

A Project Report

on

BIRD SPECIES DETECTION USING VGG-16 ALGORITHM

submitted in partial fulfillment of the requirements for the award

of

Degree of Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING

by

O. JWALA PRASAD

(20FE1A05C8)

N. BHARATH

(20FE1A05C0)

K. LAHARI

(20FE1A0567)

N. VINAY KUMAR REDDY

(20FE1A05C2)

Under the Guidance of

Mr. K. PRADEEP, M.Tech

ASSISTANT PROFESSOR

Department of Computer Science and Engineering



VIGNAN'S LARA
INSTITUTE OF TECHNOLOGY & SCIENCE
(AUTONOMOUS)

Approved by AICTE New Delhi & Affiliated to JNTUK Kakinada
Accredited by **NAAC 'A+'** and **NBA** | **ISO 9001 : 2015**
Vadlamudi - 522 213, Guntur District

May-2024



VIGNAN'S LARA

INSTITUTE OF TECHNOLOGY & SCIENCE

(AUTONOMOUS)

Approved by AICTE New Delhi & Affiliated to JNTUK Kakinada

Accredited by **NAAC 'A+' and NBA | ISO 9001 : 2015**

Vadlamudi - 522 213, Guntur District

DEPATMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that a Project Report entitled " **BIRD SPECIES DETECTION USING VGG-16 ALGORITHM** " is a bonafide work done by **O. JWALA PRASAD (20FE1A05C8), N. BHARATH (20FE1A05C0), K. LAHARI (20FE1A0567) and N.VINAY KUMAR REDDY (20FE1A05C2)** under my guidance and submitted in fulfilment of the requirements for the degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING** from **JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY KAKINADA, KAKINADA.** The work embodied in this Project Report is not submitted to any University or Institution for the award of any Degree or diploma.

Project Guide

Mr. K. PRADEEP ,M .Tech

Assistant Professor

Head of the Department

Dr. K. VENKATESWARA RAO

Professor

External Examiner

DECLARATION

We hereby declare that a Major Project report entitled “**BIRD SPECIES DETECTION USING VGG-16 ALGORITHM**” is a record of original work done by us under the guidance of **Mr. K. PRADEEP, M.Tech.**, Assistant Professor of Computer Science and Engineering, and submitted in partial fulfillment of the requirements for the Degree of Bachelor of Technology in Computer Science and Engineering. The results embodied in this project report are not submitted to any other University or Institute.

Project Members

Signature

O. Jwala prasad (20FE1A05C8)

N. Bharath (20FE1A05C0)

K. Lahari (20FE1A0567)

N. Vinay Kumar Reddy (20FE1A05C2)

Place: Vadlamudi

Date:

ACKNOWLEDGEMENT

The satisfaction that accomplishes the successful completion of any task would be incomplete without the mention of people whose ceaseless cooperation made it possible , whose constant guidance and encouragement crown all efforts with success.

We are grateful to **Mr .K.PRADEEP**, Assistant professor, Department of Computer Science and Engineering for guiding us through this project and for encouraging us right from the beginning of the project till its successful completion of the project.

We thank **Dr. K.VENKATESWARA RAO** , Professor & HOD ,Department of Computer Science and Engineering for their support and valuable suggestions.

We also express our thanks to **Dr. K .PHANEENDRA KUMAR**, Principal , Vignan's Lara Institute of Technology & Science for providing the resources to carry out the project.

We also express our sincere thanks to our beloved **Chairman Dr. LAVU RATHAIAH** for providing support and stimulating the environment for developing the project.

We also place our floral gratitude to all other teaching and lab teaching and lab technicians for their constant support and advice throughout the project.

Project Members

O.Jwala prasad (20FE1A05C8)

N.Bharath (20FE1A05C0)

K.Lahari (20FE1A0567)

N.Vinay Kumar (20FE1A05C2)

LIST OF CONTENTS

DESCRIPTION	PAGE NUMBERS
ABSTRACT	i
LIST OF FIGURES	ii
CHAPTER 1 : INTRODUCTION	1-5
1.1 Literature Survey	2
1.2 Existing System	3
1.3 Proposed System	4
CHAPTER 2 : PROJECT PLANNING AND MANAGEMENT	6-7
2.1 Objective	6
2.2 Scope	6
2.3 System Functions	7
CHAPTER 3 : SYSTEM DESIGN	8-15
3.1 Data Dictionary	8
3.2 Logical Database Design	9-11
3.2.1 Database Design	9
3.2.2 Dataset	10
3.3 UML Diagrams	12-14
3.3.1 UseCase Diagram	12
3.3.2 Sequence Diagram	13
3.3.3 Activity Diagram	14

CHAPTER 4 : REQUIREMENT ANALYSIS	16-17
4.1 H/w & S/w Requirements	16
4.1.1 Software Requiremnets	16
4.1.2 Hardware Requirements	16
4.2 Software Requirements Specification	17
CHAPTER 5 :SYSTEM IMPLEMENTATION	18-38
5.1 Machine Learning	18
5.1.1 Machine Learning Tasks	18-20
5.1.1.1 Supervised Learning	18
5.1.1.2 Unsupervised Learning	19
5.1.2 Machine Learning Approaches	20
5.1.3 Machine Learning Process	20-21
5.1.3.1 Data Preparation	20
5.1.3.2 Choosing Model	20
5.1.3.3 Training	21
5.1.3.4 Evaluation	21
5.2 Selected Software	21-26
5.2.1 Python	21
5.2.2 Packages using Python	25
5.2.3 Spyder Editor	26
5.3 Modules	27-38
5.3.1 Pre-Processing	27
5.3.2 Classification	27
5.3.3 Multilayer feed forward Network	28
5.3.4 Convolution Neural Network	29

CHAPTER 6: TESTING	39-50
6.1 Introduction	39
6.1.1 Types Of Tests	39
6.1.2 Testing Methods	40
6.2 Need for Testing	44
6.3 Test cases	45
6.4 Software Testing	47
6.5 Testing Strategies	48
CHAPTER 7 : SCREENS&REPORTS	51-55
7.1 screens	51
7.2 Reports	53
CHAPTER 8 : CONCLUSION AND FUTURE SCOPE	56
11.1 Conclusion	56
11.2 Future Scope	56
REFERENCES	57-57
APPENDIX	58-67

ABSTRACT

Birds are an integral part of an environment and they are of the utmost importance to nature. Considering this, it is clear how necessary it is to be able to identify birds in the wilderness. Nowadays some bird species are being found rarely and if found classification of bird species prediction is difficult. Ornithologists who often work on reporting bird activity need some kind of assistance to deal with the reporting. Numerous bird books have been published to assist bird watchers and ornithologists in order to determining the correct species. However, the identification of birds is an impractical piece of work to be done manually. Image-based bird species identification involves various techniques like Open CV, CNNs and few more which are the most important image processing techniques to predict the type of bird species. Deep learning techniques can be used for identifying the birds .We developed a model by using VGG(visual Geometry Group)-16 which is a deep convolution neural network for easily identifying bird species. Additionally, the model discusses the importance of benchmark datasets for evaluating the performance of different bird species detection algorithms. Several widely used datasets are analyzed, along with their characteristics and limitations, highlighting the need for standardized evaluation protocols in this field.

LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO.
3.1	Gull Dataset	10
3.2	Oriole Dataset	10
3.3	Sparrow Dataset	11
3.4	Use Case Diagram	12
3.5	Sequence Diagram	14
3.6	Activity Diagram	15
5.1	Training Chart	21
5.2	CNN Architecture	29
5.3	Padding	31
5.4	Activation Function	31
5.5	Pooling Layer Structure	32
5.6	Dense Layer Structure	35
5.7	Components of CNN	35
5.8	Pooling Layer	36
5.9	Drop out Technique	38
7.1	Output Interface	51

7.2	User Folder Interface	52
7.3	User File selection Interface	52
7.4	Bird Information Buttons	53
7.5	Predicted Output Interface	54
7.6	Bird Information Interface	55

CHAPTER 1

INTRODUCTION

Bird behavior and population trends have become an important issue now-a-day. Birds help us to detect other organisms in the environment. An important problem in ecology, which is the study of interactions between organisms and environment, is to monitor bird populations. The use of acoustics to monitor and classify birds in their natural environments has received a lot of interest lately. Classification of bird species based on image data so, for example useful when monitoring breeding behavior, biodiversity and population dynamics. Bird behavior and population trends have become an important issue now-a-day. Birds help us to detect other organisms in the environment. An important problem in ecology, which is the study of interactions between organisms and environment, is to monitor bird populations. The use of acoustics to monitor and classify birds in their natural environments has received a lot of interest lately. Classification of bird species based on image data so, for example useful when monitoring breeding behavior, biodiversity and population dynamics.

Now a day's bird watching is a recreational activity that can provide relaxation in daily life and it's a responsibility to know about our nature because birds are part of our society. Someone who does this is called a birdwatcher or birder. The scientific study of birds is called ornithology. People who study birds as a profession are called ornithologists.

We have nearly 18,000 bird species on our beautiful earth by the new research led by American Museum of Natural History. Birds that look similar to one another or thought to interbreed, but actually different species.

Our main Aim is about bird identification technology to maintain a data base of birds species , like a gallery for the generations because our ancestors history data is given in the form of book and papers but for our future generations we have to give the data by using technology So, that we built this technology for everyone to check the birds thesis easily by the image identification.

1.1 Literature Survey:-

Aditya Bhandari, Ameya Joshi, Rohit Patki used data set Caltech UCSD 200 bird species. The entire design was based on a python library Scikit and algorithms like **Naive Bayes, Support Vector Machines (SVM), KNN**, were tried out. The final observation was that the highest accuracy obtained was for **Logistic Regression** method using **Mturks** as an extraction method. The accuracy obtained is higher compared to modules when SVM or SVM+CNN - Learning method but lower when compared Logistic Regression-Learning method. The proposed model generates output with an accuracy rate of **53.65 %** where Mturks is the feature extraction method and Logistic Regression is Learning method.

Andreia Marini, Jacques Facon and Alessandro L. Koerich used Caltech UCSD-200 dataset in this module. First, a **colour segmentation method** for removing background elements and possible locations where the bird might be present. Thereafter, the image is split into planes, and normalised histograms are generated for each plane. The number of bins is then reduced via aggregation processing. An algorithm uses these histogram bins as feature vectors to discriminate between the number of species. The proposed module achieved a segmentation accuracy rate of **75%**.

Saundarya Junjur, Punam Avhad, Deepika Tendulkar used **Deep Learning algorithm** victimization with CNN architecture. After transforming the image uploaded into grey scale this method was applied. The main goal of the application is to identify the name of the bird with image as an input. The dataset used here is Caltech-UCSD with across 200 different types of bird species. The architecture consists of five convolution layers, one activation layer ,one pooling and one dense layer. The accuracy of predicating the bird's name given image is about **83.3%**.

Suleyman A. Al-Showarah, Sohyb T. Al-qbailat used **ANN (Artificial Neural Network)** algorithm and the various operations were performed like combine, max, min and average between the f6/g7. Based on the result of classifiers: The classification accuracy of ANN was **70.9908%**, the recall was 0.71%, and the f-measure was 0.708. The highest accuracy of image identification of about 70.9908% with ANN algorithm.

Bhandare, Amit Tambade developed an application that utilizes Deep Convolution Neural Network (**DCNN**) and Unsupervised learning calculation respectively. The data set preparation is completed by Google-Collab. The architecture consists of five convolution layers, one activation layer ,one pooling and one dense layer. The result of the application has the range of exactness between **80% to 90%** respectively which utilized Tensor Stream library.

Huang(author) designed an automatic model which was made to classify the **27 endemic birds** of Taiwan by skipped **CNN** (Convolution Neural Network) model. The purpose behind skip connection was to give an uninterrupted gradient flow from the first layer to last layer, so it can solve the disappearing gradient problem. The proposed model was able to identify the uploaded image of a bird as bird with **95%** accuracy.

Nadimpalli(author) focuses on bird detection and analyzes the motion detection with image subtraction, bird detection with template matching and bird detection with the **Viola-Jones Algorithms**. Out of all the methods, bird detection with Viola Jones Algorithm had the highest accuracy (**87%**) with a low false positive rate. The Viola-Jones algorithm can be trained for almost any object as long as there are many similar positive images that can be used for training the classifier.

Marcelo T. Lopes, Lucas L. Gioppo et al focused on the automatic identification of bird species from their audio recorded song. Here the authors dealt with the bird species identification problem using **signal processing** and **machine learning** techniques with the MARSYAS feature set. Presented a series of experiments conducted in a database composed of bird songs from 75 species out of which problem obtained in performance with 12 species.

Peter Jancovic and Munevver Kokuer Investigated acoustic modelling for recognition of bird species from audio field recordings. Developed a hybrid deep neural network hidden Markov model (**DNNHMM**). The developed models were employed for bird species identification, detection of specific species and recognition of multiple bird species vocalizing in a given recording. In this paper, the authors achieved an identification accuracy of **92.7%**.

Nyaga,G.M A Mobile-based image recognition system for identifying bird species in Kenya indicates that a **Machine Learning Algorithm** is used to classify images to detect bird

species and to predict behavioral patterns. An image classification algorithm used to detect the bird species. The accuracy of this model is around **87%**.

1.2 Bird Species Prediction

Several existing systems for bird species detection use combination on machine Learning. These systems typically involve training models on small datasets of bird images and then using these models to classify and identify bird species in new image. By using machine learning approach we get upto 85% accuracy

Disadvantages of Bird Species Prediction

Limited Accuracy: The accuracy from these models are less when compared to the newly generated model. The newly generated model comes with a high accuracy of 95%. **Lack Of**

Features: Features like bird description, bird voice are not mentioned in these existing systems.

Applicable for only small dataset: The size of the dataset is less when compared to the newly generated model. The newly generated model has a dataset size of 525.

1.3 Bird Species Detection Using VGG-16 Algorithm

Our project leverages the VGG-16 algorithm, a deep learning approach, to discern bird species from images. Utilizing a pre-trained VGG-16 model as a feature extractor, we refine its capabilities through tuning on a specialized bird species dataset, allowing adaptation to specific avian characteristics. Employing an 80:20 ratio for training and testing data, our model achieves an impressive accuracy rate of up to 95%.

Advantages:

Improved accuracy: The proposed system increases the accuracy (95%) of bird species detection comparing with all other systems because we use the VGG-16 algorithm.

Scalability : The proposed system supports huge images comparing with all other systems. This work on 525 bird species consists of 84,635 training images and consists of 2,622 testing images.

Features : This proposed system not only predict name of the bird and also gather the complete information about bird like location, voice of bird.

Research and Education: The proposed system contribute to scientific research by generating data for studies on avian behavior, migration patterns, population dynamics, and

evolutionary. biology. Additionally, these projects offer opportunities for public education and engagement in citizen science initiatives.

Ecological Balance: Birds play crucial roles in ecosystems such as seed dispersal, pollination, insect control, and nutrient cycling. Detecting bird species helps in assessing ecosystem health and promoting ecological balance.

CHAPTER 2

REQUIREMENT ANALYSIS

2.1 Objective :-

Identification of bird species is a challenging task often resulting in ambiguous labels. Even professional bird watchers sometimes disagree on the species given an image of a bird. It is difficult problem that pushes the limits of the visual abilities for both humans and computers. Although different bird species share the same basic set of parts, different bird species can vary dramatically in shape and appearance. Intraclass variance is high due to variation in lighting and background and extreme variation in pose (e.g., flying birds, swimming birds, and perched birds that are partially occluded by branches). Our project aims to employ the power of machine learning to help amateur bird watchers identify Bird species from the images they capture.

2.2 Scope :-

Overview The proposed model is used to classify the image identify bird from the image that we uploaded. It will take less time to complete the process. This system gives peculiar results with good accuracy.

- **Exclusions:**

Displays the name of the specific bird

- **Assumptions:**

It can only find the bird which is in database.

2.3 System Functions

System functions refer to the set of operations or tasks performed by a computer system or software application to accomplish specific objectives or provide desired functionality. These functions are the building blocks of a system's behavior and are designed to perform various tasks efficiently and reliably. System functions can encompass a wide range of activities, including data processing, input/output operations, communication with external devices or systems, and management of resources.

2.3.1 Input:

We give image as input. The system may process the image to enhance its quality, extract features, or perform various transformations such as resizing, cropping, filtering, or noise reduction. Image processing techniques can be applied for tasks like image recognition, object detection, or image enhancement.

2.3.2 Model:

The system may extract relevant features or descriptors from the image to represent its characteristics in a suitable format for further analysis or processing. Feature extraction is commonly used in machine learning and pattern recognition tasks to identify distinctive attributes of objects or patterns within images. The system may employ pattern recognition techniques to identify specific patterns, structures, or objects within the image. This can involve comparing the image against predefined templates,

CHAPTER 3

SYSTEM DESIGN

3.1 Data Dictionary

After understanding the requirements of the candidates, the entire data storage requirements are divided into tables. The below tables are normalized to avoid any anomalies during the course of data entry. A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is a crucial component of any relational database.

A data dictionary contains metadata that is data about the database. The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc. The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators. The data dictionary, in general, contains information about the following:

- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views that is visible.

Importance of data dictionary:

Analysts use data dictionaries for the following reasons:

- To manage the details in large systems.
- To communicate a common meaning for all elements
- To document the features of the system
- To locate errors and omissions in the system.

3.2 Logical database design

A data dictionary is a file or a set of files that contains a database's metadata. The data dictionary contains records about other objects in the database, such as data ownership, data relationships to other objects, and other data. The data dictionary is a crucial component of any relational database. Ironically, because of its importance, it is invisible to most database users. Typically, only database administrators interact with the data dictionary.

3.2.1 Database Design

Database design is the process of designation database files, which are the key source of information of the system. The files should properly design, planned for collection, accumulation, editing the required information. The objectives of the database design are to provide effective auxiliary storage and to contribute to the overall efficiency of the computer program component of the system.

3.2.2 Dataset

In Machine learning, we use the data set as the data that is given to the algorithm to create a model. A data set is a collection of related, discrete items of related data that may be accessed individually or in combination or managed as a whole entity. A data set is organized into some type of data structure. In a database, for example, a data set might contain a collection of business data (names, salaries, contact information, sales figures, and so forth). The database itself can be considered a data set, as can bodies of data within it related to a particular type of information, such as sales data for a particular corporate department. A data set (or dataset) is a collection of data. Most commonly a data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. The data set may comprise data for one or more members, corresponding to the number of rows. The term data set may also be used more loosely, to refer to the data in a collection of closely related tables, corresponding to a particular experiment or event.



Fig 3.1 Gull dataset

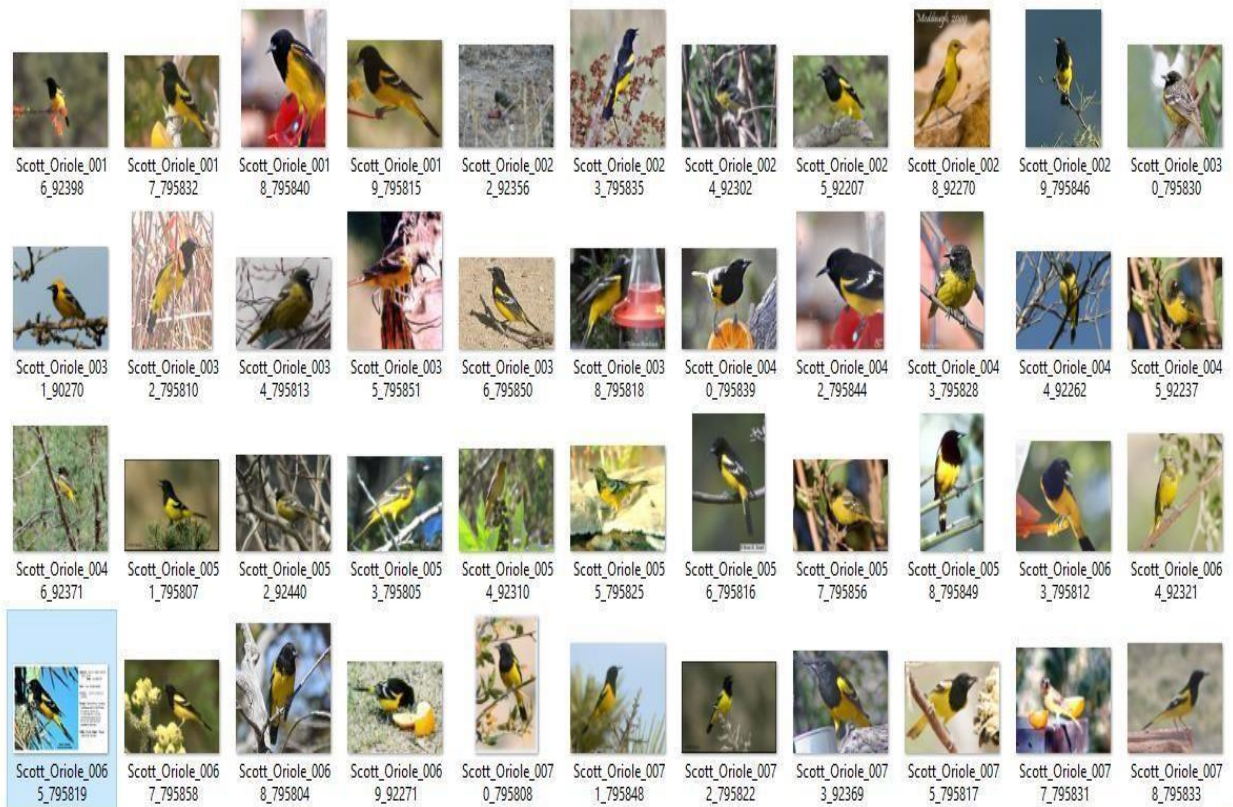


Fig 3.2 Oriole dataset

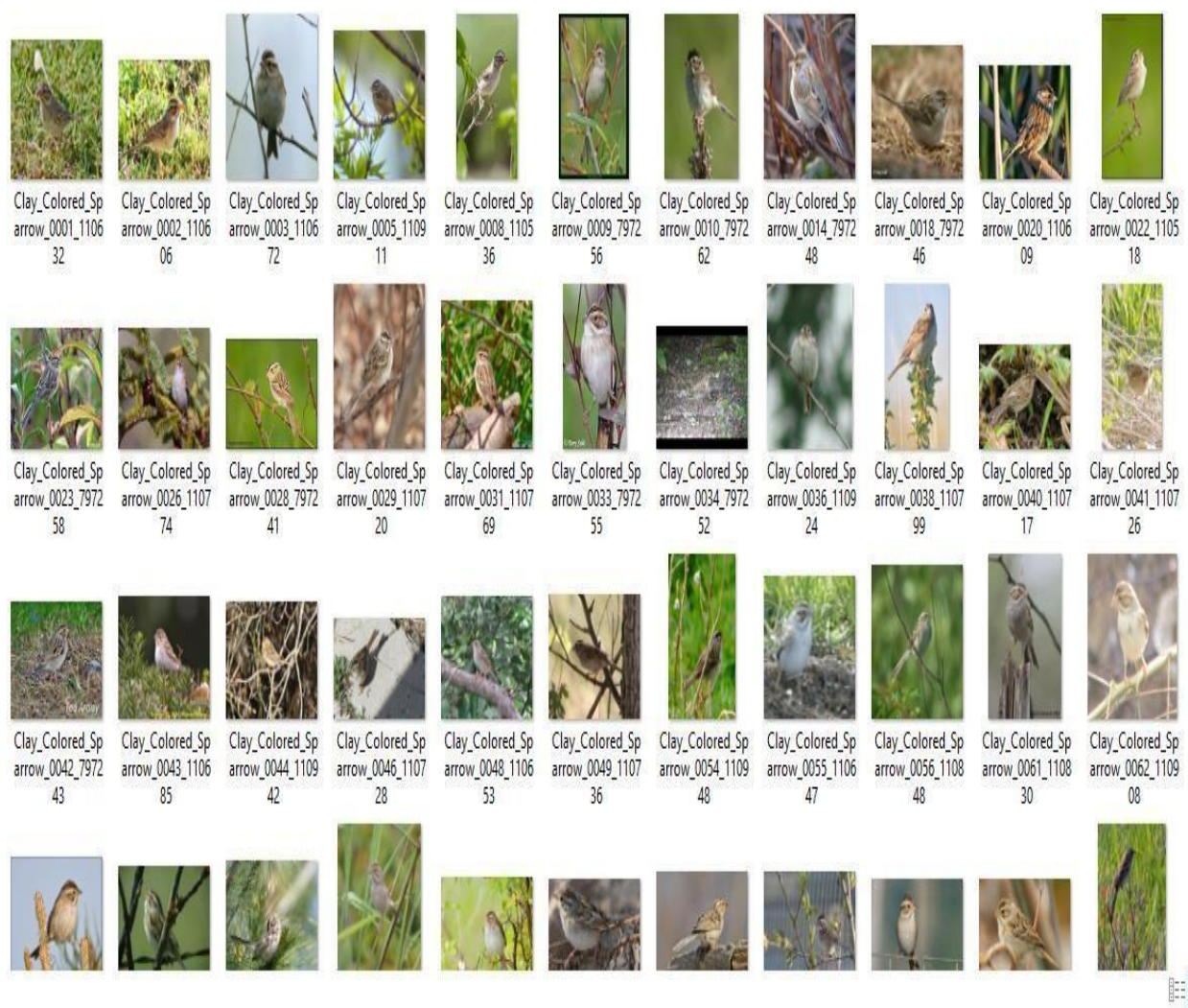


Fig 3.3 Sparrow dataset

3.3 UML DIAGRAM

Unified Modeling Language (UML) diagrams are graphical representations used to visualize and communicate various aspects of software systems and their components. UML provides a standardized notation for modeling software architecture, design, and behavior, making it easier for stakeholders to understand and discuss system requirements, structure, and behavior.

3.3.1 Use case diagram

Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents (actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high-level view of what the system or a part of the system does without going into implementation details.

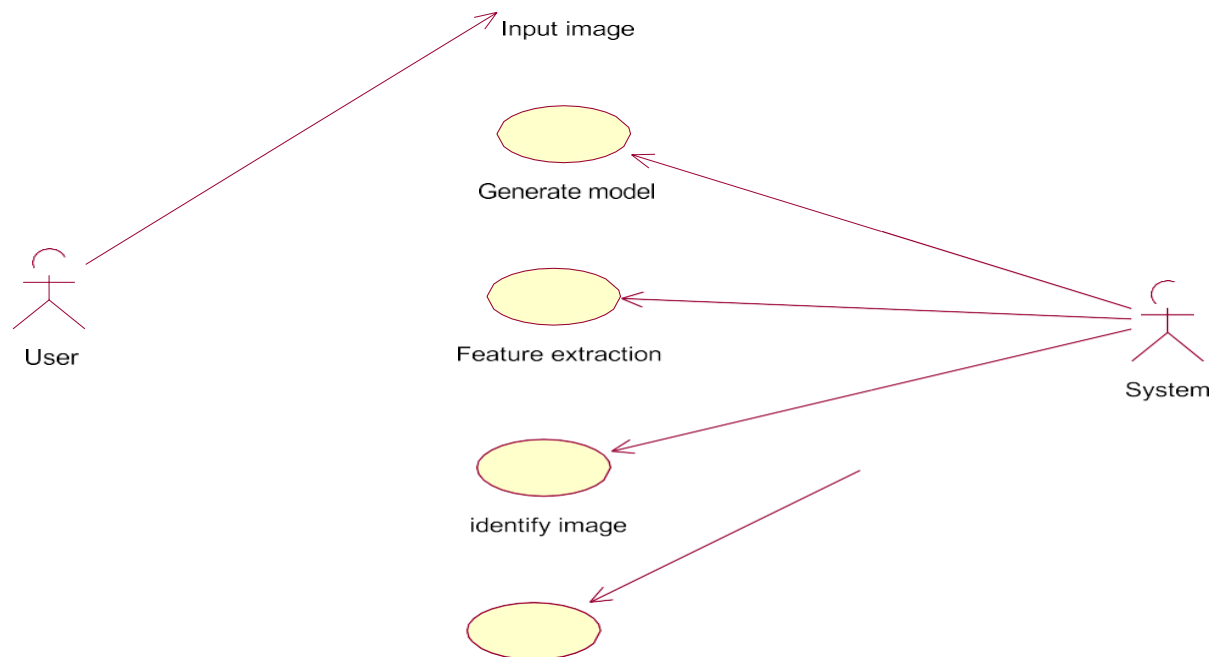


Fig 3.4 Use Case Diagram

3.3.2 Sequence Diagram

A sequence diagram simply depicts the interaction between objects in a sequential order i.e. the order in which these interactions take place. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

A sequence diagram emphasizes the time ordering of messages. A type of interaction diagram, a sequence diagram shows the actor of objects participating in an interaction and the events they generate arranged in a time sequence. Often, a sequence diagram shows the event that results from a particular instance of a use case but a sequence diagram can also exist in a more generic form. Graphically, a sequence diagram is a table that shows object arranged along the X-axis and the messages, ordered in increasing time, along with the Y-axis. The vertical dimension in a sequence diagram represents time, with time proceeding down the page. The horizontal dimension represents different actors or objects.

Sequence diagrams offer several benefits throughout the software development lifecycle. During the design phase, they help developers visualize and refine the interactions between system components, aiding in the identification of potential design flaws or bottlenecks. In the implementation phase, sequence diagrams serve as a blueprint for writing code, providing clear guidance on how different modules or classes should interact with each other. Moreover, sequence diagrams facilitate testing and debugging by providing a structured representation of expected system behavior, enabling developers to validate the correctness and robustness of their implementations. Finally, sequence diagrams are valuable for documentation purposes, serving as a concise and accessible reference for understanding system architecture and behavior.

It is also beneficial to annotate the diagram with relevant comments or notes to provide additional context or explanation where needed. Finally, review and validate the sequence diagram with stakeholders to ensure that it accurately reflects the intended system behavior and meets the requirements of the project.

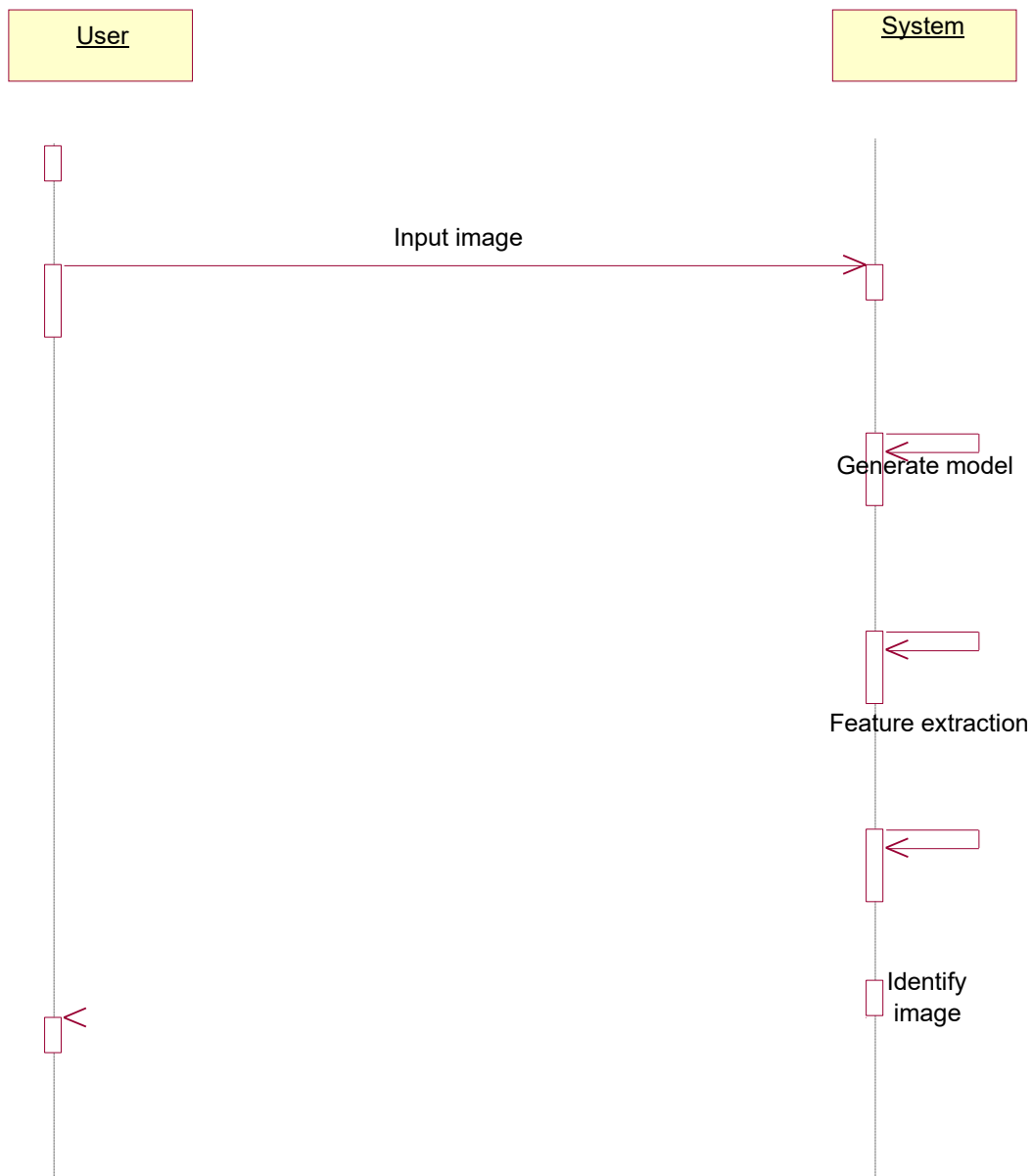


Fig 3.5 Sequence Diagram

3.3.2 Activity Diagram:

An activity diagram is another important diagram in UML to describe the dynamic aspects of the system. An activity diagram is basically a flowchart to represent the flow from one

activityto another activity. The activity can be described as an operation of the system.

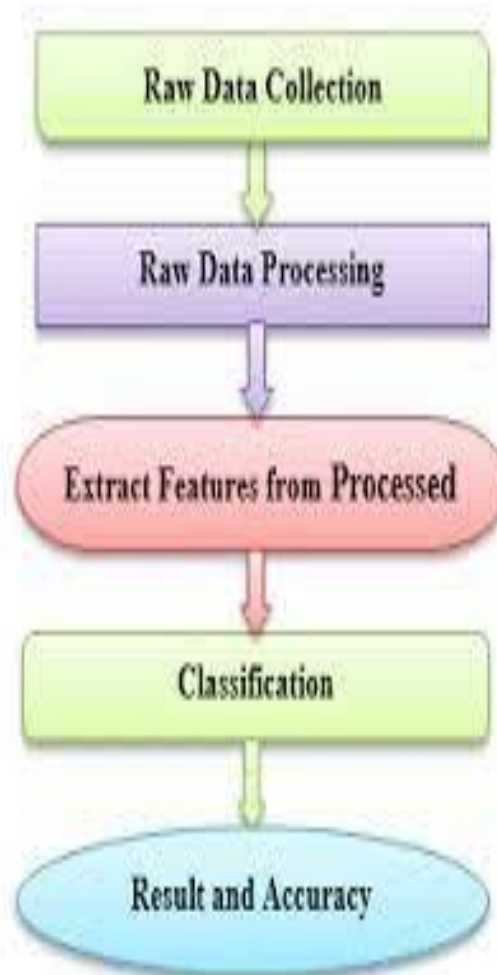


Fig 3.6 Activity Diagram

CHAPTER 4

REQUIREMENT ANALYSIS

4.1 Hardware and Software Requirements

Hardware and software requirements can vary significantly depending on the specific application or system being developed. However, I can provide a general overview of the typical hardware and software requirements for developing and running software applications, including web applications, desktop applications, and mobile applications:

4.1.1 Software Requirements Programming languages:

Python : Python is a high-level, interpreted programming language known for its simplicity, versatility, and readability. Created by Guido van Rossum and first released in 1991, Python has since gained widespread popularity and has become one of the most widely used programming languages in various domains, including web development, data science, artificial intelligence, scientific computing, and more.

Deep learning: Deep learning is a subset of machine learning that utilizes artificial neural networks with multiple layers (hence the term "deep") to model and extract high-level features from complex data. It has gained significant attention and popularity in recent years due to its remarkable performance in various fields, including computer vision, natural language processing, speech recognition, and reinforcement learning.

Deep Learning packages:

- **Tensor Flow :** TensorFlow is an open-source machine learning framework developed by Google Brain, initially released in 2015. It is widely used for building and training deep learning models for various tasks, including image classification, natural language processing, speech recognition, and reinforcement learning. TensorFlow provides a flexible and comprehensive ecosystem for developing machine learning applications, with support for both research and production use cases.

- **Keras :** Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). It was developed with a focus on enabling fast experimentation and prototyping of deep learning models.

4.1.2 Hardware Requirements

- RAM: 4GB
- Hard Disk: 1TB HD

4.2 Software Requirements specifications

Name of the project	Bird species detection
Vision	The vision of the project is to develop a model for identifying bird species using Machine learning.
Users/Actors of the System	Admin User
System Features & Functional Capabilities	The system is trained on a dataset which consists of bird images and their corresponding labels. The system will be able to take an image as input and identify the bird species.
Technologies/Tools to be Used	Deep Learning, Python
Third Party libraries /APIs/Services to be used	Tensor-flow, Streamlit, Keras

Table 4.1 Software Requirements

CHAPTER 5

SYSTEM IMPLEMENTATION

System implementation in the context of machine learning (ML) refers to the stage in the development process where the ML model is integrated into a functional system or application to solve real-world problems. This phase involves deploying the trained model into a production environment where it can make predictions or perform tasks based on new input data.

5.1 Machine Learning:

Machine learning is a field of computer science that gives computer systems the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed. Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly.

5.1.1 Machine learning tasks:

Machine learning tasks are typically classified into two broad categories, depending on whether there is a learning "signal" or "feedback" available to a learning system:

5.1.1.1. Supervised learning

The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. As special cases, the input signal can be only partially available, or restricted to special feedback.

Semi-supervised learning: The computer is given only an incomplete training signal: a training set with some (often many) of the target outputs missing.

Active learning: The computer can only obtain training labels for a limited set of instances (based on a budget), and also has to optimize its choice of objects to acquire labels for. When used interactively, these can be presented to the user for labelling.

Reinforcement learning: Training data (in form of rewards and punishments) is given only as feedback to the program's actions in a dynamic environment, such as driving a vehicle or playing a game against an opponent.

5.1.1.2. Unsupervised learning

No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

5.1.2 Machine Learning Approaches:

Decision tree learning: Decision tree learning uses a decision tree as a predictive model, which maps observations about an item to conclusions about the item's target value.

Association rule learning: Association rule learning is a method for discovering interesting relations between variables in large databases.

Artificial neural networks: An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is vaguely inspired by biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, capture the statistical structure in an unknown joint probability distribution between observed variables.

Deep learning: Deep learning consists of multiple hidden layers in an artificial neural network. This approach tries to model the way the human brain processes light and sound into vision and hearing. Some successful applications of deep learning are computer vision and speech recognition.

5.1.3 Machine Learning process:

Gathering data: Once we have our equipment and booze, it's time for our first real step of machine learning: gathering data. This step is very important because the quality and quantity of data that you gather will directly determine how good your predictive model can be.

5.1.3.1 Data preparation (cleaning data):

We load our data into a suitable place and prepare it for use in our machine learning training. We'll first put all our data together, and then randomize the ordering. We don't want the order of our data to affect what we learn. This is also a good time to do any pertinent visualization of your data, to help you see if there are any relevant relationships between different variables you can take advantage of, as well as show you if there are any data imbalances. We'll also need to split the data into two parts. The first part, used in training our model, will be the majority of the dataset. The second part will be used for evaluating our trained model's performance. Sometimes the data we collect needs other forms of adjusting and manipulation. Things like de-duping, normalization, error correction, and more. These would all happen at the data preparation step.

5.1.3.2 Choosing a model:

The next step in our workflow is choosing a model. There are many models that researchers and data scientists have created over the years. Some are very well suited for image data, others for sequences (like text, or music), some for numerical data, and others for text-based data.

5.1.3.3 Training:

Now we move onto what is often considered the bulk of machine learning the training. In this step, we will use our data to incrementally improve our model's ability to predict. In particular, the formula for a straight line is $y = m \cdot x + b$, where x is the input, m is the slope of that line, b is the y-intercept, and y is the value of the line at the position x . The values we have available to us for adjusting, or "training", are m and b . There is no other way to affect the position of the line since the only other variables are x , our input, and y , our output. In machine learning, there are many m 's since there may be many features. The collection of these m values is usually formed into a matrix, that we will denote W , for the "weights" matrix. Similarly, for b , we arrange them together and call that the biases. The training process involves initializing some random values for W and b and attempting to predict the output with those values. As you might imagine, it does pretty poorly. But we can compare our model's predictions with the output that it should produce, and adjust the values in W and b such that we will have more correct predictions. This process then repeats. Each iteration or cycle of updating the weights and biases is called one training "step".

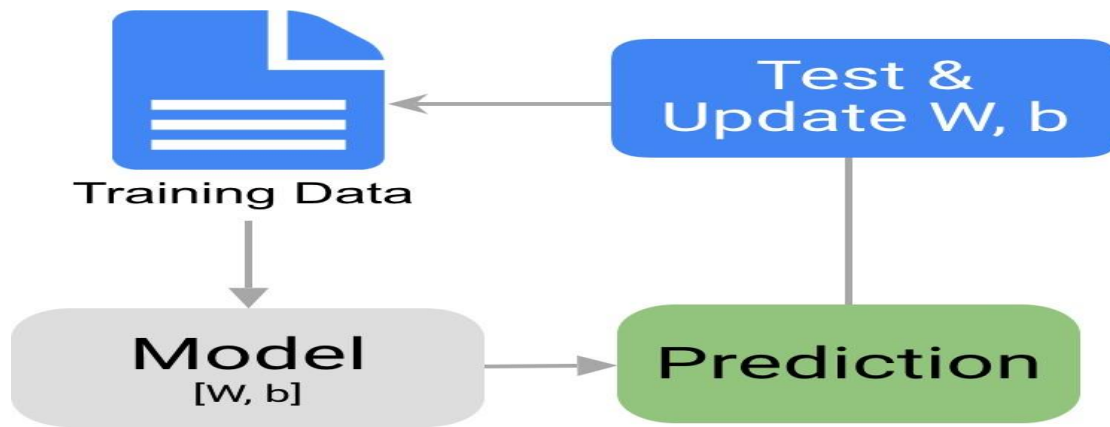


Fig 5.1 Training Chart

5.1.3.4 Evaluation:

Once training is complete, it's time to see if the model is any good, using Evaluation. This is where that dataset that we set aside earlier comes into play. Evaluation allows us to test our model against data that has never been used for training. This metric allows us to see how the model might perform against data that it has not yet seen. This is meant to be representative of how the model might perform in the real world. A good rule of thumb I use for a training-evaluation split somewhere on the order of 80/20 or 70/30. Much of this depends on the size of the original source dataset. If you have a lot of data, perhaps you don't need as big of a fraction for the evaluation dataset.

5.2 Selected Software

5.2.1 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it every attractive for Rapid Application Development., as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python was conceived in the late 1980's, and its implementation began I December 1989 by Guido Van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the amoeba operating system. Van Rossum remains Python's principal author.

His continuing central role in Python's development is reflected in the title given to him by the Python community: Benevolent Dictator For Life (BDFL).

Often, programmers all in love with the Python because of the increased productivity it provides. Since there is no compilation step, the edit-test- debug cycle is incredibly fast. Debugging Python programs is easy: a bug or bad input will never cause a segmentation fault. Instead, when the interpreter discovers an error it raises an exception. When the program does not't catch the exception, the interpreter prints a stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting break points, stepping through the code a line at a time, and so on. The debugger is written in Python itself, testifying to Python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: the fast edit-test- debug cycle makes this simple approach very effective.

The languages core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex .
- Complex is better than complicated.
- Readability counts. Reason for choosing python:
 - Emphasis on code readability, shorter codes, eases of writing.
 - Programmers can express logical concepts in fewer lines of code in comparison to languages such as C++ or Java.
 - Python supports multiple programming paradigms, like object-oriented, imperative and functional programming or procedural.
 - There exist inbuilt functions for almost all of the frequently used concepts.
 - Philosophy is "Simplicity is the best". Industrial Importance: Most of the companies are now looking for candidates who know about Pyt Programming. Those having the knowledge of python may have more chances of impressing the interviewing panel. So I would suggest that beginners should start learning python and excel in it.

Applications of Python:

- A number of Linux distributions use installers written in Python example in Ubuntu we have the Ubiquity
- Python has seen extensive use in the information security industry, including in exploit development.
- Raspberry Pi– single board computer uses Python as its principal user programming language
- . ➤ Python is now being used Game Development areas also.
- The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:
 - Beautiful is better than ugly.
 - Explicit is better than implicit.
 - Simple is better than complex o The complex is better than complicated
 - Readability counts

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuation. Unlike many other languages, it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Whitespaces as Indentation:

Python's syntax is quite easy, but still you have to take some care in writing the code. Indentation is used in writing python codes. Whitespaces before a statement have a significant role and are used in the indentation. Whitespace before a statement can have a different meaning.

Whitespace is mostly ignored, and mostly not required, by the Python interpreter. When it is clear where one token ends and the next one starts, whitespace can be omitted. This is usually the case when special non-alphanumeric characters are involved. Some parts of the standard library are covered by specifications (for example, the Web Server Gateway Interface (WSGI) implementation wsgi ref follows PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However because most of the standard

library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations.

Python is meant to be an easily readable language. Its formatting is visually uncluttered, and it often uses English keywords where other languages use punctuations. Unlike many other languages it does not use curly brackets to delimit blocks, and semicolons after statements are optional. It has fewer syntactic exceptions and special cases than C or Pascal.

Python uses Whitespace indentation, rather than curly brackets or keywords, to delimit blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block. This feature is also sometimes termed the off-side rule. Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (well defined (for example, adding a number to string) rather than silently attempting to make sense of them.

Python's large standard library, commonly cited as one of its greatest strengths, provides tools suited to many tasks. For Internet-facing applications, many standard formats and protocols such as MIME and HTTP are supported. It includes modules for creating graphical user interfaces, connecting to relational databases, generating pseudorandom numbers, arithmetic with arbitrary precision decimals, manipulating regular expressions, and unit testing.

Some parts of the standard library are covered by specifications (for example, the web Server Gateway Interface (WSGI) implementation will follow (PEP 333), but most modules are not. They are specified by their code, internal documentation, and test suites (if supplied). However, because most of the standard library is cross-platform Python code, only a few modules need altering or rewriting for variant implementations. As of March 2018, the Python package Index (PPI), the official repository for third-party Python software, contains over 130,000 packages with a wide range of functionality, including:

- Graphical user interfaces
- Web frameworks
- Multimedia
- Databases
- Networking

- Test frameworks
- Automation
- Web scraping
- Documentation
- System administration
- Scientific computing
- Text processing
- Image processing

5.2.2 Packages using Python

NumPy: It is a library for python programming language, adding support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features the competing Num array into numeric, with extensive modifications. NumPy is open source software and has many contributors.

OS: The OS module in Python provides functions for interacting with the operating system. Os, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The `*OS*` and `*os.path*` modules include many functions to interact with file system.

TensorFlow: TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It allows developers to create large-scale neural networks with many layers. TensorFlow is mainly used for classification, perception, understanding, discovering, prediction and creation.

Flask: Flask is a light weight WSGI web application framework. It is designed to make getting started quick and easy, with the ability to scale up to complex applications. It provides the user with libraries, modules and tools to help build web applications. Flask does however include a web server that can be used for testing and development.

5.2.3 Spyder Editor: Spyder is an open source cross-platform Integrated Development Environment (IDE) for scientific programming in the python language. Spyder integrates with a number of prominent packages in the scientific Python stack, including Numpy, Scipy, Matplotlib, Pandas, IPython, SymPy and Cython, as well as other open source software. Spyder is extensible with first-and third-party plugins, includes support for interactive tools for data inspection and embeds python specific code quality assurance. It is available crossplatform through Anaconda, on windows, on macOS through MacPorts, and on major Linux distributions such as Arch Linux, Debian and Ubuntu. Spyder uses Qt for its GUI, and is designed to use either of the PyQt or PySide Python bindings. QtPy, a thin abstraction layer developed by the Spyder project and later adapted by multiple other packages, provides the flexibility to use either backend.

Features: Features include:

- An editor with syntax highlighting, introspection, code completion.
- Support for multiple IPython consoles.
- The ability to explore and edit variables from a GUI.
- A Help pane able to retrieve and render rich text documentation on functions, classes and methods automatically or on demand
- . • A debugger linked to IPdb, for step-by-step execution
- . • Static code analysis, powered by Pylint
- . • A run-time profiler, to benchmark code.
- Project support, allowing work on multiple development efforts simultaneously.
- A built-in file explorer, for interacting with the file system and managing file projects.
- A “Find in Files” feature, allowing full regular expression search over a specified scope.
- An online help browser, allowing users to search and view Python and package documentation inside the IDE.
- A history log, recording every user command entered in each console.
- An internal console, allowing for introspection and control over spyder’s own operation.

Core components:

1. Editor: Work efficiently in a multi-language editor with a function/class browser, real-time code analysis tools, automatic code completion, horizontal/vertical splitting, and goto-definition.

2. Interactive console: Harness the power of as many IPython consoles as you like with full work space and debugging support, all within the flexibility of a full GUI interface. Instantly run your code by line, cell, or file, and render plots right inline with the output or in interactive inverse.

3. Documentation viewer: Render documentation in real-time with Sphinx for any class or function, whether external or user-created, from either the Editor or a Console.

4. Variable Explorer: Inspect any variables, functions or objects created during your session. Editing and interaction is supported with many common types, including numeric/strings/booleans, Python lists/tuples/dictionaries, dates/time deltas, Numpy arrays, Pandas index/series/data frames, PIL/Pillow images, and more.

5. Development Tools: Examine your code with the static analyser, trace its execution with the interactive debugger, and unleash its performance with the profiler. Keep things organized with project support and a built-in file explorer, and use find in files to search across entire projects with full regex support.

5.3 Modules

5.3.1 Pre-processing:

Deficiencies in the data acquisition process due to limitations of the capturing device sensor, or to prepare the data for subsequent activities later in the description or classification stage. Data pre-processing describes any type of processing performed on raw data to prepare it for another processing procedure. Hence, preprocessing is the preliminary step which transforms the data into a format that will be more easily and effectively processed. Therefore, the main task in pre-processing the capture data is to decrease the variation that causes the reduction in the recognition rate and increases the complexities. Thus, pre-processing is an essential stage prior to feature extraction since it controls the suitability of the results for the successive stages.

5.3.2 Classification:

A neural network is set of connected input/output units in which each connection has a weight associated with it. During the learning phase, the networks learns by adjusting the weights so as to be able to predict the correct class label of the input tuples. Neural network learning is also referred to as connectionist learning due to the connections between units. Advantages of neural networks, however, include their high tolerance of noisy data as well as their ability to

classify patterns on which they have not been trained. They have been successful on a wide array of real-world data, including Malaria Detection, Pathology and laboratory medicine and training a computer to pronounce English text. There are many different kinds of neural networks and neural networks algorithms. The most popular neural network is back propagation.

5.3.3 A Multilayer Feed-Forward Neural Network:

A multilayer feed-forward neural network consists of an input layer, one or more hidden layers, and an output layer. Each layer is made up of units. The inputs to the network correspond to the attributes measured for each training tuple. The units are fed simultaneously into the units making up the input layer. These inputs pass through the input layer and are then weighted and fed simultaneously to a second layer of “neuron like” units, known as a hidden layer. The outputs of the hidden layer units can be input to another hidden layer, and so on. The number of hidden layers is arbitrary, although in practise, usually only one is used. The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network’s prediction for given tuples. The units in the input layer are called input units. The units in the hidden layer and output layer are sometimes referred to as neurodes, due to their symbolic biological basis, or as output units. The multilayer neural network has two layers of output units. Therefore, we say that it as a two-layer neural network. (The input layer is not counted because it serves only to pass the unit values to the next and so on. The network is feed-forward in that none of the weight cycles back to an input unit or to an output unit of a previous layer. It is fully connected in that each unit provides input to each unit in the next forward layer.

5.3.4 Convolutional Neural Network (CNN): The classifier presented by Sprengel et is a convolutional neural network (CNN). A convolutional neural network is a neural network architecture pioneered by LeCun et, and later popularized by Krizhevsky et with the introduction of the AlexNet. A typical CNN consists of convolutional layers and pooling layers followed by a fully connected neural network. The convolutional layers and the pooling layers are supposed to learn how to extract relevant, locally distortion invariant, features from the input, and the fully connected neural network is supposed to learn how to classify these features. The features learned by the convolutional layers could be, e.g., edges of objects in an image. In the rest of this section, convolutional layers and pooling layers will be defined, the input of the network will be summarized, the architecture used in the baseline will be defined, and the initialization, optimization and loss function used will be explained.

- **Architecture:** All CNN models a similar structure.

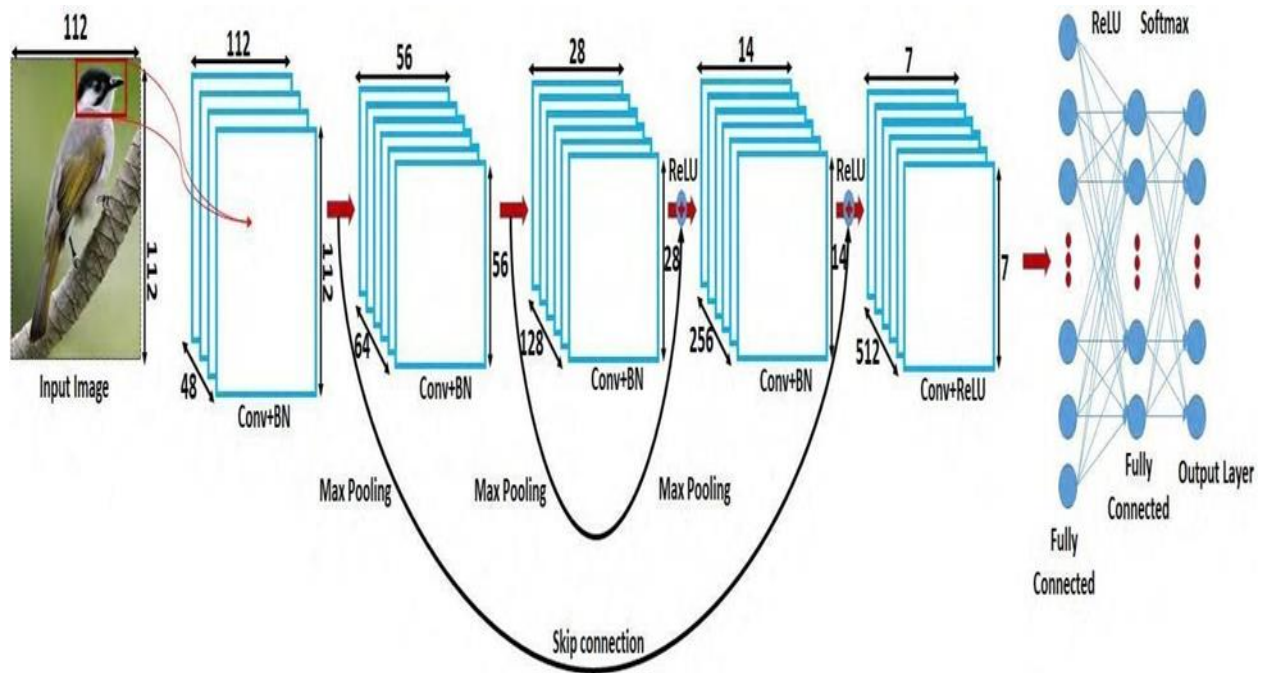


Fig 5.2 CNN Architecture

There is an input image that we're working with. We perform a series convolution + pooling operations, followed by a number of fully connected layers. If we are performing multiclass classification the output is SoftMax. We will now dive into each component.

1. **Convolutional layer:** A convolutional layer takes an image as input, and produces a set of feature maps as output. The input image can contain multiple channels (e.g. RGB), which means that the convolutional layer learns a mapping from a 3D volume to another 3D volume. The layer consists of a number of convolution kernels each of which is made up of adjustable weights. The weights are adjusted during optimization using stochastic gradient decent. The feature maps are produced by performing a discrete convolution between each kernel in the convolutional layer and the input volume yielding one feature map for each kernel. Performing a convolution can be thought of as sliding a window along the input image and computing the dot product between the kernel and the values of the neighbourhood patch, or the so called receptive field, in the image as seen through the

window. The window has the same size as the kernel, and it is important to understand that the kernel has a depth extending through all the channels of the input image.

Formally, $w_{l,k}$ and $b_{l,k}$ denote the weights and the bias term for the l -th layer of the k -th convolution kernel, and let $x_l \in \mathbb{R}^{W \times H \times D}$ be the l -th channel, or layer, of an input image of shape (W, H, D) . We can then define the feature value, $c_{k,i,j}$, at position (i, j) of the resulting k -th feature map as:

$$c_{k,i,j} = f\left(\sum_{l=1}^L \sum_{d=1}^D w_{l,k}^{(d)} x_{l,i,j}^{(d)} + b_{l,k}\right)$$

Where D is the depth of the input (the number of channels), $x_{l,i,j}$ is the receptive field centered around (i, j) of the l -th layer and f is a rectified linear unit (ReLU) defined as:

$$f(x) = \max(0, x).$$

Note that the resulting number of feature maps is the same as the number of kernels used, and that the weights and bias are reused in the computation of the dot product for each receptive field, for each kernel and layer, called weight sharing. The idea is that if the kernel is useful for finding local features at position (x_1, y_1) in the image, then it should also be useful at a different position (x_2, y_2) .

The configuration of the convolutional layer is thus specified by the number of kernels used, the width and height of the kernels, and the stride size which determines the spacing between the receptive fields. For example, a stride size of 1×2 would slide the window along the image with a step size of one pixel horizontally and two pixels vertically making the width of the resulting feature map the same as that of the input image, but the height of the feature map half the height of the input image.

The output size of the convolutional layer is therefore determined by the number of kernels used and the stride size. The number of kernels determines the depth of the output volume and the stride size determines the width and height of the output volume. For example, if the input volume of a convolutional layer is an image of shape $(32, 32, 3)$, the number of kernels used is 16, and the stride size is 1×2 , then the output volume would be of shape $(32, 16, 16)$.

Zero padding of the input image has been assumed, window will extend outside the bounds of the input at the edges some values will not be defined, and zero padding is basically assuming that all these values are zero.

Padding:

Sometimes filter does not perfectly fit the input image. We have two inputs:

- Pad the picture with zeros (zero-padding) so that it fits.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

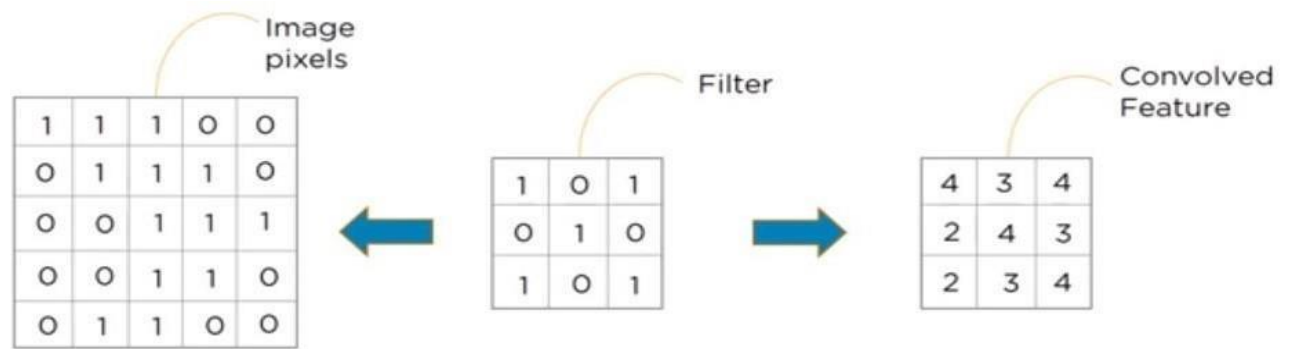


Fig 5.3 Padding

Activation Function: Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

- RELU • Soft max

1.Non-Linearity (RELU layer):

RELU: - Stands for Rectified linear unit. It is the most widely used activation function. Chiefly implemented

in hidden layers of Neural network.

- **Equation:** - $A(x) = \max(0, x)$

. It gives an output x if x is positive and 0 otherwise.

- **Value Range** :- $[0, \infty)$

- **Nature:** - non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.

- **Uses:** - ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation

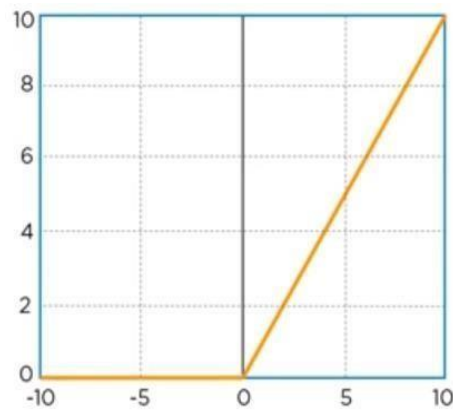


Fig 5.4 Activation Function Chart

Advantages of RELU:

- Computational Simplicity
- Representational Sparsity
- Linear Behaviour
- Train Deep Networks

Soft max Function:- The soft max function is also a type of sigmoid function but is handy when we are trying to handle classification problems.

- **Nature:** - non-linear
- **Uses:** - Usually used when trying to handle multiple classes. The soft max function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.

- **Output:** - The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.

2.Pooling layer:

The feature maps computed by a convolutional layer are then run through a pooling layer, which is designed to merge semantically similar features and thus reduce the size of the feature maps.

The pooling layer used is max pooling which simply computes the maximum value of local receptive fields in each of the feature maps. The spacing of the fields is determined by the stride size and the size of the fields is determined by the kernel size of the pooling layer. Again, let x_l be the receptive field of the l -th input layer centred around (i, j) , then the value of the l -th pooled layer, p_l , can be defined as:

$$p_l = \max (x_l)_{i,j}$$

Where max takes the maximum value of the receptive field.

For example, if the stride size is 2×2 then the width and the height of the pooled feature maps are halved because we step through the 2×2 neighbourhoods of each feature map with a step size of 2 pixels horizontally, and 2 pixels vertically. That is, if the input of a max pooling layer with a kernel of size 2×2 and a stride size of 2×2 has shape (32, 32, 4), then the output will have shape (16, 16, 4). The input is assumed to be zero padded to handle edge cases.

In pooling layer, it will reduce number of parameters when image is too large. Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified map. Average pooling takes the average from largest element. Sum of all elements in the feature map is known as sum pooling.

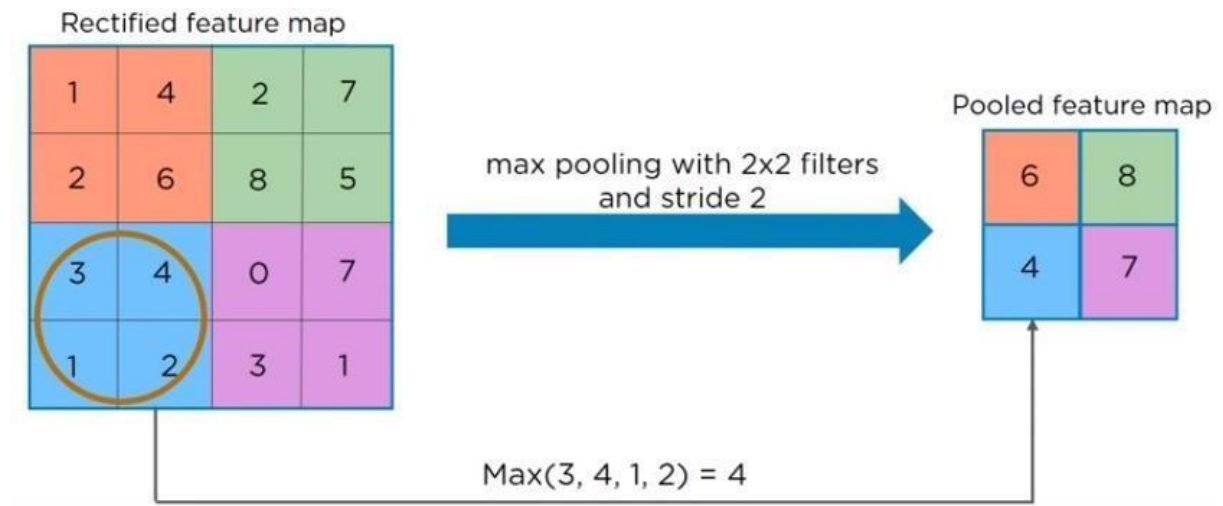


Fig 5.5 Pooling Layer Structure

Hyper parameters:

- **Filter size:** we typically use 3x3 filters, but 5x5 or 7x7 are also used depending on the application. There are also 1x1 filters which we will explore in another article, at first sight it might look strange but they have interesting applications. Remember that these filters are 3D and have a depth dimension as well, but since the depth of a filter at a given layer is equal to the depth of its input, we omit that.

- **Filter count:** this is the most variable parameter; it's a power of two anywhere between 32 and 1024. Using more filters results in a more powerful model, but we risk overfitting due to increased parameter count. Usually we start with a small number of filters at the initial layers, and progressively increase the count as we go deeper into the network.

- **Stride:** we keep it at the default value

- **Padding:** we usually use padding

- **Fully connected layer:** Fully connected layer is the last layer in CNN. That identifies object in the image and predicts the output. We flattened out, matrix into vector and feed it into fully connected layer like neural network.

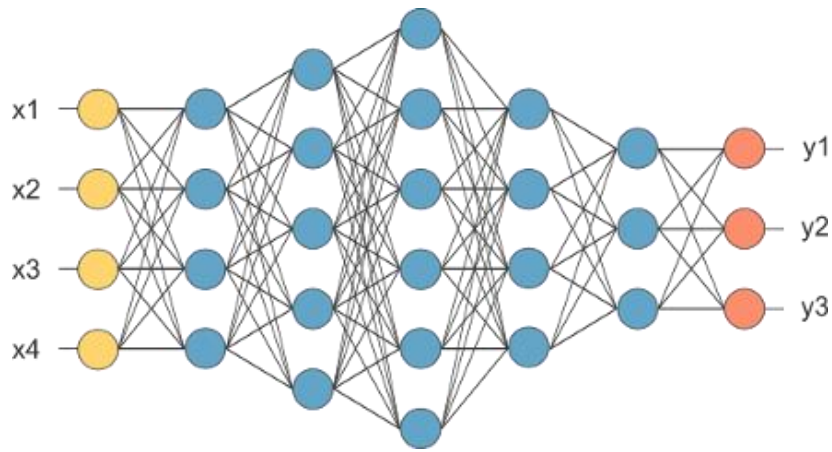


Fig 5.6 Dense Layer Structure

The feature map matrix will be converted as vector (x_1, x_2, x_3, \dots). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as soft max or sigmoid to classify the outputs as cat, dog, car, truck etc.,

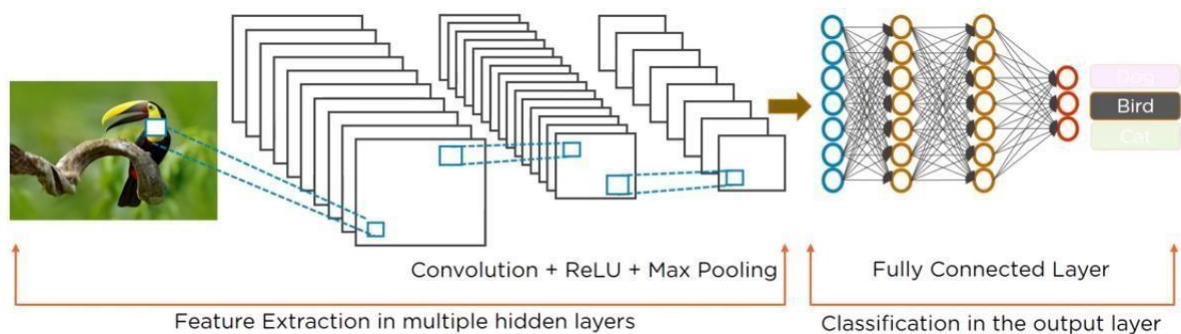


Fig 5.7 Components of CNN

- **Intuition:**

A CNN model can be thought as a combination of two components: feature extraction part and the classification part. The convolution + pooling layers perform feature extraction. For example given an image, the convolution layer detects features such as two eyes, long ears, four legs, a short tail and so on. The fully connected layers then act as a classifier on top of these features, and assign a probability for the input image being a dog.

The convolution layers are the main powerhouse of a CNN model. Automatically detecting meaningful features given only an image and a label is not an easy task. The convolution layers

learn such complex features by building on top of each other. The first layers detect edges, the next layers combine them to detect shapes, to following layers merge this information to infer that this is a nose. To be clear, the CNN doesn't know what a nose is. By seeing a lot of them in images, it learns to detect that as a feature. The fully connected layers learn how to use these features produced by convolutions in order to correctly classify the images.

All this might sound vague right now, but hopefully the visualization section will make everything more clearly.

- **Implementation:** After this lengthy explanation let's code up our CNN. We will use the Dogs vs Cats dataset from Kaggle to distinguish dog photos from cats. We will use the following architecture: 4 convolutions + pooling layers, followed by 2 fully connected layers. The input is an image of a cat or dog and the output is binary.

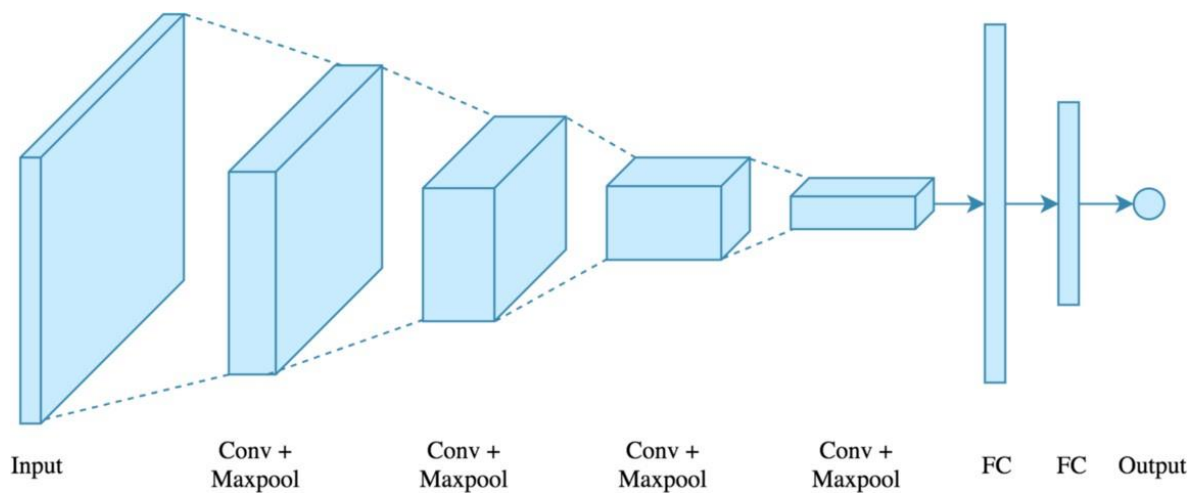


Fig 5.8 Pooling Layer

Code for CNN:

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(300, 300, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.1),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
model.summary()

model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.RMSprop(lr=0.001),
              metrics=['acc']) #RMSprop(lr=0.001)Adam(lr=0.4)
total_sample=train_generator.n
```

Conv2D: This method creates a convolutional layer. The first parameter is the filter count, and the second one is the filter size. For example in the first convolution layer we create 32 filters of size 3x3. We use relu non-linearity as activation. We also enable padding. In Keras there are two options for padding: same or valid. Same means we pad with the number on the edge and valid means no padding. Stride is 1 for convolution layers by default so we don't change that. This layer can be customized further with additional parameters.

MaxPooling2D: creates a max-pooling layer, the only argument is the window size. We use a 2x2 windows as it's the most common. By default stride length is equal to the window size. Flatten: After the convolution+ pooling layers we flatten their output to feed into the fully connected layers.

Dropout: Dropout is by far the most popular regularization technique for deep neural networks. Even the state-of-the-art models which have 95% accuracy get a 2% accuracy boost just by adding dropout, which is a fairly substantial gain at that level. Dropout is used to prevent overfitting and the idea is very simple. During training time, at each iteration, a neuron is temporarily "dropped" or disabled with probability p. This means all the inputs and outputs to this neuron will be disabled at the current iteration. The dropped neurons are resampled with probability p at every training step, so a dropped out neuron at one step can be active at the next

one. The hyperparameter p is called the dropout-rate and it's typically a number around 0.5, corresponding to 50% of the neurons being dropped out.

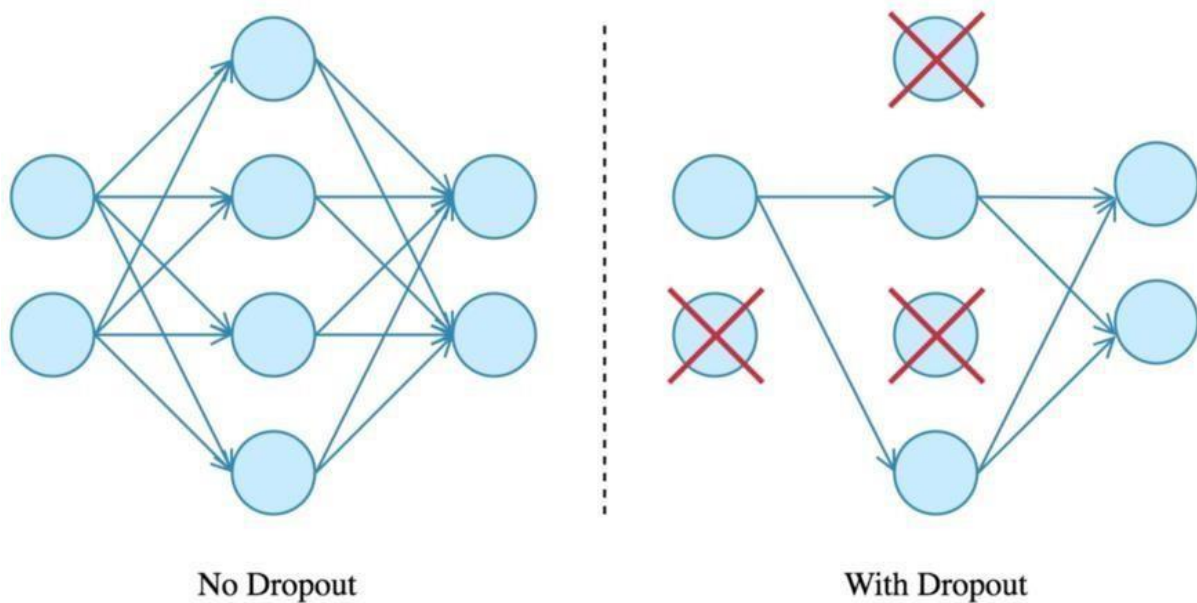


Fig 5.9 Drop out Technique

- **Data Augmentation:** Overfitting happens because of having too few examples to train on, resulting in a model that has poor generalization performance. If we had infinite training data, we wouldn't overfit. Data augmentation is done dynamically during training time. We need to generate realistic images, and the transformations should be learnable, simply adding noise won't help. Common transformations are: rotation, shifting, resizing, exposure adjustment, contrast change etc. This way we can generate a lot of new samples from a single training example. Also, data augmentation is only performed on the training data, we don't touch the validation or test set.

- **Visualization:** Visualization of convolutional neural networks. Deep learning models are known to be very hard to interpret, that's why they are usually treated as black boxes.

- o Feature maps
- o Convnet filters
- o Class output

CHAPTER 6

TESTING

6.1 Introduction

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use. Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- Meets the requirements that guided its design and development.
- responds correctly to all kinds of inputs
- performs its functions within an acceptable time
- it is sufficiently usable
- can be installed and run in its intended environments, and
- Achieves the general result its stakeholder's desire.

6.1.1 Types of Tests

➤ **Unit testing:** Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors. These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other. Unit testing is a software development process that involves a synchronized application of a broad spectrum of defect prevention and detection strategies in

order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development life cycle. Unit testing aims to eliminate construction errors before code is promoted to additional testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

➤ **Integration testing:** Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together. Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed. Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

➤ **Functional testing:** Functional testing refers to activities that verify a specific action or function of the code. These are usually found in the code requirements documentation, although some development methodologies work from use cases or user stories. Functional tests tend to answer the question of "can the user do this" or "does this particular feature work." 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 centred 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.

➤ **System Testing:** System testing tests a completely integrated system to verify that the system meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

➤ **Performance Testing:** Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

➤ **Acceptance Testing:** Acceptance testing can mean one of two things: o A smoke test is used as a build acceptance test prior to further testing, e.g., before integration or regression. o Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

6.1.2 Testing Methods

There are different methods that can be used for software testing. Here briefly describes the methods available. **1. Static Testing:** Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application. Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that cannot be found using Dynamic Testing can be easily found by Static Testing. Static testing is of four types:

a. Informal: In informal review the creator of the documents put the contents in front of audience and everyone gives their opinion and thus defects are identified in the early stage.

b. Walkthrough: It is basically performed by experienced person or expert to check the defects so that there might not be problem further in the development or testing phase.

c. Peer review: Peer review means checking documents of one-another to detect and fix the defects. It is basically done in a team of colleagues. **d. Inspection:** Inspection is basically the verification of document the higher authority like the verification of software requirement specifications (SRS).

2. Dynamic Testing: Dynamic Testing is a type of Software Testing which is performed to analyse the dynamic behaviour of the code. It includes the testing of the software for the input values and output values that are analysed.

- **Black-Box Testing:** The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

Techniques of Black Box Testing: The following are the techniques employed while using Black box testing for a software application.

1. BVA or Boundary Value Analysis: It is one among the useful and critical Black box testing technique that helps in equivalence partitioning. BVA helps in testing any software having a boundary or extreme values. This technique is capable of identifying the flaws of the limits of the input values rather than focusing on the range of input value. Boundary Value Analysis also deals with edge or extreme output values.

Equivalence Class Partitioning: This technique of Black box testing is widely used to write test cases. It can be useful in reducing a broad set of possible inputs to smaller but effective ones. It is performed through the division of inputs as classes, and each class is given a value. It is applied when the need for exhaustive testing arises and for resisting the redundancy of inputs.

• **White-Box Testing:** The box testing approach of software testing consists of black box testing and white box testing. We are discussing here white box testing which also known as glass box is testing, structural testing, clear box testing, open box testing and transparent box testing. It tests internal coding and infrastructure of a software focus on checking of predefined inputs against expected and desired outputs. It is based on inner workings of an application and revolves around internal structure testing. In this type of testing programming skills are required to design test cases. The primary goal of white box testing is to focus on the flow of inputs and outputs through the software and strengthening the security of the software.

The term 'white box' is used because of the internal perspective of the system. The clear box or white box or transparent box name denote the ability to see through the software's outer shell into its inner workings. Test cases for white box testing are derived from the design phase of the software development lifecycle. Data flow testing, control flow testing, path testing, branch testing, statement and decision coverage all these techniques used by white box testing as a guideline to create an error-free software.

White-Box Testing Techniques:

- 1) Data Flow Testing
- 2) Control Flow Testing
- 3) Branch Coverage Testing

4) Statement Coverage Testing

5) Decision Coverage Testing

Data Flow Testing: Data flow testing is used to analyze the flow of data in the program. It is the process of collecting information about how the variables flow the data in the program. It tries to obtain particular information of each particular point in the process. Data flow testing is a group of testing strategies to examine the control flow of programs in order to explore the sequence of variables according to the sequence of events. It mainly focuses on the points at which values assigned to the variables the point at which these values are used by concentrating on both points, data flow can be tested.

Control Flow Testing: Control flow testing is a testing technique that comes under white box testing. The aim of this technique is to determine the execution order of statements or instructions of the program through a control structure. The control structure of a program is used to develop a test case for the program. In this technique, a particular part of a large program is selected by the tester to set the testing path. It is mostly used in unit testing. Test cases represented by the control graph of the program. Control Flow Graph is formed from the node, edge, decision node, junction node to specify all possible execution paths.

Branch Coverage Testing: Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once. Branch coverage technique is a white box testing technique that ensures that every branch of each decision point must be executed. However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two. Decision coverage technique covers all branches of each decision point whereas branch testing covers all branches of every decision point of the code.

Statement Coverage Testing: Statement coverage is one of the widely used software testing. It comes under white box testing. Statement coverage technique is used to design white box test cases. This technique involves execution of all statements of the source code at least once. It is used to calculate the total number of executed statements in the source code out of total statements present in the source code. Statement coverage derives scenario of test cases under the white box testing process which is based upon the structure of the code.

Decision Coverage Testing: Decision coverage technique comes under white box testing which gives decision coverage to Boolean values. This technique reports true and false outcomes of Boolean expressions. Whenever there is a possibility of two or more outcomes from the

statements like do while statement, if statement and case statement (Control flow statements), it is considered as decision point because there are two outcomes either true or false. Decision coverage covers all possible outcomes of each and every Boolean condition of the code by using control flow graph or chart.

GreyBox Testing: Greybox testing is a software testing method to test the software application with partial knowledge of the internal working structure. It is a combination of black box and white box testing because it involves access to internal coding to design test cases as white box testing and testing practices are done at functionality level as black box testing. GreyBox testing commonly identifies context-specific errors that belong to web systems.

For example; while testing, if tester encounters any defect then he makes changes in code to resolve the defect and then test it again in real time. It concentrates on all the layers of any complex software system to increase testing coverage. It gives the ability to test both presentation layer as well as internal coding structure. It is primarily used in integration testing and penetration testing.





6.2 Need For Testing :

Testing was essential for the following reasons :- •

Existence of program detects of inadequacies.

- A software behaviour as intended by its designer.
- Conformance with requirement s specification/user needs.
- Assess the operational reliability of the system.
- Reflect the frequency of actual user inputs.
- Find the faults, which causes the output anomaly.
- Checks for detects flaws and deficiencies in the requirements.
- Checks whether the software is operationally useful.
- Excercise the program using data like the real data processed by the program.

6.3 Test Cases

Test Case Id	Input	Actual Output	Expected Output	Status
TC01		Oriole	Oriole	Pass
TC02		Gull	Gull	Pass
TC03		Sparrow	Sparrow	Pass
TC04		Oriole	Parrot	Fail




TC05		Oriole	Oriole	Pass
TC06		Gull	Gull	Pass
TC07		Sparrow	Sparrow	Pass

Table 6.1 Test Case Table

6.4 Software Testing

Software testing is a critical phase in the software development lifecycle, aimed at identifying defects and ensuring the quality and reliability of the final product. It involves the execution of software components or systems under controlled conditions to evaluate their functionality against specified requirements. Various testing techniques, such as unit testing, integration testing, system testing, and acceptance testing, are employed throughout the development process to detect errors and inconsistencies at different levels of granularity.

Unit testing focuses on testing individual units or components of the software in isolation, typically using automated testing frameworks. This allows developers to verify the correctness of each unit's behavior and detect defects early in the development cycle. Integration testing, on the other hand, evaluates the interactions between different units or modules to ensure they function correctly when combined. System testing assesses the behavior of the entire software system as a whole, validating its compliance with functional and non-functional requirements. Finally, acceptance testing involves evaluating the software's conformity to user expectations and business needs, often performed by end-users or stakeholders.

Effective software testing requires careful planning, resource allocation, and the adoption of appropriate testing methodologies and tools. Test cases must be designed to cover a wide range of scenarios, including normal usage, boundary cases, and error conditions, to maximize test coverage and detect potential defects. Moreover, testing activities should be iterative and incremental, allowing for continuous feedback and refinement throughout the development process.

Software testing is a dynamic and evolving discipline that encompasses a wide range of techniques, methodologies, and tools aimed at verifying and validating software systems. The primary objectives of testing are to uncover defects, assess the software's compliance with specified requirements, and ensure its functionality, reliability, and performance meet user expectations. To achieve these goals, testers employ a variety of testing strategies, including functional testing, non-functional testing, and regression testing.

Functional testing focuses on verifying that the software behaves as intended, performing the functions specified in the requirements documentation. This may involve executing test cases to validate individual features, user interactions, and business processes, ensuring they produce the expected outcomes. Non-functional testing, on the other hand, evaluates aspects such as performance, scalability, usability, and security, which are essential for delivering a satisfactory user experience. Regression testing is another critical aspect of software testing, involving the re-execution of previously executed test cases to ensure that recent changes or enhancements have not introduced new defects or adversely affected existing functionality.

In addition to traditional testing techniques, modern software development practices, such as agile and DevOps, have introduced new approaches to testing, emphasizing collaboration, automation, and continuous integration. Test automation, in particular, has become increasingly prevalent, enabling testers to streamline repetitive tasks, increase test coverage, and expedite the feedback loop. By leveraging automated testing frameworks and tools, organizations can accelerate the

testing process, improve the accuracy and repeatability of tests, and enhance overall software quality.

However, despite the advancements in testing methodologies and tools, software testing remains a challenging and resource-intensive endeavor. Testers must continually adapt to evolving technologies, requirements, and user expectations, while also addressing constraints such as time, budget, and resource availability. Moreover, the complexity of modern software systems, coupled with the increasing interconnectedness of digital ecosystems, poses unique challenges for testers, requiring them to adopt innovative approaches and techniques to ensure comprehensive test coverage and

6.5 Testing Strategies

Testing strategies are systematic approaches adopted by software testers to ensure the thorough verification and validation of software systems. A robust testing strategy encompasses various techniques, methodologies, and tools tailored to the specific characteristics of the software under test. Here, we explore two fundamental testing strategies: black box testing and white box testing.

Black Box Testing:

Black box testing, also known as functional testing, focuses on evaluating the external behavior of the software system without considering its internal structure or implementation details. Testers view the software as a black box, interacting with it solely through its inputs and observing its outputs. This approach emphasizes testing the software against specified functional requirements, validating its functionality, usability, and conformance to user expectations.

There are several techniques commonly employed in black box testing, including:

Equivalence Partitioning: This technique divides the input domain into equivalence classes, ensuring that test cases are representative of each class. By selecting a single test case from each equivalence class, testers can achieve maximum test coverage with minimal redundancy.

Boundary Value Analysis: Boundary value analysis involves testing the boundaries of input ranges, focusing on values at the lower and upper limits, as well as just beyond these limits. This helps uncover potential off-by-one errors, boundary-related defects, and boundary condition violations.

Decision Table Testing: Decision tables are used to model complex business rules or logic, facilitating the generation of test cases based on different combinations of inputs and corresponding expected outcomes. This technique ensures comprehensive coverage of all possible decision paths within the software.

State Transition Testing: State transition testing is particularly applicable to software systems with finite states and transitions, such as state machines or finite automata. Test cases are designed to exercise transitions between different states, verifying the correctness of state transitions and the system's overall behavior.

Black box testing is beneficial for identifying defects related to incorrect functionality, missing features, and usability issues. It enables testers to assess the software's behavior from an end-user perspective, ensuring that it meets user requirements and expectations. However, black box testing may not uncover defects related to internal logic errors, code structure, or performance bottlenecks, which are better addressed through white box testing.

White Box Testing:

White box testing, also known as structural testing or glass box testing, focuses on assessing the internal structure and logic of the software system. Testers have access to the source code and design documents, allowing them to design test cases based on the internal workings of the software. This approach aims to ensure thorough coverage of code paths, statements, branches, and conditions, revealing defects related to code quality, logic errors, and security vulnerabilities.

Several techniques commonly employed in white box testing include:

Statement Coverage: This technique measures the percentage of executable statements in the source code that are exercised by a set of test cases. Testers aim to achieve 100% statement coverage, ensuring that every line of code is executed at least once during testing.

Branch Coverage: Branch coverage assesses the percentage of decision branches in the source code that are traversed by the test cases. Testers design test cases to cover both true and false branches of conditional statements, ensuring that all possible outcomes are evaluated.

Path Coverage: Path coverage aims to execute every possible path through the control flow graph of the software, ensuring that all feasible execution paths are tested. This technique provides a more thorough assessment of code behavior than statement or branch coverage alone but may be impractical for complex software systems due to the exponential growth of possible paths.

Code Reviews: Code reviews involve systematic examination of source code by peers or team members to identify defects, code smells, and opportunities for improvement. Code reviews can uncover issues that may not be apparent during testing, such as architectural flaws, coding standards violations, and performance inefficiencies.

White box testing complements black box testing by providing deeper insight into the internal workings of the software and uncovering defects that may not be apparent through external observation alone. It is particularly effective for identifying defects related to code quality, logic errors, and security vulnerabilities. However, white box testing requires access to the source code and a good understanding of software architecture and design, making it less accessible to testers without programming expertise.

CHAPTER 7

SCREENS&REPORTS

7.1 SCREENS

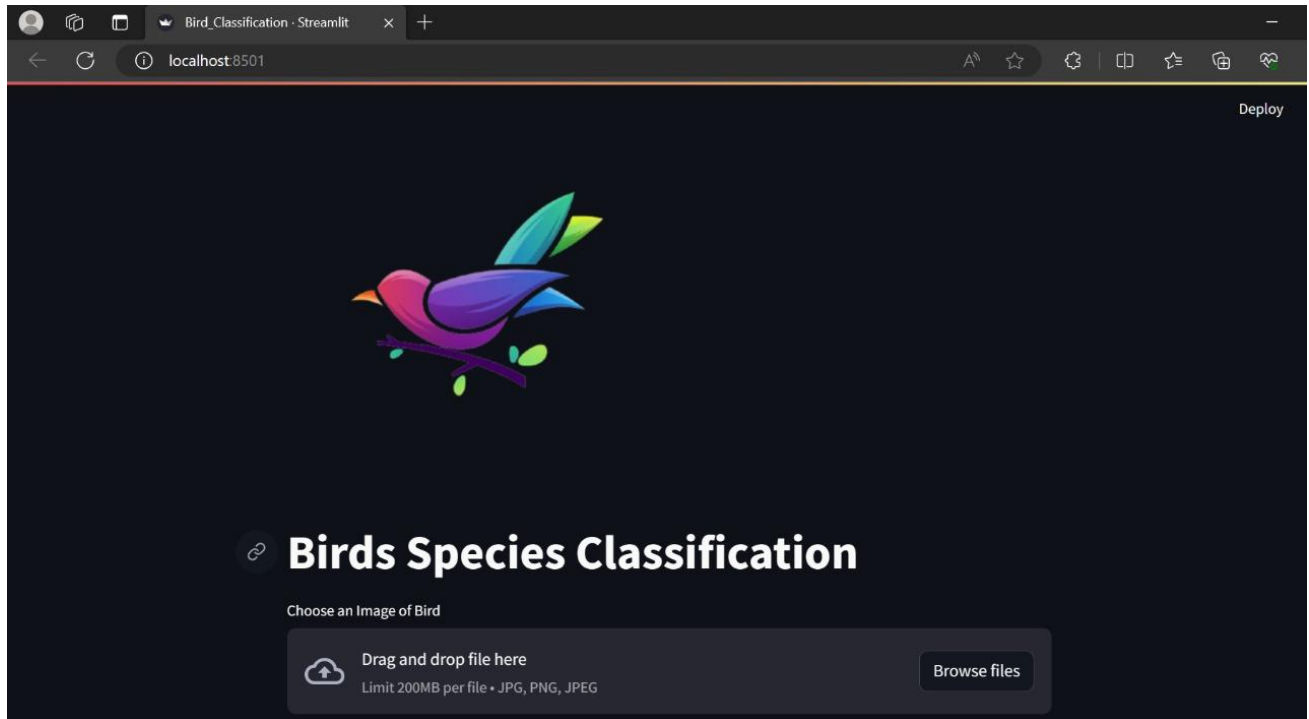


Fig 7.1 Output Interface

Description:

The output interface after execution is displayed as above.

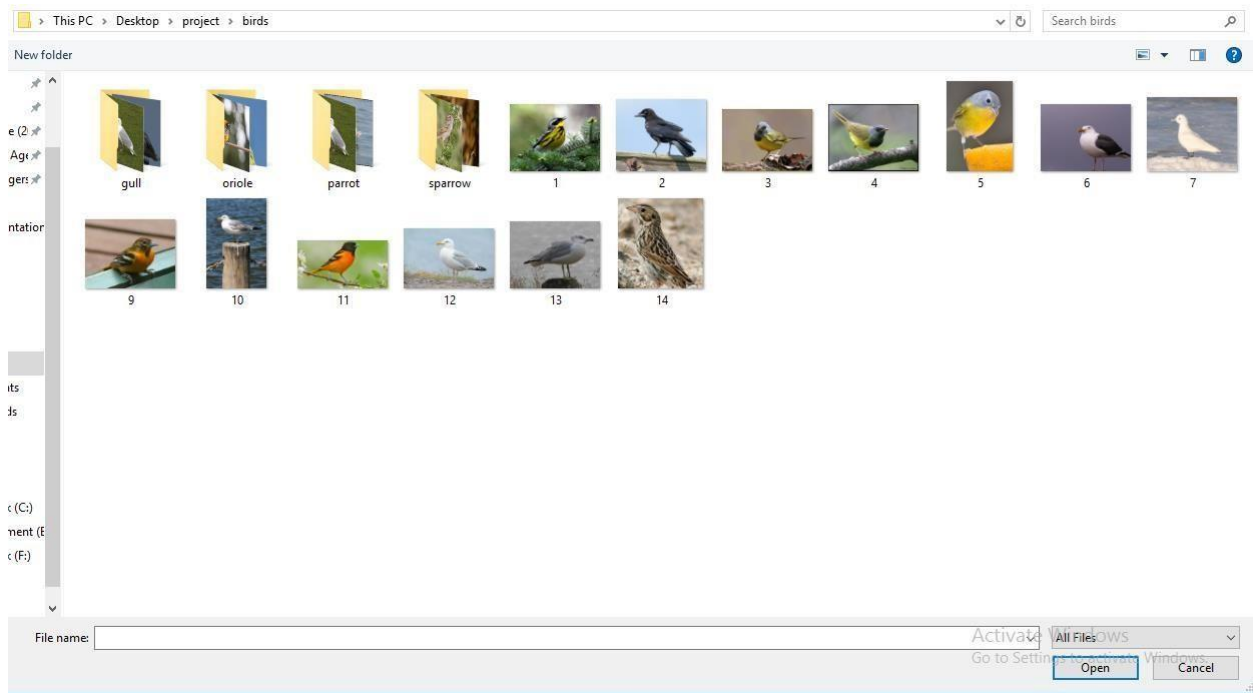


Fig 7.2 User Folder Interface

Description:

After clicking on the browse the system will display as above.

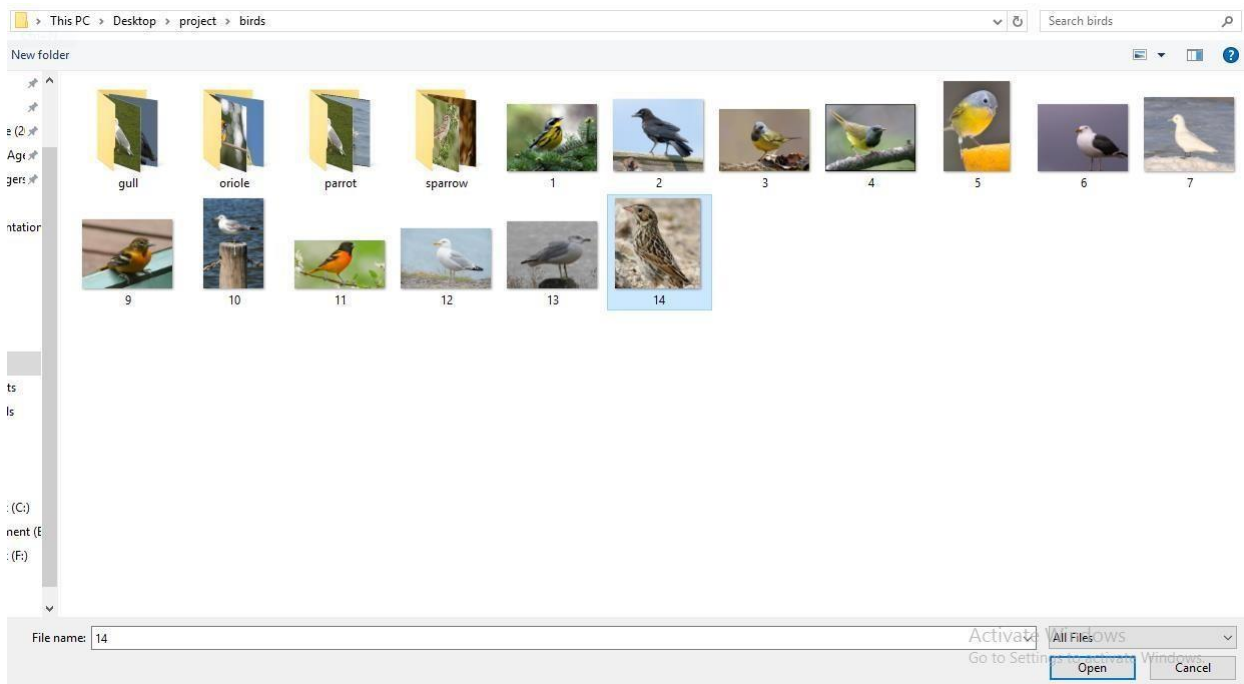


Fig 7.3 User File selection Interface

Description: User selects by clicking on choosing particular image.

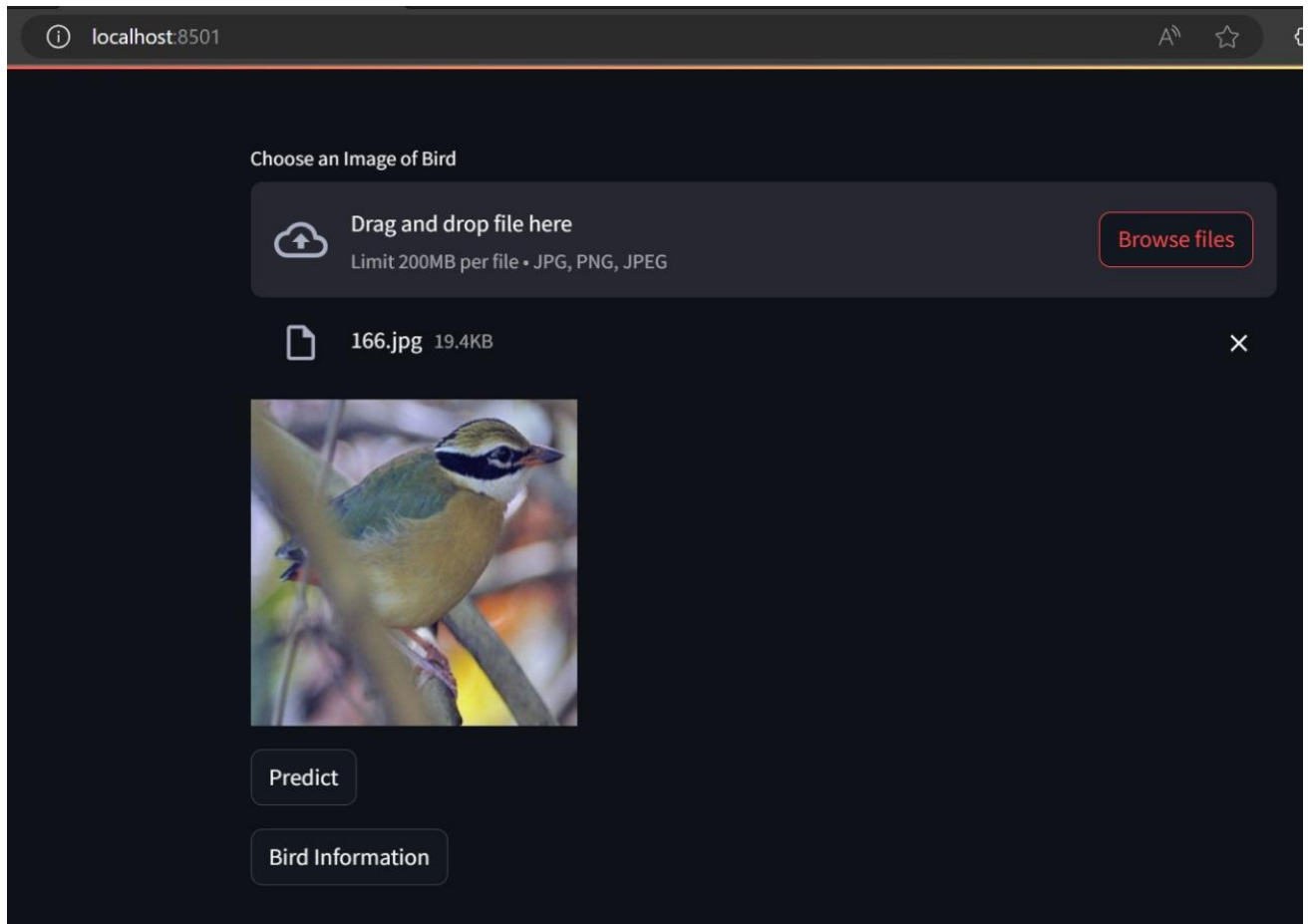


Fig 7.4 Output Interface With Predict and Bird Information Buttons

Description:

The two buttons are displaying (Predict, Bird Information) in the above figure.

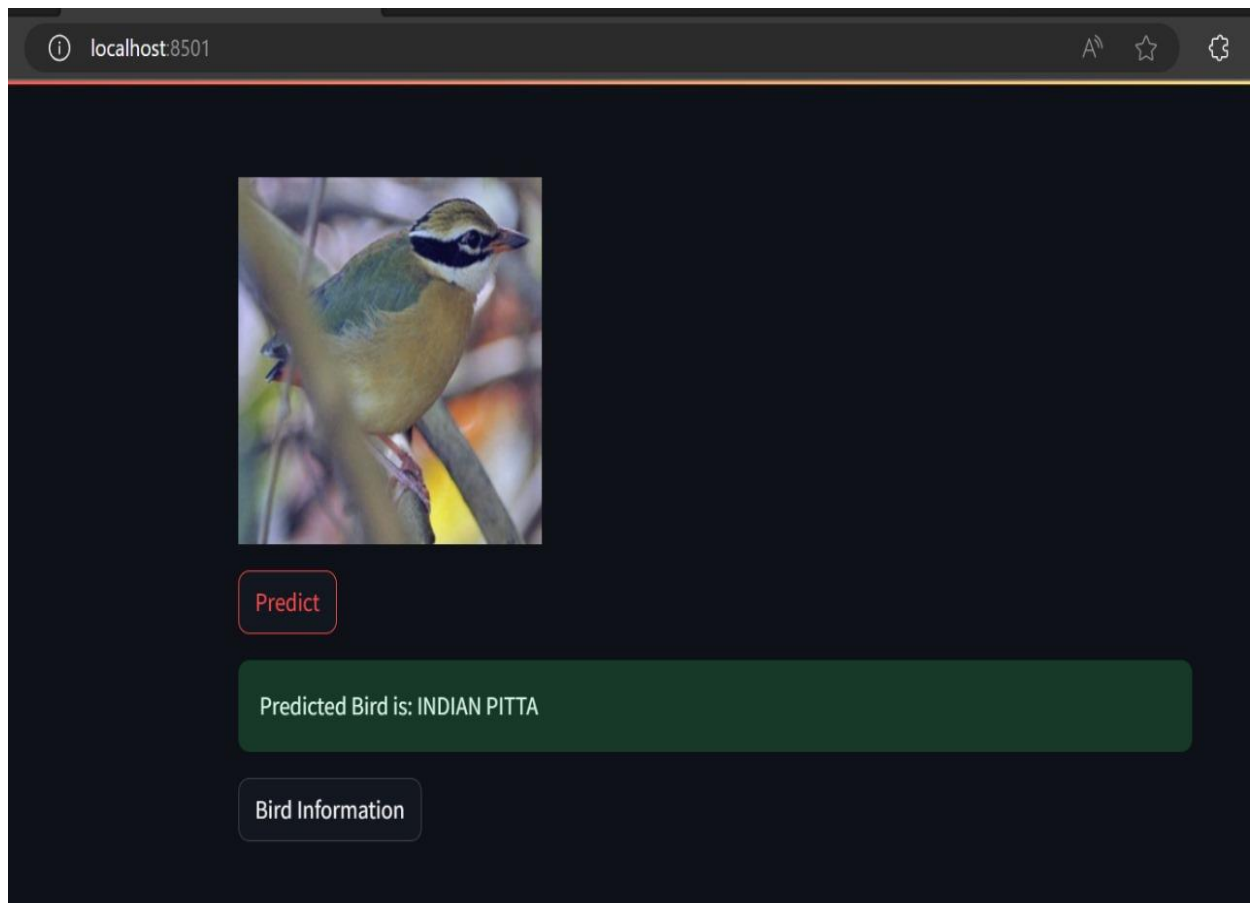


Fig 7.5 Predicted Output Interface

Description:

When we click on Predict button, the respective bird name has displayed.

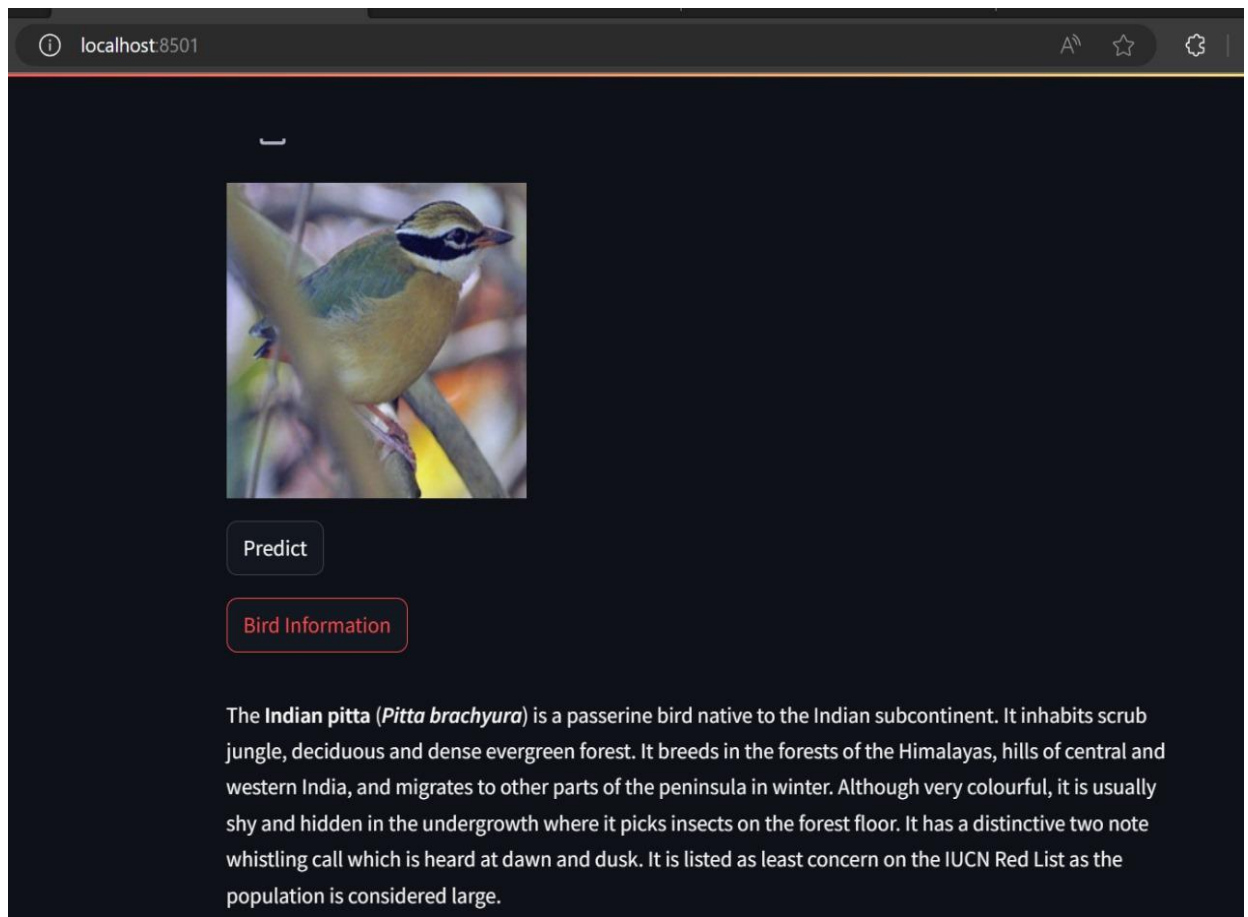


Fig 7.6 Bird Information Interface

Description:

When we click on Bird Information button, the respective bird information has displayed.

CHAPTER 8

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

To identify bird species using the deep learning methods for classification of image accuracy. The generated system is connected with a user-friendly website where user will upload photo for identification purpose and it gives the desired output. Using this project, we can easily identify species of a bird from the image that we captured. The accuracy of our algorithm is 95.09. Beyond its high accuracy, the model offers additional features that enhance its utility and appeal. By incorporating bird information alongside species detection, users can access comprehensive details about identified birds, enriching their understanding of avian diversity and behavior. Furthermore, the inclusion of bird vocalizations adds an immersive dimension to the project, allowing users to engage with the auditory aspects of bird identification.

7.2 FUTURE SCOPE

By training more objects, the application can be made to detect more objects. By increasing the training dataset, the predicting accuracy can be increased. Create an android/iOS app instead of website which will be more convenient to user. System can be implemented using cloud which can store large amount of data for comparison and provide high computing power for processing (in case of Neural network).

REFERENCES

REFERRED LINKS:

- [1] <https://www.ijert.org/bird-species-identification-deeplearning>
- [2] <https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8719894>
- [3] <http://cs229.stanford.edu/proj2014/Aditya%20Bhandari,%20Ameya%20Joshi,%20Rohit%20>
- [4] <http://www.tensorflow.org/datasets/catal>
- [5] pralhad.gavali@ritindia.edu
- [6] Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, etc. 2015. TensorFlow: large-scale machine learning on heterogeneous systems, tensorflow.org.
- [7] M. A. Acevedo, C. J. Corrada-Bravo, H. Corrada-Bravo, L. J. VillanuevaRivera, and T. M. Aide, "Automated classification of bird and amphibian calls using machine learning: A comparison of methods," *Ecological Informatics*, vol. 4, no. 4, pp. 206–214, 2009.
- [8] U. D. Nadimpalli, R. R. Price, S. G. Hall, and P. Bomma, "A comparison of image processing techniques for bird recognition," *Biotechnology Progress*, vol. 22, no. 1, pp. 9–13, 2006
- [9] A. K. A. S. Imran Khan, "Object Analysis in Image Mining," in *INDIA com*, NEW DELHI, 2015.
- [10] Image processing techniques to identify predatory birds in aquacultural settings: Nadimpalli, Uma Devi, "Image processing techniques to identify predatory birds in aquacultural settings" (2005). LSU Master's Theses. 276. https://digitalcommons.lsu.edu/gradschool_theses/27

APPENDIX

6.1 SAMPLE CODE:

```
def processed_img(img_path):
    img=load_img(img_path)
    img_size=img.size
    width=img_size[0]
    height=img_size[1]
    res="bird"
    if (width==224 and height==224) or True:
        img=load_img(img_path,target_size=(224,224,3))
        img=img_to_array(img)
        img=img/255
        img=np.expand_dims(img,[0])
        answer=model.predict(img)
        y_class = answer.argmax(axis=-1)
        print(y_class)
        y = " ".join(str(x) for x in y_class)
        y = int(y)
        # if y in lab:
        res = lab[y]
        # print(res)
        return res
    else:
        # print("Please Upload Bird Image Only")
        return "bird"

def run():
    img1 = Image.open('./meta/logo1.png')
    img1 = img1.resize((350,350))
    st.image(img1,use_column_width=False)
    st.title("Birds Species Classification")
    # st.markdown("<h4 style='text-align: left; color: #d73b5c;
'>* Data is based "270 Bird Species also see 70 Sports Dataset"</h4>",
```

```

        # unsafe_allow_html=True)

img_file = st.file_uploader("Choose an Image of Bird", type=["jpg", "png"])
if img_file is not None:
    st.image(img_file, use_column_width=False)
    save_image_path = './upload_images/'+img_file.name
    with open(save_image_path, "wb") as f:
        f.write(img_file.getbuffer())
    if st.button("Predict"):
        result = processed_img(save_image_path)
        if(result=="bird"):
            st.success("Please upload bird image")
        else:
            st.success("Predicted Bird is: "+result)
    if st.button("Bird Information"):
        result = processed_img(save_image_path)
        result1=get_wikipedia_info(result)
        st.markdown(result1, unsafe_allow_html=True)

run()

import requests
import streamlit as st

def get_wikipedia_info(bird_name):
    base_url = "https://en.wikipedia.org/w/api.php"

    # Step 1: Search for bird information
    search_params = {
        "action": "query",
        "list": "search",
        "srsearch": bird_name,
        "format": "json"
    }

    search_response = requests.get(base_url, params=search_params)

```

```

search_data = search_response.json()

# Check if there are any search results
if "query" in search_data and "search" in search_data["query"] and
search_data["query"]["search"]:
    # Step 2: Retrieve page content
    page_title = search_data["query"]["search"][0]["title"]
    content_params = {
        "action": "query",
        "titles": page_title,
        "prop": "extracts",
        "exintro": True,
        "format": "json"
    }

    content_response = requests.get(base_url, params=content_params)
    content_data = content_response.json()

    # Step 3: Extract relevant information
    page_id = list(content_data["query"]["pages"].keys())[0]
    bird_info = content_data["query"]["pages"][page_id]["extract"]

    # Print or use the information as needed

    return bird_info

else:
    print(f"No information found for {bird_name}")

# Example usage
get_wikipedia_info("Peregrine Falcon")

from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model

```

```

from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt

IMAGE_SIZE = [224, 224]

train_directory="C:\\Users\\ethik\\Downloads\\archive\\train"
test_directory="C:\\Users\\ethik\\Downloads\\archive\\test"
val_directory="C:\\Users\\ethik\\Downloads\\archive\\valid"

# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False
folders = glob("C:\\Users\\ethik\\Downloads\\archive\\train\\*")
len(folders)
# our layers - you can add more if you want
x = Flatten()(vgg.output)
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=vgg.input, outputs=prediction)

# view the structure of the model
model.summary()

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 298, 298, 16)	448
max_pooling2d (MaxPooling2D)	(None, 149, 149, 16)	0
dropout (Dropout)	(None, 149, 149, 16)	0
conv2d_1 (Conv2D)	(None, 147, 147, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 73, 73, 32)	0
dropout_1 (Dropout)	(None, 73, 73, 32)	0
conv2d_2 (Conv2D)	(None, 71, 71, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 35, 35, 64)	0
dropout_2 (Dropout)	(None, 35, 35, 64)	0
conv2d_3 (Conv2D)	(None, 33, 33, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 128)	0
dropout_3 (Dropout)	(None, 16, 16, 128)	0
conv2d_4 (Conv2D)	(None, 14, 14, 256)	295168
max_pooling2d_4 (MaxPooling2D)	(None, 7, 7, 256)	0
dropout_4 (Dropout)	(None, 7, 7, 256)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 128)	1605760
dense_1 (Dense)	(None, 3)	387

Total params: 17,047,965 (65.03 MB)

Trainable params: 2,333,277 (8.90 MB)

Non-trainable params: 14,714,688 (56.13 MB)

```

model.compile(
    loss='categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                    shear_range = 0.2,

```



```

        zoom_range = 0.2,
        horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory(train_directory,
                                                target_size = (224, 224),
                                                batch_size = 32,
                                                class_mode = 'categorical')

test_set = test_datagen.flow_from_directory(test_directory,
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')

from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam # Or any other optimizer you prefer
from tensorflow.keras.losses import CategoricalCrossentropy # Or any other loss function

r = model.fit_generator(
    training_set,
    validation_data=test_set,
    epochs=10,
    steps_per_epoch=len(training_set),
    validation_steps=len(test_set)
)

```

```
Epoch 1/10
23/23 [=====] - 270s 12s/step - loss: 1.2737 - acc:
0.4286
Epoch 2/10
23/23 [=====] - 254s 11s/step - loss: 0.8910 - acc:
0.6124
Epoch 3/10
23/23 [=====] - 257s 11s/step - loss: 0.6674 - acc:
0.7188
Epoch 4/10
23/23 [=====] - 259s 11s/step - loss: 0.5567 - acc:
0.7202
Epoch 5/10
23/23 [=====] - 256s 11s/step - loss: 0.6000 - acc:
0.7396
Epoch 6/10
23/23 [=====] - 262s 11s/step - loss: 0.4393 - acc:
0.8254
Epoch 7/10
23/23 [=====] - 248s 11s/step - loss: 0.4371 - acc:
0.8433
Epoch 8/10
23/23 [=====] - 293s 13s/step - loss: 0.3355 - acc:
0.8717
Epoch 9/10
23/23 [=====] - 259s 11s/step - loss: 0.2990 - acc:
0.8769
Epoch 10/10
23/23 [=====] - 265s 12s/step - loss: 0.3023 - acc:
0.8920
saved model to disk
```

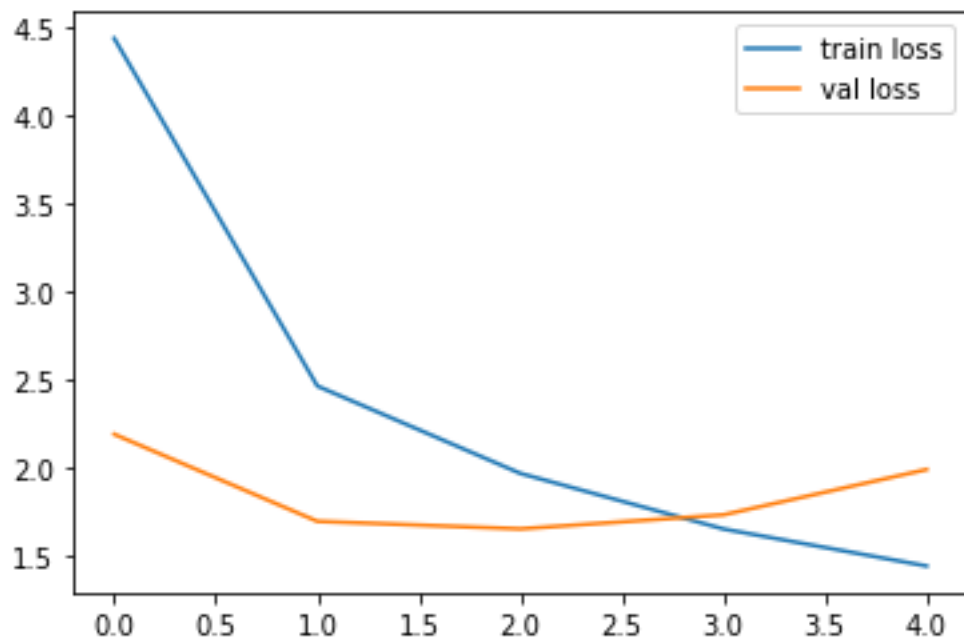
The accuracy of the model is displayed as above. As the number of epochs is increased the accuracy gets improved.

```
plt.plot(r.history['loss'], label='train loss')
```

```
plt.plot(r.history['val_loss'], label='val loss')
```

```
plt.legend()
```

```
plt.show()
```



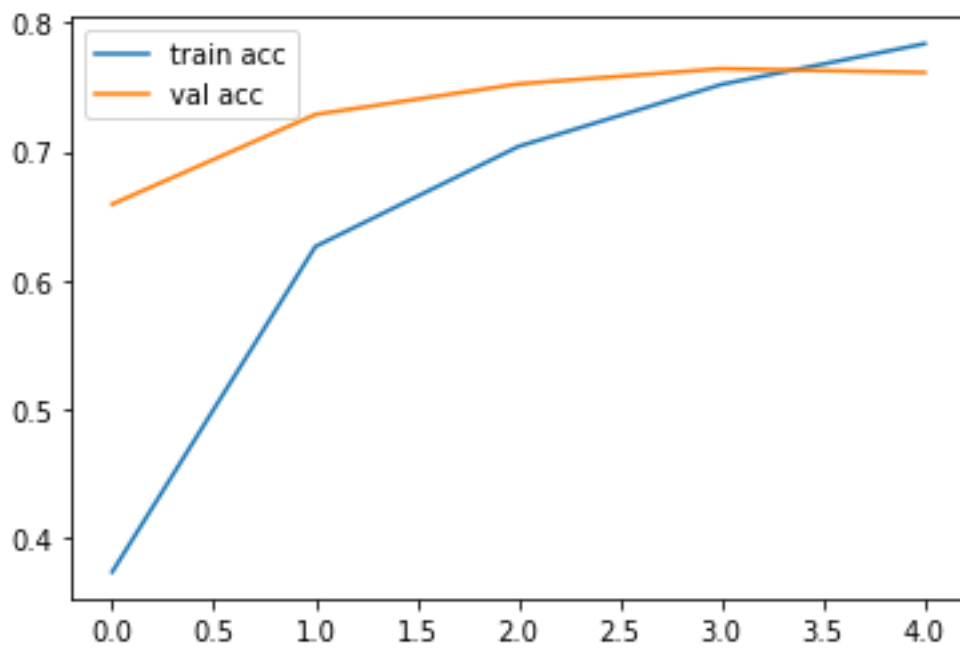
accuracies

```
plt.plot(r.history['accuracy'], label='train acc')
```

```
plt.plot(r.history['val_accuracy'], label='val acc')
```

```
plt.legend()
```

```
plt.show()
```



```
model.save('BC.h5')
```

```
from keras.models import load_model
```

```
from keras.preprocessing.image import load_img, img_to_array
```

```
model1 = load_model('./BC.h5', compile=False)
```

```
lab = training_set.class_indices
```

```
lab={k:v for v,k in lab.items() }
```

```
def output(location):
```

```
    img=load_img(location,target_size=(224,224,3))
```

```
    img=img_to_array(img)
```

```
    img=img/255
```

```
    img=np.expand_dims(img,[0])
```

```
answer=model1.predict(img)

y_class = answer.argmax(axis=-1)

y = " ".join(str(x) for x in y_class)

y = int(y)

res = lab[y]

return res
```