species identification problem from audio recordings<sup>12</sup>. They compare three types of approaches: flat, local model, and global model.
- They use the MARSYAS framework to extract acoustic features from the audio signals, such as spectral centroid, rolloff, flux, zero crossings, and MFCC.
- They use the Global-Model Naive Bayes (GMNB) algorithm as the global-model classifier, which is an extension of the classic Naive Bayes algorithm that takes into account the hierarchical class structure of the bird species taxonomy.
- They use the flat Naive Bayes and the Local Classifier Per Parent Node (LCPN) approach as the baselines for comparison. The LCPN approach trains a flat classifier for each non-leaf node in the hierarchy and uses a top-down strategy for testing.
- They use the hierarchical precision, recall, and F-measure as the evaluation metrics, which are adapted from the standard precision, recall, and F-measure to account for the hierarchical nature of the problem.
- They conduct experiments on a dataset of 1320 bird song records from 48 bird species, which are organized in a hierarchical tree according to the scientific classification of birds.

### **Advantages:**

- It addresses a challenging and practical problem of recognizing bird species from their vocalizations, which can have applications in bioacoustics, ecology, conservation, and education.
- It explores a novel and effective approach of using a global-model hierarchical classifier, the GMNB algorithm, which can leverage the taxonomic structure of the bird classes and achieve higher accuracy than flat or local-model classifiers.

### **Disadvantages:**

- It relies on a limited and imbalanced dataset of 48 bird species, which may not cover the diversity and variability of bird vocalizations in the real world. It also

does not consider the effects of noise, interference, or environmental factors on the audio signals.

- It uses a simple and fixed feature extraction method, based on MARSYAS, which may not capture the rich and dynamic information contained in the bird sounds. It also does not compare its features with other state-of-the-art methods, such as deep learning or spectrogram-based features.

## **6. Title:** Bird-SDPS: A Migratory Birds' Spatial Distribution Prediction System

**Journal:** IEEE Explore journal,2013

### **Methodology:**

- The authors introduce Bird-SDPS, a system for predicting the spatial distribution of migratory birds based on GPS tracking data and remote sensing data<sup>12</sup>.
- The system consists of four main components: GPS records manager, remote sensing data manager, multi-model manager, and visualization controller<sup>34</sup>.
- The GPS records manager preprocesses and reformats the GPS data into spatial grids, which represent the frequency of birds' occurrence in a geographic area.
- The remote sensing data manager extracts environmental variables from remote sensing files, such as temperature, precipitation, land cover type, etc., and stores them in a NoSQL database (HBase) for efficient retrieval<sup>5</sup>.
- The multi-model manager integrates five different species distribution models: Maxent, SVM, GLM, RF, and DT, and provides an expansion interface for new models. It also evaluates the models using AUC and other metrics.
- The visualization controller uses Web-GIS to display the GPS data, the environmental data, and the prediction results on a map, and provides interactive functions for users.

### **Advantages:**

- **Helps Birds:** The system uses GPS and remote sensing to track birds' travels. This helps us understand where they go, where they like to live, and what they need. This info is super useful for studying nature, planning to protect birds, and preventing diseases they might get.
- **Compares Different Ideas:** The system is like a smart tool that can use many ways to understand bird movements. It can compare different ways of figuring things out, helping us choose the best methods.

### **Disadvantage:**

- **Needs Big and Good Data:** The system relies on lots of info about bird travels and the environment. Handling and storing this big and complicated data is a challenge. If the data isn't good, it can make the system less accurate.
- **Might Miss Some Things:** The system uses models that guess where birds might be based on the environment. But it might not understand all the reasons or interactions that affect birds. It also might not consider how things change over time or in different places.

### **7. Title: SIMULTANEOUS SEGMENTATION AND CLASSIFICATION OF BIRD SONG USING CNN**

**Journal:** IEEE Explore journal,2017

### **Methodology:**

Researchers used a smart computer program to find and identify bird sounds in audio recordings. The program, like a special brain (CNN), could do both jobs at once. It looked at graphs of sound intensity over time (spectrograms), labeled pixels as background or bird species, and learned to do this by looking at many examples.

The program predicted labels for each pixel and identified bird sounds by looking at connected groups of labeled pixels. It combined multiple bird species if they were in the same recording.

### **Advantages:**

- 1. Innovation:** This method is new and combines finding bird sounds and identifying species in one step, unlike traditional methods.
- 2. Better Results:** It performs better than other methods, giving more accurate results.

### **Disadvantages:**

- 1. Needs a lot of computing power:** Training the computer model is computationally expensive and requires a lot of labeled examples.
- 2. Limited use:** It's good for finding bird sounds in certain recordings but might not work well for other tasks or environments. Success depends on having good examples to learn from.

**8. Title:** Automatic Bird Species Detection Using Periodicity of Salient Extremities

**Journal:** IEEE Explore journal, 2013

**Methodology:**

- The authors develop a kinematic model of the bird wing using three revolute joints and derive the inter-wing tip distance (IWTD) in 3D space and in the image coordinate system.
- The authors propose a method to extract the IWTD series from video frames by performing motion segmentation, finding the maximum IWTD across frames in a wingbeat period, and recognizing IWTD series for the entire period using the wing spreading direction (WSD).
- The authors show that the IWTD series is a periodic function that reflects the wingbeat frequency (WF) of the bird, which is invariant to camera parameters and a reliable feature for bird species filtering.
- The authors conduct a frequency analysis on the IWTD series by applying Fast Fourier Transformation (FFT) and a high pass filter to obtain the WF and its error bound.
- The authors propose a species prediction metric using likelihood ratios based on the extracted WF and the prior knowledge of WF distributions for different bird species.

**Advantages:**

- It can provide a non-invasive and efficient method to identify bird species from a distance, without requiring physical capture, tagging, or audio recording.
- It can exploit a reliable and invariant feature of bird flight, the wingbeat frequency, which is related to the bird's morphology, physiology, and ecology.

**Disadvantages:**

- It requires high-quality video data with sufficient resolution, frame rate, and contrast to capture the wingbeat motion and extract the inter-wing tip distance.
- It may not be able to distinguish between closely related or similar-sized bird species that have overlapping wingbeat frequency ranges.



### **3. SYSTEM STUDY**

#### **3.1 Feasibility Study:**

The feasibility of the project is analyzed in this phase and a business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

#### **3.2 Three key considerations involved in the feasibility analysis are,**

- ECONOMICAL FEASIBILITY
- TECHNICAL FEASIBILITY
- SOCIAL FEASIBILITY

##### **3.2.1 Economical Feasibility:**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

##### **3.2.2 Technical Feasibility:**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

### **3.2.3 Social Feasibility:**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

## **4.SYSTEM ANALYSIS**

### **4.1 Existing System:**

The existing system for bird species prediction may rely on traditional computer vision techniques or basic machine learning algorithms. These systems often lack the sophistication and accuracy provided by deep learning methods like Convolutional Neural Networks (CNNs). They might utilize handcrafted features or simple classifiers, resulting in limited performance, especially on fine-grained classification tasks with a large number of classes.

#### **4.1.1 Disadvantages of Existing System**

- **Limited Accuracy:** Traditional methods may struggle to accurately classify bird species, especially when dealing with a large number of fine-grained classes.
- **Manual Feature Engineering:** Handcrafting features requires domain expertise and may not capture all relevant information present in the images.
- **Scalability Issues:** Traditional methods may not scale well to large datasets or complex image data.

**Algorithm:** Support Vector Machines (SVM), k-Nearest Neighbors (k-NN), or decision trees for classification. Feature extraction techniques like Histogram of Oriented Gradients (HOG) or Scale-Invariant Feature Transform (SIFT) might also be used to represent images.

**4.2 Proposed System:** The proposed system aims to address the limitations of the existing system by leveraging deep learning techniques, specifically CNNs, for bird species prediction. CNNs are capable of automatically learning hierarchical features from raw pixel data, leading to improved performance on complex classification tasks like bird species identification.

#### **4.2.1 Advantages of Proposed System:**

- **Improved Accuracy:** CNNs can learn discriminative features directly from data, resulting in higher accuracy compared to handcrafted feature-based approaches.

- **Scalability:** Deep learning models can scale effectively to large datasets and complex image data, making them suitable for tasks with a large number of classes.
- **End-to-End Learning:** CNNs enable end-to-end learning, eliminating the need for manual feature engineering and simplifying the model development process.

**Algorithm:** The proposed system will utilize a CNN architecture tailored for bird species prediction. This architecture will consist of convolutional layers, pooling layers, and fully connected layers, trained using backpropagation with techniques like stochastic gradient descent.

### 4.3 Input and Output Design:

#### 4.3.1 Input Design:

The input to the system consists of bird images in digital format. These images are preprocessed to ensure uniform size and format before being fed into the CNN model for prediction.

Input design refers to how we prepare and present data (in this case, bird images) to the system for processing. In the context of our bird species prediction system, the input design involves several key considerations:

- **Image Format:** Ensure that all input images are in a consistent format (e.g., JPEG, PNG) to maintain uniformity and compatibility with the system.
- **Image Size and Resolution:** Resize and standardize the dimensions of images to a predetermined size suitable for processing by the CNN model. This ensures that all images have the same dimensions, facilitating efficient processing and consistent performance.
- **Preprocessing:** Apply necessary preprocessing techniques to enhance the quality and suitability of input images for analysis. This may include operations such as noise reduction, contrast adjustment, and normalization of pixel values.
- **Data Augmentation:** Optionally, augment the dataset by applying transformations such as rotation, flipping, and cropping to increase the diversity of training examples and improve the model's robustness to variations in image appearance.

- **Data Splitting:** Divide the dataset into training, validation, and testing sets to facilitate model training, tuning, and evaluation. Ensure that the distribution of bird species is balanced across the different sets to prevent biases.
- **Data Loading:** Implement efficient mechanisms for loading batches of images into memory during training and inference to minimize computational overhead and optimize resource utilization.

#### **4.3.2 Objectives:**

- Develop a CNN-based system capable of accurately predicting bird species from images.
- Train the model on a dataset containing a diverse range of bird species to ensure robustness and generalization.
- Evaluate the performance of the model using appropriate metrics and validation techniques.
- Deploy the trained model in a real-world application or system for bird species identification.

#### **4.3.3 Output Design:**

The output of the system is the predicted bird species label for a given input image. This output can be presented to the user along with confidence scores or probability estimates for each predicted class, providing insights into the model's certainty in its predictions. Output design involves determining how the system presents results or responses to users or downstream processes. In our bird species prediction system, the output design encompasses the following aspects:

❖ **Predicted Bird Species:** The primary output of the system is the predicted label corresponding to the bird species identified in the input image. This label indicates the name of the bird species predicted by the model.

❖ **Confidence Score:** Optionally, the system may provide a confidence score or probability estimate associated with each predicted label. This score indicates the model's certainty or confidence in its prediction, helping users assess the reliability of the output.

❖ **Visualization:** Provide visual feedback to users by displaying the input image along with the predicted bird species label and associated confidence score, if applicable. Visualization enhances the interpretability of results and facilitates user understanding.

❖ **Error Handling:** Implement mechanisms to handle and communicate errors or uncertainties in predictions effectively. For example, if the model is uncertain about a prediction, the system may indicate low confidence or suggest additional steps for verification.

❖ **Integration:** Ensure seamless integration of the output with downstream processes or applications where the predicted bird species information may be utilized. This may involve standardizing output formats or providing APIs for easy consumption by other systems.

❖ **Feedback Mechanism:** Establish a feedback loop to collect user feedback and improve the system's performance over time. Users can provide input on the accuracy and relevance of predictions, enabling iterative refinement of the model and enhancing user satisfaction.

## **5. SYSTEM SPECIFICATION**

System specifications outline the technical details and requirements of a computer system or software application. In the context of our bird species prediction system, the specifications can be categorized into hardware requirements, software dependencies, and performance considerations. Here's an overview:

### **5.1 Hardware Requirements:**

#### **❖ Processor (CPU):**

- A multi-core processor (e.g., Intel Core i5 or higher) is recommended for efficient training and inference with deep learning models like CNNs.
- High-performance CPUs can accelerate computations and reduce training times.

#### **❖ Graphics Processing Unit (GPU):**

- A dedicated GPU (e.g., NVIDIA GeForce GTX or RTX series) with CUDA support is highly recommended for accelerated training of deep learning models.
- GPUs with a large number of CUDA cores and ample VRAM (Video RAM) facilitate faster model training and inference.

#### **❖ Memory (RAM):**

- At least 16 GB of RAM is recommended for handling large datasets and training deep learning models efficiently.
- Higher RAM capacity can improve system performance, especially when working with memory-intensive tasks like image processing and model training.

### ❖ **Storage:**

- Sufficient storage space (e.g., SSD or HDD) is required to store the dataset, model checkpoints, and other related files.
- SSDs offer faster read/write speeds, which can reduce data loading times and improve overall system responsiveness.

## **5.2 Software Dependencies:**

### ❖ **Deep Learning Framework:**

- Install a deep learning framework such as TensorFlow, PyTorch, or Keras to implement and train the CNN model.
- These frameworks provide high-level APIs and optimized operations for building and training neural networks.

### ❖ **CUDA Toolkit:**

- If using NVIDIA GPUs, install the CUDA Toolkit to enable GPU acceleration for deep learning computations.
- CUDA provides libraries and tools for parallel computing on NVIDIA GPUs, enhancing the performance of deep learning workflows.

### ❖ **Python Environment:**

- Set up a Python environment with the necessary packages and libraries for data preprocessing, model training, and evaluation.



➤ Commonly used libraries include NumPy, pandas, Matplotlib, and scikit-learn for data manipulation and analysis.

❖ **Image Processing Libraries:**

➤ Utilize image processing libraries such as OpenCV or Pillow for tasks like image loading, resizing, and augmentation.

➤ These libraries offer efficient algorithms and functions for manipulating and preparing image data for training.

### **5.3 Performance Considerations:**

❖ **Parallelism:**

➤ Leverage parallel processing capabilities of GPUs to accelerate model training and inference.

➤ Optimize the utilization of CPU cores for data preprocessing tasks to improve overall system efficiency.

❖ **Memory Management:**

➤ Implement memory-efficient data loading and preprocessing techniques to minimize memory usage and avoid out-of-memory errors.

➤ Utilize batch processing to optimize GPU memory utilization during model training.

❖ **Optimization Techniques:**

➤ Apply optimization techniques such as gradient clipping, weight regularization, and batch normalization to improve model convergence and generalization.

➤ Experiment with different learning rates, optimizers, and learning rate schedules to fine-tune model performance.

**❖ Monitoring and Tuning:**

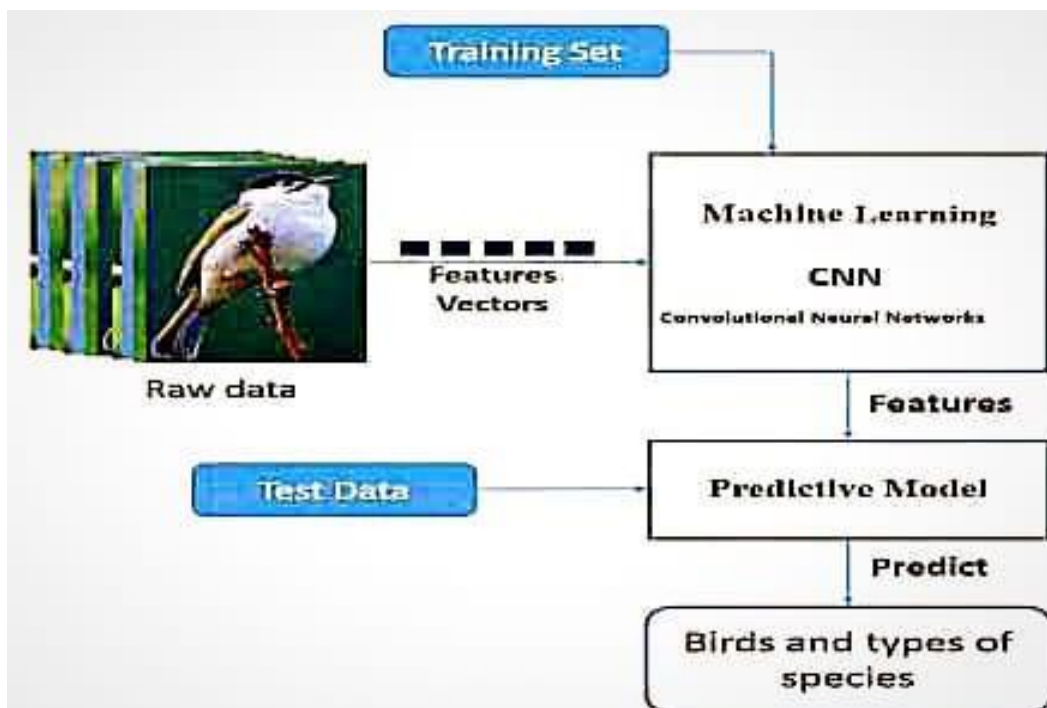
➤ Monitor system resource usage (CPU, GPU, memory) during training to identify bottlenecks and optimize performance.

➤ Perform hyperparameter tuning and model optimization experiments to achieve the best possible accuracy and efficiency.

## 6. SYSTEM DESIGN

System design is the process of defining the architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements. In the context of our bird species prediction system, system design involves structuring and organizing the various elements of the software to effectively implement the desired functionalities. Here's an overview of the key aspects of system design:

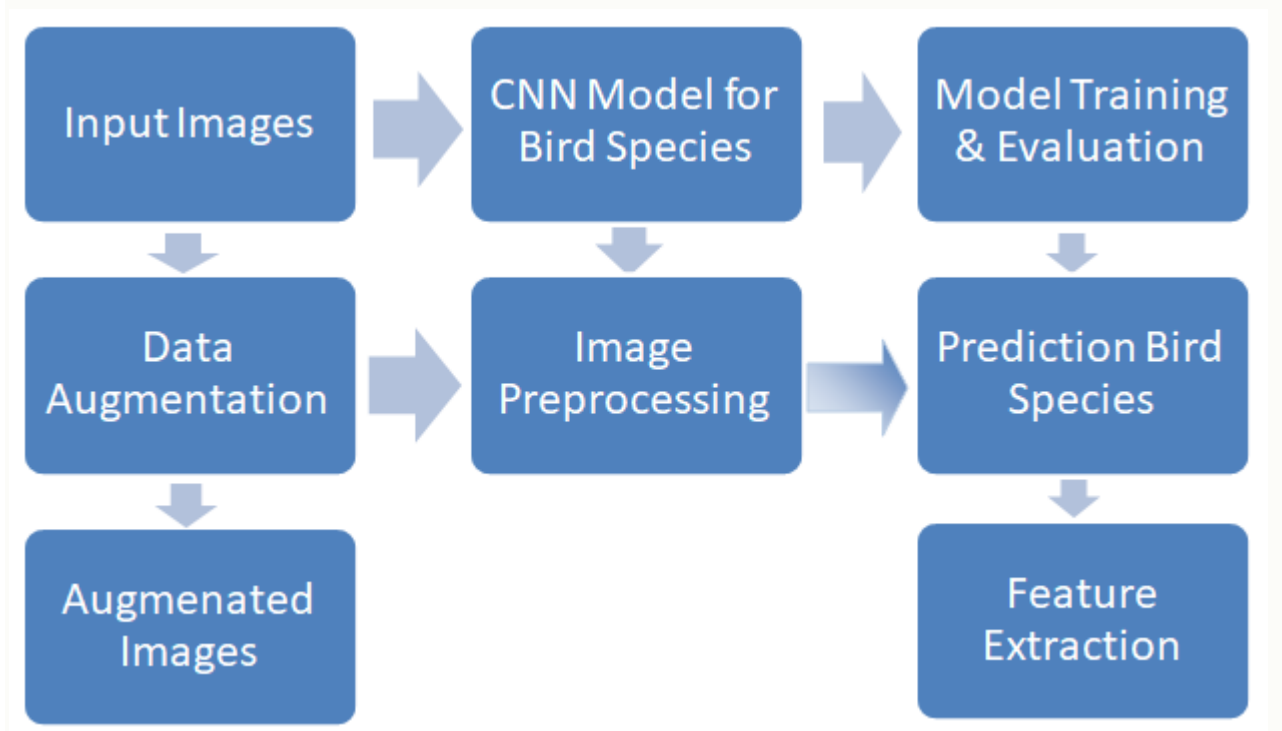
### 6.1 Architectural Design:



### 6.2 Data Flow Design:

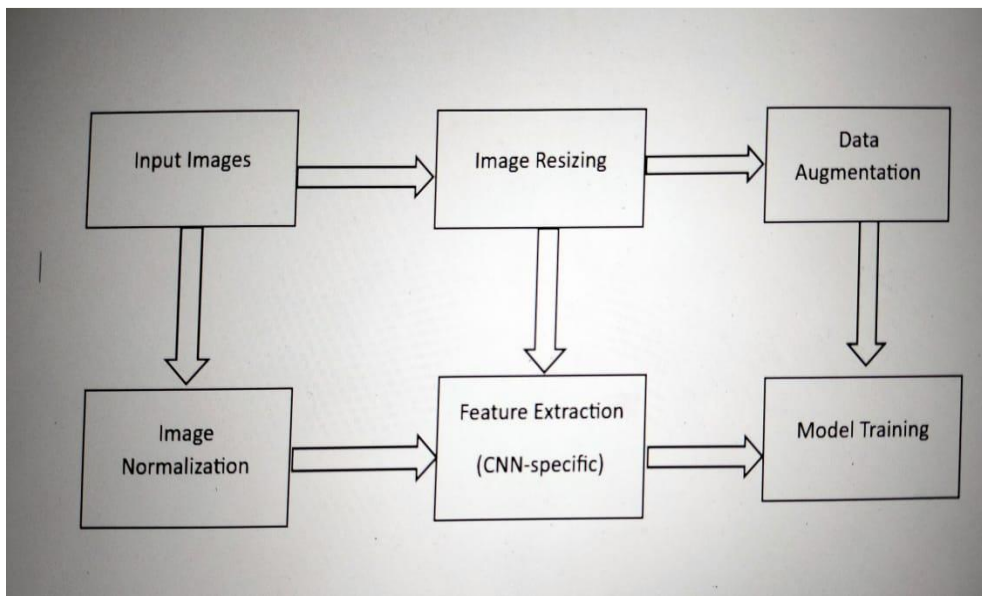
- The process starts with "Input Images" of birds that are fed into the system for prediction.
- The images undergo "Image Preprocessing" steps such as resizing, normalization, and enhancement to prepare them for input into the CNN model.
- Data Augmentation techniques are applied to increase the diversity of the training dataset and improve model generalization.
- "Feature Extraction" is performed by the CNN model to learn relevant features from the images for classification.

- The CNN model is trained and evaluated using the preprocessed images and extracted features.
- The trained model then predicts the bird species based on the input images, providing the "Predicted Bird Species" as the output.



### 6.3 Preprocessing Pipeline:

- The pipeline starts with input images of bird species that need to be classified by the CNN model.
- The images undergo resizing to ensure they are in a consistent format suitable for processing by the model.
- Data augmentation techniques are applied to increase the diversity of the training dataset and improve model generalization.
- Following that, image normalization is performed to standardize pixel values and enhance model performance.
- Feature extraction specific to CNNs is carried out to extract relevant features from the images for classification.
- The final step involves training the CNN model on the preprocessed image data to learn the patterns and make predictions on bird species.



#### 6.4 System Interaction:

- **User Interaction:** Since there's no user interface for uploading pictures, the system interaction may be limited to system administrators or automated processes responsible for managing data ingestion and preprocessing.
- **Command-Line Interface (CLI):** Develop a command-line interface or scripting interface to enable administrators to control system operations and configure data ingestion parameters.

#### 6.5 Integration with External System:

- **Data Integration:** Ensure seamless integration with external systems or data sources that provide input images for the bird species prediction system.
- **API Integration:** If the system interacts with other applications or services, design APIs for data exchange and communication, allowing for interoperability and integration.

## **6.6 Error Handling and Monitoring:**

- **Error Reporting:** Implement error handling mechanisms to detect and handle issues related to data ingestion, preprocessing, and system operation.
- **Logging and Monitoring:** Set up logging and monitoring systems to track system activities, detect anomalies, and provide visibility into the data ingestion pipeline.

## **6.7 Performance Optimization:**

- **Efficient Data Processing:** Optimize data processing algorithms and workflows to ensure efficient handling of incoming images and minimize processing latency.
- **Scalability Considerations:** Design the system to scale gracefully as the volume of incoming data increases, ensuring that it can handle large datasets effectively.

## **6.8 Security Measures:**

- **Access Control:** Apply access control measures to restrict access to sensitive system functionalities and data ingestion processes.
- **Data Encryption:** Implement encryption techniques to secure data transmission and storage, especially if the system interacts with external sources over insecure networks.

## **6.9 Data flow diagram:**

### **Data Collection:**

- ❖ Data sources such as bird databases, wildlife observations, and research studies provide information on various bird species.
- ❖ Data may include attributes like physical characteristics, habitat, location, behavior, and images/audio recordings.

### **Data Preprocessing:**

- ❖ Cleanse and preprocess the collected data to remove errors, inconsistencies, or missing values.
- ❖ Perform feature engineering to extract relevant features from raw data.
- ❖ Optionally, perform dimensionality reduction techniques to reduce the complexity of the dataset.

**Model Training:**

- ❖ Utilize machine learning algorithms such as convolutional neural networks (CNNs), random forests, or support vector machines (SVMs) to train the predictive model.
- ❖ Train the model using the preprocessed dataset, where the features are used to predict the bird species labels.

**Model Evaluation:**

- ❖ Assess the performance of the trained model using evaluation metrics such as accuracy, precision, recall, and F1-score.
- ❖ Validate the model's performance using techniques like cross-validation to ensure robustness and generalization.

**Model Deployment:**

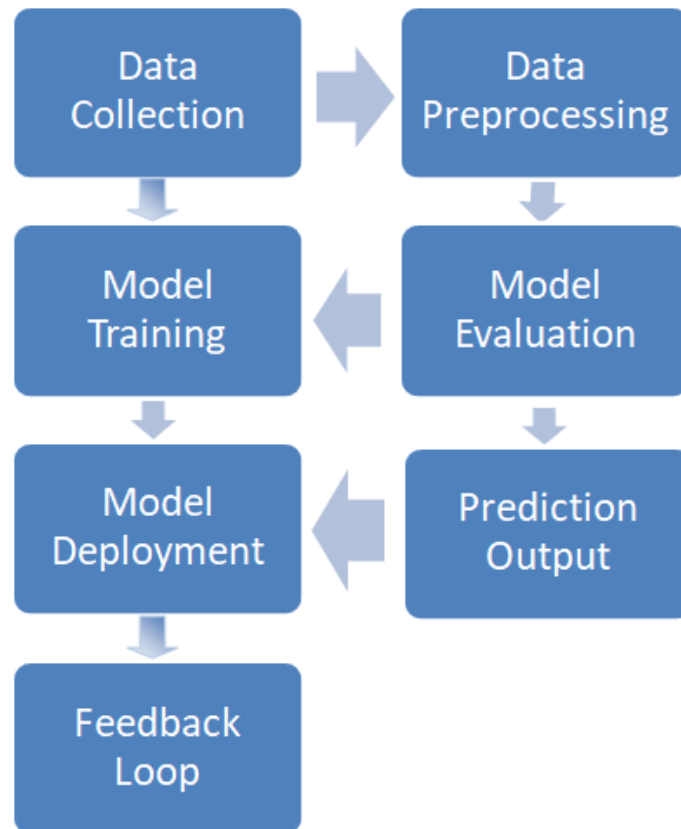
- ❖ Deploy the trained model as a user-friendly application or integrate it into existing platforms.
- ❖ Users input data such as bird images, or observational data into the deployed model for species prediction.

**Prediction Output:**

- ❖ The deployed model processes the input data and predicts the bird species label based on learned patterns.
- ❖ Output the predicted bird species label along with any additional information or metadata associated with the species.

**Feedback Loop:**

- ❖ Incorporate user feedback and model performance metrics to continuously improve the predictive model.
- ❖ Update the model periodically with new data and retrain it to adapt to changes in bird populations.



### 6.10 UML Diagrams:

UML stands for Unified Modeling Language. UML is a standardized general purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

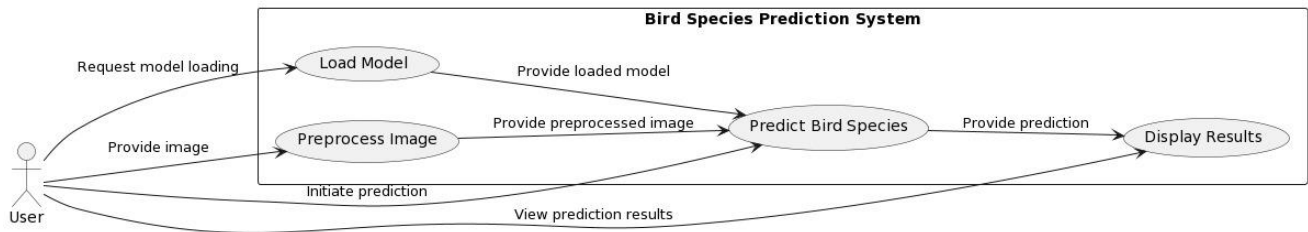
The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.



The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.



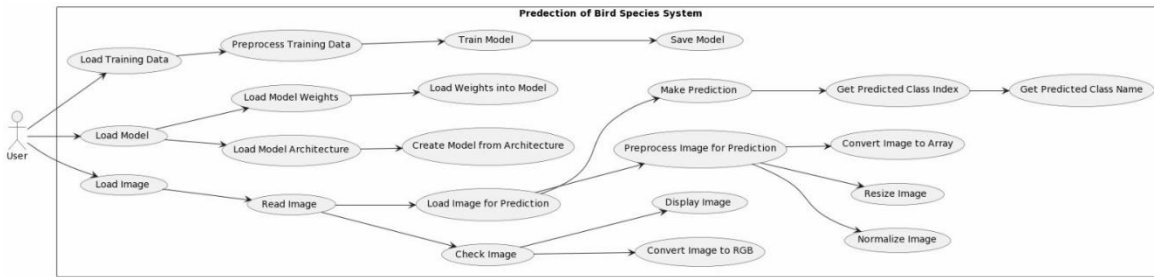
### 6.11 Goals:

The Primary goals in the design of the UML are as follows:

1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices

### 6.12 Use case Diagrams:

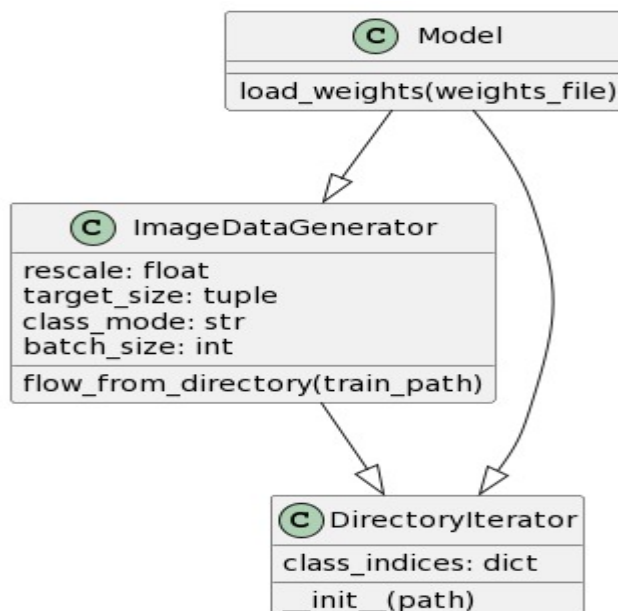
A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



### 6.13 Class Diagram:

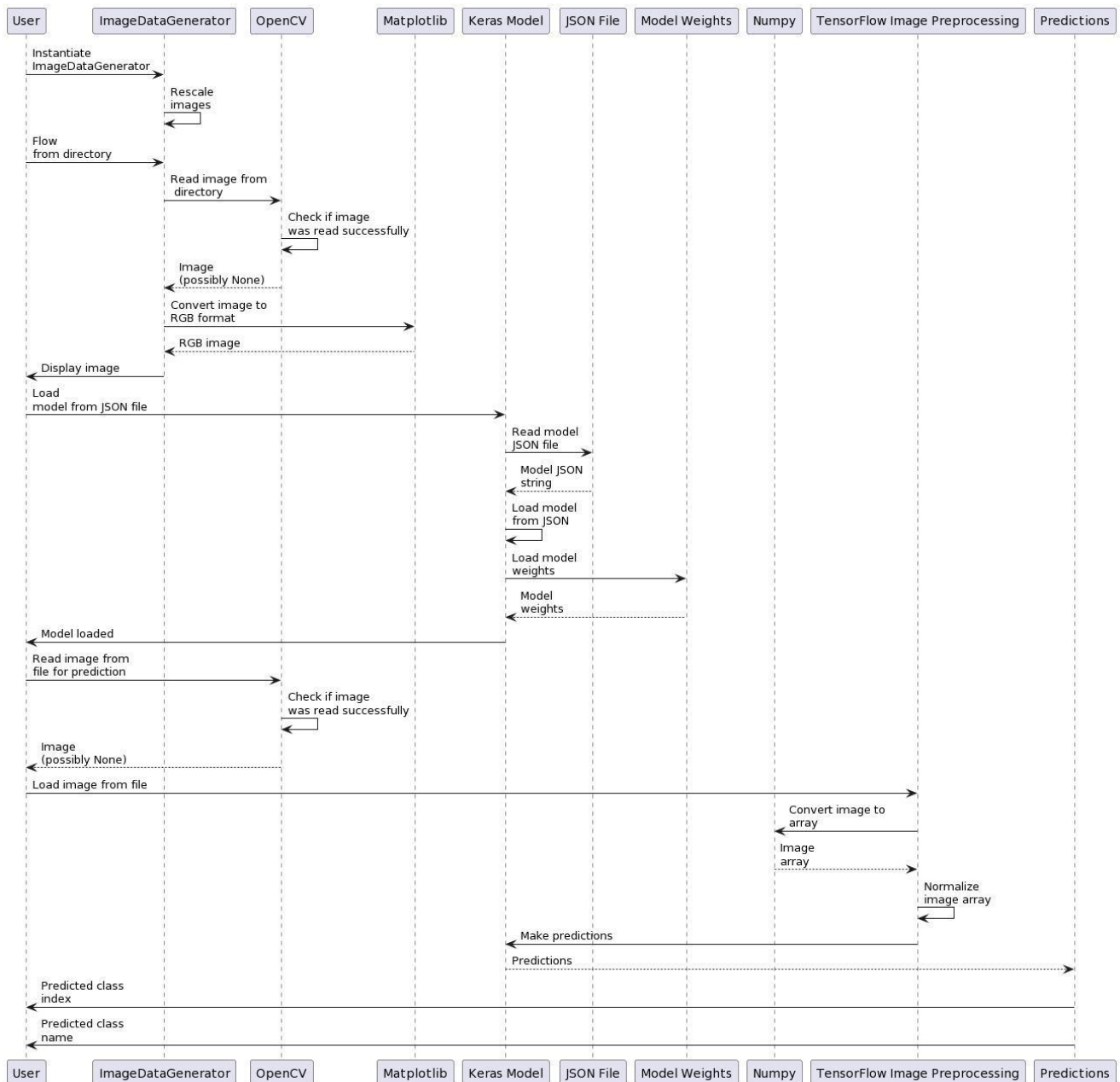
In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

In this study, we present a comprehensive approach for the prediction of bird species using Convolutional Neural Networks (CNNs). The dataset comprises images representing 200 different bird species, with 80% allocated for training and 20% for testing. We used a special kind of artificial intelligence called Convolutional Neural Networks (CNNs) to help our program understand.



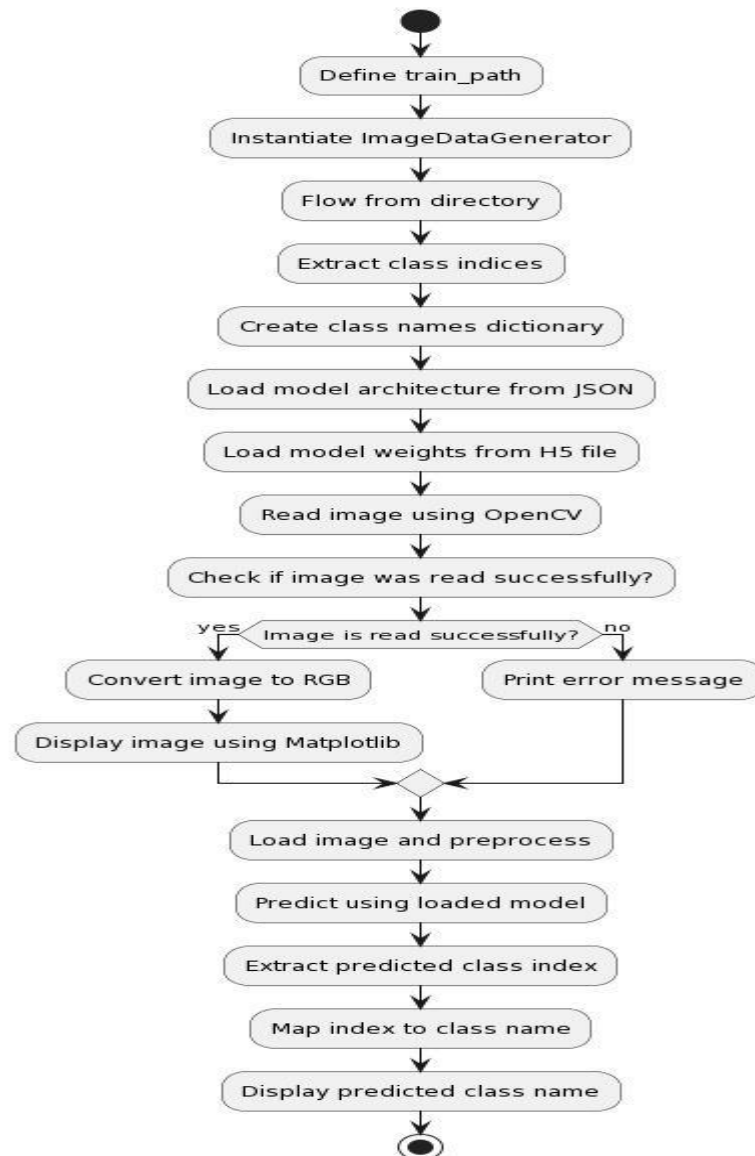
## 6.14 Sequence Diagram:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.



### 6.15 Activity Diagram:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



## **7. SOFTWARE ENVIRONMENT**

### **7.1 Programming Languages:**

#### **Python:**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An [interpreted language](#), Python has a design philosophy that emphasizes code [readability](#) (notably using [whitespace](#) indentation to delimit [code blocks](#) rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer [lines of code](#) than might be used in languages such as [C++](#) or [Java](#). It provides constructs that enable clear programming on both small and large scales. Python interpreters are available for many [operating systems](#). [CPython](#), the [reference implementation](#) of Python, is [open source](#) software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit [Python Software Foundation](#). Python features a [dynamic type](#) system and automatic [memory management](#). It supports multiple [programming paradigms](#), including [object oriented](#), [imperative](#), [functional](#) and [procedural](#), and has a large and comprehensive [standard library](#).

#### **Interactive Mode Programming**

Invoking the interpreter without passing a script file as a parameter brings up the following prompt –

```
$ python
```

```
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
```

```
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information. >>>
```

Type the following text at the Python prompt and press the Enter –

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in `print ("Hello, Python!")`; However in Python version 2.4.3, this produces the following result –

```
Hello, Python!
```

### **Script Mode Programming**

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py.

Type the following source code in a test.py file –

```
Live Demo print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows –

```
$ python test.py
```

This produces the following result –

```
Hello, Python!
```

Let us try another way to execute a Python script. Here is the modified test.py file –

```
Live Demo
```

```
#!/usr/bin/python
```

```
print "Hello, Python!"
```

We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows –

```
$ chmod +x test.py    # This is to make file executable
$ ./test.py
```

This produces the following result –

Hello, Python!

## **Python Identifiers**

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (\_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, \$, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers –

Class names start with an uppercase letter. All other identifiers start with a lowercase letter.

Starting an identifier with a single leading underscore indicates that the identifier is private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language defined special name.

## **Reserved Words**

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and      exec   not      assert finally   or   break   for      pass      class      from  
print continue   global   raise   def if      return   del      import try   elif   in  
while   else is   with   except   lambda   yield

## **Lines and Indentation**

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example –

```
if True:
```

```
    print "True" else:    print "False"
```

However, the following block generates an error –

```
if True:
```

```
print "Answer" print "True" else:
```

```
print "Answer" print "False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks –



Note – Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

```
#!/usr/bin/python
```

```
import sys
```

```
try:
```

```
    # open file stream
```

```
    file = open(file_name, "w") except IOError: print "There was an error writing to",  
file_name sys.exit() print "Enter ", file_finish, print "" When finished" while  
file_text != file_finish:
```

```
    file_text = raw_input("Enter text: ") if file_text == file_finish: # close the file  
file.close break file.write(file_text) file.write("\n") file.close() file_name =  
raw_input("Enter filename: ") if len(file_name) == 0:
```

```
    print "Next time please enter something" sys.exit() try:
```

```
    file = open(file_name, "r") except IOError: print "There was an error reading file"  
sys.exit() file_text = file.read() file.close() print file_text
```

### Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example –

```
total = item_one + \    item_two + \    item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example –

```
days = ['Monday', 'Tuesday', 'Wednesday',  
        'Thursday', 'Friday']
```

## Quotation in Python

Python accepts single ('), double (") and triple ('' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal –

```
word = 'word' sentence = "This is a sentence." paragraph = """This is a paragraph. It
is made up of multiple lines and sentences."""
```

## Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

### Live Demo

```
#!/usr/bin/python
```

```
# First comment
```

```
print "Hello, Python!" # second comment
```

This produces the following result –

```
Hello, Python!
```

You can type a comment on the same line after a statement or expression –

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows –

```
# This is a comment.
```

```
# This is a comment, too.
```

```
# This is a comment, too.
```

```
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
'''
```

```
This is a multiline comment.
```

```
'''
```

### Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

### Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action –

```
#!/usr/bin/python
```

```
raw_input("\n\nPress the enter key to exit.")
```

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

### Multiple Statements on a Single Line

The semicolon ( ; ) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon.

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

### Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite.

Header lines begin the statement (with the keyword) and terminate with a colon ( : ) and are followed by one or more lines which make up the suite. For example –

```
if expression :
```

```
    suite elif expression :
```

```
    suite else :    suite
```

### **Command Line Arguments**

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h –

```
$ python -h usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ... Options and arguments (and corresponding environment variables):
```

```
-c cmd : program passed in as string (terminates option list)
```

```
-d : debug output from parser (also PYTHON DEBUG=x)
```

```
-E : ignore environment variables (such as PYTHONPATH)
```

```
-h : print this help message and exit
```

You can also program your script in such a way that it should accept various options.

Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

### **Python Lists**

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example –

```
list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5 ]; list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example

–

```
tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5 ); tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing –

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value –

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

## Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example –

Live Demo

```
#!/usr/bin/python
```

```
tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5, 6, 7 ); print "tup1[0]:  
", tup1[0]; print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result –

```
tup1[0]: physics tup2[1:5]: [2, 3, 4, 5]
```

Updating Tuples

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} print "dict['Name']: ", dict['Name']  
print "dict['Age']: ", dict['Age']
```

When the above code is executed, it produces the following result –

```
dict['Name']: Zara dict['Age']: 7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} print "dict['Alice']: ", dict['Alice']
```

When the above code is executed, it produces the following result –

```
dict['Alice']:
```

Traceback (most recent call last):

```
File "test.py", line 4, in <module>      print "dict['Alice']: ", dict['Alice'];
```

```
KeyError: 'Alice'
```

### Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example

–

### Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
```

```
dict['Age'] = 8; # update existing entry dict['School'] = "DPS School"; # Add new entry
```

```
print "dict['Age']: ", dict['Age'] print "dict['School']: ", dict['School']
```

When the above code is executed, it produces the following result –

```
dict['Age']: 8 dict['School']: DPS School Delete Dictionary Elements
```

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'} del dict['Name']; # remove entry with  
key 'Name' dict.clear();    # remove all entries in dict del dict ;    # delete entire  
dictionary
```

```
print "dict['Age']: ", dict['Age'] print "dict['School']: ", dict['School']
```

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more –

```
dict['Age']:
```

```
Traceback (most recent call last):    File "test.py", line 8, in <module>        print  
"dict['Age']: ", dict['Age'];
```

```
TypeError: 'type' object is unsubscriptable
```

Note – del() method is discussed in subsequent section.

## Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys –



(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example –

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'} print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

```
dict['Name']: Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example –

Live Demo

```
#!/usr/bin/python
```

```
dict = [{'Name': 'Zara', 'Age': 7}] print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result –

```
Traceback (most recent call last):  File "test.py", line 3, in <module>      dict =  
[{'Name': 'Zara', 'Age': 7}];
```

```
TypeError: unhashable type: 'list'
```

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –

Live Demo

```
#!/usr/bin/python
```

```
tup1 = (12, 34.56); tup2 = ('abc', 'xyz');
```

```
# Following action is not valid for tuples
```

```
# tup1[0] = 100;
```

```
# So let's create a new tuple as follows tup3 = tup1 + tup2; print tup3;
```

When the above code is executed, it produces the following result –

```
(12, 34.56, 'abc', 'xyz')
```

### Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example –

### Live Demo

```
#!/usr/bin/python
```

```
tup = ('physics', 'chemistry', 1997, 2000); print tup; del tup; print "After deleting tup :  
"; print tup;
```

This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more –

```
('physics', 'chemistry', 1997, 2000) After deleting tup :
```

```
Traceback (most recent call last):  File "test.py", line 9, in <module>      print tup;  
NameError: name 'tup' is not defined
```

## **7.2 Deep Learning Frameworks:**

➤ TensorFlow and PyTorch are widely used deep learning frameworks that offer high-level APIs for building and training neural networks, including CNNs. These frameworks provide pre-trained models, optimization tools, and APIs for model deployment.

## **7.3 Data Processing and Analysis:**

➤ Libraries like NumPy, Pandas, and scikit-learn are used for data manipulation, preprocessing, and feature extraction. These tools enable efficient handling of large datasets and preparation of input data for training the CNN model.

## **7.4 Image Processing:**

➤ OpenCV (Open Source Computer Vision Library) is commonly used for image processing tasks such as resizing, normalization, and augmentation. It provides various functions for manipulating and analyzing images, which are essential for preparing training data and processing input images during inference.

## **7.5 Model Architecture and Training:**

➤ The CNN model architecture is designed using frameworks like TensorFlow's Keras API or PyTorch's nn.Module. These frameworks offer customizable layers and modules for building neural network architectures suited to image classification tasks.

➤ Training of the CNN model involves optimization algorithms like stochastic gradient descent (SGD) or its variants (e.g., Adam, RMSprop) to minimize the loss function. Techniques such as transfer learning, where pre-trained models (e.g., ImageNet) are fine-tuned on bird image datasets, can also be employed to boost performance with limited data.

## **7.6 Deployment and Integration:**

➤ Once trained, the CNN model can be deployed using various methods, including as a standalone application, web service, or integrated into existing platforms. Frameworks like TensorFlow Serving or Flask facilitate model deployment as RESTful APIs, allowing users to interact with the model via HTTP requests.

➤ Front-end technologies such as HTML, CSS, and JavaScript may be used to develop user interfaces for uploading images and displaying prediction results in web applications.

### **7.7 Feedback Loop and Model Updates:**

➤ Mechanisms for collecting user feedback and performance metrics are integrated into the software environment to continuously monitor the model's accuracy and performance. This feedback loop informs future iterations of model training and updates to improve prediction accuracy and user experience.

## 8. SAMPLE CODE

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.image as mpimg
import os
import random

from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

from matplotlib import pyplot as plt

from pathlib import Path
```

```
train_path = 'train'
valid_path = 'valid'
test_path = 'test'

batch_size = 128
image_size = (224, 224)
seed = 42
data_augmentation = True

print(f'Number of classes: {len(os.listdir(train_path))}')
```

```
def plot_random_image():
    random_class = random.sample(os.listdir(train_path), 1)
    random_img = random.sample(os.listdir(train_path + '/' + random_class[0]), 1)
    img = mpimg.imread(train_path + '/' + random_class[0] + '/' + random_img[0])

    plt.imshow(img)
    plt.title(random_class[0])
```

```
plt.subplot(2,3,2)
plot_random_image()

plt.subplot(2,3,3)
plot_random_image()
```

```
plt.subplot(2,3,4)
plot_random_image()
```

```
plt.subplot(2,3,5)
plot_random_image()
```

```
plt.subplot(2,3,6)
plot_random_image()
```

```
def get_random_image():
    random_class = random.sample(os.listdir(train_path), 1)
    random_img = random.sample(os.listdir(train_path + '/' + random_class[0]), 1)
    img = mpimg.imread(train_path + '/' + random_class[0] + '/' + random_img[0])

    return img
```

```
print(f'Image shape (height, width, channels) - {get_random_image().shape}')
```

```
def count_images_per_class(path):
    elements_by_class = { }

    for name_class in os.listdir(path):
        subdir_path = os.path.join(path, name_class)
        num_images = len([file for file in os.listdir(subdir_path) if file.lower().endswith(('.png', '.jpg', '.jpeg'))])
        elements_by_class[name_class] = num_images

    return sorted(elements_by_class.items(), key=lambda dict_element: dict_element[1], reverse=True)
```

```
elements_by_class = count_images_per_class(train_path)
```

```
print(f'The class with the largest number of elements is {elements_by_class[0][0]} with  
{elements_by_class[0][1]} elementos, and the class with the fewest elements is {elements_by_class[-1][0]} with  
{elements_by_class[-1][1]} elements')
```

```
n_classes= 75
classes, num_images = zip(*elements_by_class[:n_classes])
```

```
fig, ax = plt.subplots(figsize=(18, 5))
```

```
ax.bar(classes, num_images, color='skyblue', edgecolor='black')
```

```
ax.set_xticks(classes)
```

```
ax.set_xticklabels(classes, rotation='vertical', ha='center')
```

```

ax.set_xlabel('Classes')
ax.set_ylabel('Number of images')
ax.set_title(f'Images per class (First {n_classes} classes)')

plt.show()

```

```

if data_augmentation:
    train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255,
        rotation_range=30,
        zoom_range=0.15,
        horizontal_flip=True
    )
else:
    train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        rescale=1./255
    )

valid_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)

test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)

```

```

train_df = train_generator.flow_from_directory(
    train_path,
    target_size=image_size,
    seed=seed,
    class_mode='categorical',
    batch_size=batch_size
)

validation_df = valid_generator.flow_from_directory(
    valid_path,
    target_size=image_size,
    seed=seed,
    class_mode='categorical',
    batch_size=batch_size
)

```

```

test_df = valid_generator.flow_from_directory(
    test_path,
    target_size=image_size,
    seed=seed,
    class_mode='categorical',
    batch_size=batch_size
)

```

```

def get_model():
    model = Sequential()

    # Base Model
    model.add(layers.Conv2D(filters=16, kernel_size=2, input_shape=(image_size[0], image_size[1], 3),
padding='same', activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(2, padding='same'))

    model.add(layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(2, padding='same'))

    model.add(layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(2, padding='same'))

    model.add(layers.Conv2D(filters=128, kernel_size=4, padding='same', activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(2, padding='same'))

    model.add(layers.Conv2D(filters=256, kernel_size=3, padding='same', activation='relu'))
    model.add(layers.BatchNormalization())
    model.add(layers.MaxPooling2D(2, padding='same'))

    #Top Model
    model.add(layers.Flatten())
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.2))

    model.add(layers.Dense(256))
    model.add(layers.BatchNormalization())
    model.add(layers.Dropout(0.2))

```



```
model.add(layers.Dense(525, activation='softmax'))

model.summary()

return model

model = get_model()
```

```
checkpoint_path = "checkpoints"

early_stopping = tf.keras.callbacks.EarlyStopping(
    monitor='val_accuracy',
    patience=4,
    restore_best_weights=True,
    verbose=0
)

model_checkpoint = tf.keras.callbacks.ModelCheckpoint(
    checkpoint_path,
    monitor="val_accuracy",
    save_best_only=True,
    verbose=0
)
```

```
lr = 0.0015
epochs = 15
```

```
model.compile(
    optimizer = tf.keras.optimizers.Adam(lr),
    loss = 'categorical_crossentropy',
    metrics = ['accuracy']
)

H = model.fit(
    train_df,
    epochs=epochs,
    validation_data=validation_df,
    callbacks=[early_stopping, model_checkpoint]
)
```

```
model.summary()
```

```
validation_loss, validation_accuracy = model.evaluate(validation_df)
print("Validation Accuracy:", validation_accuracy)
```

```
test_loss, test_accuracy = model.evaluate(test_df)
print("Test Accuracy:", test_accuracy)
```

```
import cv2
img = cv2.imread('images (1).jpeg')

import matplotlib.pyplot as plt
plt.imshow(img)
```

```
from tensorflow.keras.preprocessing import image
img_path = 'images (1).jpeg' # Replace 'path_to_your_image.jpg' with the actual path to your image
img = image.load_img(img_path, target_size=(224,224)) # Replace height and width with your image size
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
```

```
img_array = img_array / 255.0 # Assuming you trained your model with images in the range [0, 255]
```

```
# Make predictions
predictions = model.predict(img_array)
```

```
predicted_class_index = np.argmax(predictions)
print(predicted_class_index)
```

```
class_indices = train_df.class_indices
class_names = {v: k for k, v in class_indices.items()}
print("Class names:", class_names)
```

```
predicted_class_name = class_names.get(predicted_class_index)
print("Predicted class name:", predicted_class_name)
```

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model_weights.h5")
```

## **9. SYSTEM TEST**

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides way to check the functionality of components, sub assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

### **9.1 TYPES OF TESTS**

#### **9.1.1 Unit testing**

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

#### **9.1.2 Integration testing**

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent.

Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

### **9.1.3 Functional test**

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

- Valid Input : identified classes of valid input must be accepted.
- Invalid Input : identified classes of invalid input must be rejected.
- Functions : identified functions must be exercised.
- Output : identified classes of application outputs must be exercised.
- Systems/Procedures : interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing. Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

### **9.1.4 System Test**

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

### **9.1.5 White Box Testing**

White Box Testing is a testing in which the software tester has knowledge of the inner workings, structure and language of the software, or at least

its purpose. It has a purpose. It is used to test areas that cannot be reached from a black box level.

### **9.1.6 Black Box Testing**

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as specification or requirements document. It is a testing in which the software under test is treated, as a black box .You cannot “see” into it. The test provides inputs and responds to outputs without considering how the software works.

### **9.1.7 Unit Testing**

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

## **9.2 Test strategy and approach**

Field testing will be performed manually and functional tests will be written in detail.

### **Test objectives**

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

### **Features to be tested**

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

### ***9.2.1 Integration Testing***

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.







### **Acceptance Testing**

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.







**Test Results:** All the test cases mentioned above passed successfully. No defects encountered.

## 10. Sample Test Cases

Test Case Id	Input	Actual Output	Expected Output	Status
TC01		CROW	CROW	PASS
TC02		PEACOCK	PEACOCK	PASS
TC03		GOLDEN EAGLE	GOLDEN EAGLE	PASS
TC04		VIOLET CUCKOO	VIOLET CUCKOO	PASS
TC05		TURKEY VULTURE	TURKEY VULTURE	PASS
TC06		JAVA SPARROW	JAVA SPARROW	PASS

TC07		BLACK-THROATED SPARROW	BLACK-THROATED SPARROW	PASS
TC08		TURKEY VULTURE	BLACK COCKATO	FAIL
TC09		SPOTTED WHISTLING DUCK	SPOTTED WHISTLING DUCK	PASS
TC10		LUCIFER HUMMING BIRD	LUCIFER HUMMING BIRD	PASS
TC11		OILBIRD	OILBIRD	PASS
TC12		DALMATIAN PELICAN	DALMATIAN PELICAN	PASS



<b>TC13</b>		<b>FAIRY PENGUIN</b>	<b>FAIRY PENGUIN</b>	<b>PASS</b>
<b>TC14</b>		<b>RUFOUS KINGFISHER</b>	<b>RUFOUS KINGFISHER</b>	<b>PASS</b>
<b>TC15</b>		<b>RUDY KINGFISHER</b>	<b>RUDY KINGFISHER</b>	<b>PASS</b>
<b>TC16</b>		<b>VARIED THRUSH</b>	<b>VARIED THRUSH</b>	<b>PASS</b>
<b>TC17</b>		<b>JAPANESE ROBIN</b>	<b>JAPANESE ROBIN</b>	<b>PASS</b>
<b>TC18</b>		<b>MIKADO PHEASANT</b>	<b>ABYSSINIAN GROUND HORNBILL</b>	<b>FAIL</b>
<b>TC19</b>		<b>ROSY FACED LOVEBIRD</b>	<b>ROSY FACED LOVEBIRD</b>	<b>PASS</b>

## Screen Shots:

```
1s ✓ ▶ from keras.models import model_from_json
    json_file = open('model.json', 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    emotion_model = model_from_json(loaded_model_json)

    # load weights into new model
    emotion_model.load_weights("model_weights.h5")
    print("Loaded model from disk")

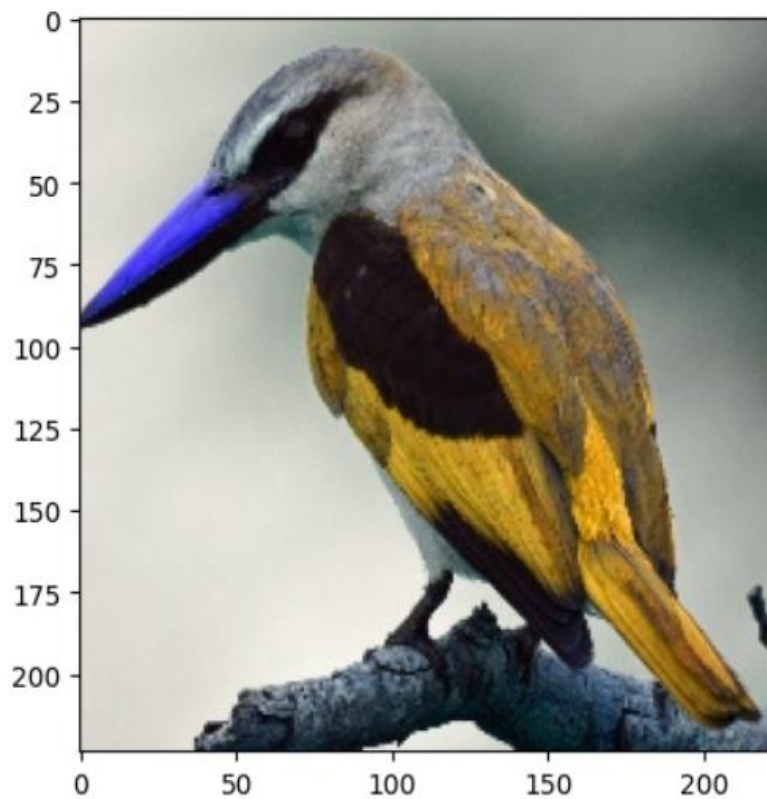
Loaded model from disk
```

```
✓ ▶ import cv2
    img = cv2.imread('2.jpg')
```

✓  
0s

```
▶ import matplotlib.pyplot as plt  
plt.imshow(img)
```

➞ <matplotlib.image.AxesImage at 0x7d8609b774c0>



```
✓ [45] import numpy as np  
      from tensorflow.keras.preprocessing import image  
      img_path = '2.jpg' # Replace 'path_to_your_image.jpg' with the actual path to your image  
      img = image.load_img(img_path, target_size=(224,224)) # Replace height and width with your image size  
      img_array = image.img_to_array(img)  
      img_array = np.expand_dims(img_array, axis=0)
```

```
✓ [46] img_array = img_array / 255.0 # Assuming you trained your model with images in the range [0, 255]  
      predictions = emotion_model.predict(img_array)
```

1/1 [=====] - 0s 70ms/step

```
✓ [47] predicted_class_index = np.argmax(predictions)  
      print(predicted_class_index)
```

518

```
✓ [51] predicted_class_name = Class_names.get(predicted_class_index)  
      print("Predicted class name:", predicted_class_name)
```

Predicted class name: WOODLAND KINGFISHER

## **12. CONCLUSION**

This project presents a robust approach for bird species prediction utilizing Convolutional Neural Networks (CNNs). With a dataset encompassing images of 200 bird species split into training and testing sets, the CNN architecture was employed to discern distinct features within the images. By preprocessing the images—resizing, converting to grayscale, and applying Gaussian blur—the program was primed to recognize birds accurately. Through training on 80% of the data and testing on the remaining 20%, the program demonstrated promising accuracy in identifying bird species. This technology holds potential for wildlife monitoring and conservation efforts, laying a foundation for future research in leveraging computer vision for animal identification, particularly in bird watching.

### **12.1 FURTHER ENHANCEMENT**

The work exhibited the expected utilization of machine learning methods in prediction of bird species on the given attributes. The created web application is easy to understand and the testing accuracy is over 80%. So, it's better to design a system with more Accuracy rate

### **12.2 BIBLIOGRAPHY**

1. P. Srinadh Department of CSE Koneru Lakshmiah Education Foundation Vaddeswaram, Andhra Pradesh, India.” Image Classification using Deep Learning and Neural Networks”.2023 International Journal of Engineering Research & Technology (IJERT) <http://www.ijert.org> ISSN: 2278-0181 IJERTV12IS030117. DOI: 10.17577/IJERTV12IS030117
2. Carlos N. Silla Jr. School of Computing, University of Kent at Canterbury Canterbury, UK. “Automatic Bird Species Identification for Large Number of Species” . 2011 IEEE International Symposium on Multimedia. DOI: 10.1109/ISM.2011.27
3. Prof. Pralhad Gavali, Ms. Prachi Abhijeet Mhetre, Ms. Neha Chandrakhant Patil, Ms. Nikita Suresh Bamane, Ms. Harshal Dipak Buva Rajarambapu Institute of Technology, Rajaramnagar, India “Bird Species Identification using Deep Learning “ 2019 International Journal of Engineering Research & Technology

DOI : 10.17577/IJERTV8IS040112

4. YOSIHIRO NATUHARA\* and CHOBEI IMAI Osaka City Institute of Public Health and Environmental Sciences, 8-34 Tojo-cho, Tennoji-ku, Osaka 543, Japan .” Prediction of species richness of breeding birds by landscape-level factors of urban woods in Osaka Prefecture, Japan “ 1999 INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY DOI :10.1023/a:1008869410668

5. Carlos N. Silla Jr Post-Graduate Program in Informatics (PPGI) Federal University of Technology of Parana (UTFPR-CP) Cornelio Procopio, Brazil ´ carlosjunior@utfpr.edu.br “Hierarchical Classification of Bird Species Using Their Audio Recorded Songs “ 2013 IEEE International Conference on Systems DOI: 10.1109/SMC.2013.326

6. Yuanchun Zhou\* , Jing Shao\* , Xuezhi Wang\* , Ze Luo\* , Jianhui Li\* and Baoping Yan\* \* Computer Network Information Center Chinese Academy of Sciences, Beijing, China P.O.BOX 100190 “Bird-SDPS: A Migratory Birds’ Spatial Distribution Prediction System “2013 IEEE 9th International Conference on e-Science DOI: 10.1109/eScience.2013.12

7. Revathy Narasimhan, Xiaoli Z. Fern, Raviv Raich School of EECS, Oregon State University, Corvallis, OR 97331-5501, USA {narasimr, xfern, raich} “SIMULTANEOUS SEGMENTATION AND CLASSIFICATION OF BIRD SONG USING CNN” 2017 IEEE International Symposium on Multimedia DOI: 10.1109/ICASSP.2017.7952135

8. Wen Li and Dezhen Song “Automatic Bird Species Detection Using Periodicity of Salient Extremities” 2013 IEEE International Conference on Robotics and Automation (ICRA) DOI: 10.1109/ICRA.2013.6631407