**ABSTRACT**

Bird watching is an art of observing, studying and researching on birds. People who pursue bird watching as a hobby or passion can be termed as bird watchers and others involved in scientific study and research on birds are termed as ornithologists. All ornithologists are bird-watchers but not all bird-watchers are ornithologists. Accurate bird-identification is an important aspect of bird-watching. In India, not many software are available that use input image for identification of a specie, thus making the identification process cumbersome for bird-watchers. Also, it requires a lot of study as well as time to identify birds from their colour, shape, size, habitat, etc. Our Project aims to overcome this issue by building a software as a Project. Concept of transfer learning and pre-trained algorithm made feature extraction possible from an image. The results obtained were of high efficiency as the software could easily identify a bird species from an image whose data was present in database. Our application can be used by anyone who wants to know what kind of bird species the user is watching. All that the user must do is open the application and upload the image and click on predict button. The algorithm correctly predicts the bird species.

The application is provided as a Web Application that works on all kind of devices like laptop, mobiles or tablets. This application delivered through the web, built using common web technologies including HTML, CSS and JavaScript which is developed on python Flask. It is fast, reliable and intended to work on any platform that uses a standard-compliant browser exactly like any native application.

*(Keywords: Image Recognition, Bird Identification, Transfer Learning, Feature Extraction, Machine Learning)*

1. **INTRODUCTION**

Many people across countries are getting into this interest of bird-watching as a hobby or extra-curricular activity. In modern world, it acts as a great stress buster and a cheap way of getting connected with nature. Another benefit of bird-watching is awareness about nature conservation by observing behaviour, migratory pattern population, and conservation status of bird species. From conservation point of view, it is important that more and more number of people turn towards bird-watching and help collect data that can be used to study birds.

Sometimes, bird identification can be difficult for beginners as well as experienced bird-watchers. The focus of our project is to simplify the identification process. As far as India is concerned, there is no bird identification software available that takes image as an input and gives the identity of the birds as output. There are bird identification software and websites available for counties such as US and Canada (e.g. eBird, Merlin Bird ID) but very few good quality bird identification software are available in India.

For beginners, the software will be of great help because at the early stages of bird-watching, identifying or differentiating between different species can be difficult and frustrating. To experienced bird-watchers also this software will majorly help in identification

* 1. **Traditional Approach to Identify Bird Species**

It is intimidating to try and identify the birds seen by the bird watchers when they first begin bird watching. The sheer number and variety of birds, along with the slight variations within some species or families of birds, can overwhelm and discourage the bird watcher at first. They will soon realize, however, that they can learn to identify the birds they see by familiarizing themself with particular traits or features that make differentiating between them easier.The factors one needs to consider when identifying birds include family traits, behaviour, size, colour, location and field markings.

**Family Features:** Most bird families have several distinct features or traits in common that they don't share with other families.

**Behaviour:** Behaviours such as where it is gathering food, the bird's flight pattern, the way it interacts with other birds and how shy or gregarious it is. Some birds are quite solitary while others tend to be very sociable. Some birds hop about while others perch with very little movement except when in flight. Some birds can creep up or down a tree trunk while others aren't able to do this. All of these behaviours can be very revealing.

**Size**: Birds vary in size from hummingbirds, which can be only a few inches or less long, to enormous predators such as eagles or hawks.

**Colour:** Birds can be found in variety of colour variations with various colour shades on their body parts.

**Location:** Location like where the bird is sitting -- in a low shrub, high on a telephone wire or in a tree? What about whether you have spotted the bird repeatedly in fields or whether it is usually trying to hide in trees? Birds have very entrenched, instinctive patterns of behaviour that involve their location even within specific habitats.

**Field Markings**: Roger Tory Peterson, the developer of the first widely used birding field guides in the 1930's, developed the "field marking" system that is still used today to identify birds based on particular physical features of birds. Peterson realized that the most distinguishing features of birds tended to fall into four specific areas of the bird -- the eye area (eyebrow), the rump (the back of the bird just above the tail feathers), the tail feathers and the wing bars (the distinctive striping or stippling patterns on the wings of the bird.

For beginner to identify the bird species with this approach will take a lot of time and will require a professional guide to get them verified. This altogether will require a lot of time and effort. Our project aims to identify bird species by taking an image as an input and giving the output as the name of the identified bird species.

* 1. **Objectives**

The objective of this project is to deploy good quality software which correctly identifies the bird species from and image and gives appropriate results. It also strives to stand as a guideline for bird-watchers and ornithologists. Our aim is to identify bird species by giving an image as an input to the software and which give us result as name of the predicted bird species. We aim to create a software which is

* Easy to use.
* Which correctly identifies bird species and gives appropriate results.
* To provide result to the user in very less time
* To keep people motivated and inclined towards nature.
  1. **Methodology**

The project is divided into two phases i.e. The Training Phase and the Prediction Phase or the end application. The end application is a web application which is developed in Python Flask which is a micro web framework written in Python. To develop this application and to achieve the desired result which is “Bird Species Classification through an Image” we will be using Keras which is an open-source neural network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, R, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System). Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing deep neural network code. Advantages of using Keras are:

* Keras prioritizes developer experience
* Keras has broad adoption in the industry and the research community
* Keras makes it easy to turn models into products
* Keras has strong multi-GPU & distributed training support
* Keras is at the nexus of a large ecosystem

From Keras Applications we will be using VGG16 which is a pre-trained model for training our model on our dataset. Keras Applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. A pre-trainedmodel is a model that was trained on a large benchmark dataset to solve a problem similar to the one that we want to solve. We will be training our model on our dataset.

* 1. **Dataset**

Our Dataset consists of 50 bird species, 6558 training images, and 250 validation images (5 per species). All images are 224 X 224 X 3 colour images in jpg format. Images were gather from internet searches by species name. Once the image files for a species was downloaded they were checked for duplicate images. All duplicates detected were deleted in order to prevent their being images common between the training, validation sets. After that the images were cropped so that the bird occupies at least 50% of the pixel in the image. Then the images were resized to 224 X 224 X3 in jpg format. The cropping ensures that when processed by a CNN there is adequate information in the images to create a highly accurate classifier. All files were also numbered sequential starting from one for each species. So validation images are named 1.jpg to 5.jpg. Training images are also numbered sequentially with "zeros" padding. For example 001.jpg, 002.jpg ….010.jpg, 011.jpg …..099.jpg, 100jpg, 102.jpg etc. The zero's padding preserves the file order when used with python file function and Keras flow from directory. The training set is not balanced, having a varying number of files per species. However each species has at least 100 training image files. This imbalanced did not affect kernel classifier as it achieved over 80% accuracy on the test set.

* 1. **Organization of the Project**

1. **ANALYSIS OF BIRD SPECIES CLASSIFICATION**
   1. **Existing System**

Basically bird identification is done visually or acoustically. The main visual components comprise of birds shape, its wings, size, pose, colour, etc. However, while considering the parameters time of year must be taken into consideration because birds wings changes according to their growth. The acoustics components comprise the songs and call that birds make. The marks that distinguish one bird from another are also useful, such as breast spots, wing bars which are described as thin lines along the wings, eye rings, crowns, eyebrows. The shape of the beak is often an important aspect as a bird can recognized uniquely. The characteristics of bird such as shape and posture are the mostly used to identify birds. Mostly experts can identify a bird from its silhouette because this characteristic is difficult to change. A bird can also be differentiated using its tail. The tail can be recognized in many ways such as notched, long and pointed, or rounded. Sometimes legs are also used for recognizing an image in format long, or short.

By considering a single parameter will not yield an accurate result. So, multiple parameters are to be considered in order to get appropriate output. The size of a bird in an image varies depending upon factors such as the resolution, distance between the birds and the capturing device, and the focal distance of the lens. Therefore, based on a practical observation for large number of images, images are differentiated on the basis of colour which consists of various pixel. In depth it is found that greater the image quality greater is its accuracy.

There are a few websites available like eBird India which is designed for the use of birders and eBirders from India. For identification of Bird Species there are different volunteers available. All you do is upload the image and wait for the reply. This process takes lot of time and effort. We aim to overcome this by creating a model which has good accuracy.

* 1. **Proposed Approach**
     1. **Data Augmentation**

To increase the number of training samples per class and reduce the effect of class imbalance, data augmentation is used. Relevant image augmentation techniques are chosen according bird type of each classes. Those techniques are Rotation Range, Width Shift Range, Height Shift Range, Shear Range, Zoom Range, Horizontal Flip, Fill Mode (viz. nearest). After data augmentation, training dataset increased from to 6558 images. We have already applied data augmentation to our dataset.

* + 1. **Transfer Learning**

Training an Image Classification model even with Deep Learning is not an easy task. In order to get sufficient accuracy, without over-fitting requires a lot of training data. If we try to train a deep learning model from scratch, and hope build a classification system with similar level of capability of an ImageNet-level model, then we'll need a dataset of about a million training examples (plus, validation examples also). Needless to say, it's not easy to acquire, or build such a dataset practically. Deep Learning supports an immensely useful feature called 'Transfer Learning'. Basically, we are able to take a pre-trained deep learning model - which is trained on a large-scale dataset such as ImageNet - and re-purpose it to handle an entirely different problem. The idea is that since the model has already learned certain features from a large dataset, it may be able to use those features as a base to learn the particular classification problem we present it with. This task is further simplified since popular deep learning models such as VGG16 and their pre-trained ImageNet weights are readily available. The Keras framework even has them built-in in the keras.applications package.

The basic technique to get transfer learning working is to get a pre-trained model (with the weights loaded) and remove final fully-connected layers from that model. We then use the remaining portion of the model as a feature extractor for our smaller dataset. These extracted features are called "Bottleneck Features" (i.e. the last activation maps before the fully-connected layers in the original model). We then train a small fully-connected network on those extracted bottleneck features in order to get the classes we need as outputs for our problem.

* 1. **Advantages and Disadvantages**

The advantage of the existing system is that the volunteers will correctly identify the bird species but the disadvantage of it is that it requires lots of time and human efforts moreover if you have a lot of bird images to classify it becomes a tiresome task. For classification of the correct bird species it requires lots of detail study and analysis of the birds and their features. With increasing number of bird watchers all over the world more and more number of volunteers are needed to come up with accurate results.

1. **Using Transfer Learning for Multi-Class Classification in Keras and TensorFlow**
   1. **VGG16 Architecture**

The input to cov1 layer is of fixed size 224 x 224 RGB image. The image is passed through a stack of convolutional (conv.) layers, where the filters were used with a very small receptive field: 3×3 (which is the smallest size to capture the notion of left/right, up/down, centre). In one of the configurations, it also utilizes 1×1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv.  Layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2×2 pixel window, with stride 2.

Three Fully-Connected (FC) layers follow a stack of convolutional layers (which has a different depth in different architectures): the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks.

All hidden layers are equipped with the rectification (ReLU) non-linearity. It is also noted that none of the networks (except for one) contain Local Response Normalisation (LRN), such normalization does not improve the performance on the ILSVRC dataset, but leads to increased memory consumption and computation time. We modify this architecture according to our needs.

* 1. **Extracting Bottleneck Features**

Bottleneck Features are the last activation maps before the fully-connected layers in a VGG16 model. If we only use the VGG16 model up until the fully-connected layers, we can convert the input X (image of size 224 x 224 x 3, for example) into the output Y with size 512 x 7 x 7. We then train a simple CNN with fully connected layers using Y as input and categorical values Z as output.

In order to build out model, we need to go through the following steps,

* Save the bottleneck features from the VGG16 model.
* Train a small network using the saved bottleneck features to classify our classes, and save the model (we call this the 'top model').
* Use both the VGG16 model along with the top model to make predictions.

The code snippet below extracts bottleneck features for training data. It takes around 30 minutes to extract bottleneck features for 50 classes and 6558 Images. We do the same for validation set.

* 1. **Drawback of VGG16**

Unfortunately, there are two major drawbacks with VGGNet:

* It is painfully slow to train.
* The network architecture weights themselves are quite large (concerning disk/bandwidth).

Due to its depth and number of fully-connected nodes, VGG16 is over 533MB. This makes deploying VGG a tiresome task.VGG16 is used in many deep learning image classification problems; however, smaller network architectures are often more desirable (such as SqueezeNet, GoogLeNet, etc.). But it is a great building block for learning purpose as it is easy to implement.

1. **Implementation and Coding**
   1. **Software and Libraries used.**

Python, with its rich technology stack, has an extensive set of libraries for artificial intelligence and machine learning. Here are some of them which we have used in our project:

* **Keras**, **TensorFlow**, and **Scikit-learn** for machine learning
* **NumPy** for high-performance scientific computing and data analysis
* **SciPy** for advanced computing
* **Pandas** for general-purpose data analysis
* **Seaborn** for advanced data visualization
* **Matplotlib** for data visualization

Scikit-learn features various classification, regression, and clustering algorithms, including support vector machines, random forests, gradient boosting, k-means, and DBSCAN, and is designed to work with the Python numerical and scientific libraries NumPy and SciPy.

* 1. **Keras Library Applications**

Keras was created to be user friendly, modular, easy to extend, and to work with Python. The API was “designed for human beings, not machines,” and “follows best practices for reducing cognitive load.” Neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes are all standalone modules that you can combine to create new models. New modules are simple to add, as new classes and functions. Models are defined in Python code, not separate model configuration files.

The biggest reasons to use Keras stem from its guiding principles, primarily the one about being user friendly. Beyond ease of learning and ease of model building, Keras offers the advantages of broad adoption, support for a wide range of production deployment options, integration with at least five back-end engines (TensorFlow, CNTK, Theano, MXNet, and PlaidML), and strong support for multiple GPUs and distributed training. Plus, Keras is backed by Google, Microsoft, Amazon, Apple, Nvidia, Uber, and others.

* 1. **Coding**

# importing the necessary packages

import numpy as np

from keras.preprocessing.image import ImageDataGenerator, img\_to\_array, load\_img

from keras.models import Sequential

from keras.layers import Dropout, Flatten, Dense

from keras import applications

from keras.utils.np\_utils import to\_categorical

import matplotlib.pyplot as plt

import math

# dimensions of our images.

img\_width, img\_height = 224, 224

# paths to folders

top\_model\_weights\_path = 'D:/Bird Species Classification/bottleneck\_fc\_model.h5'

train\_data\_dir = 'D:/Bird Species Classification/birdImages/train'

validation\_data\_dir = 'D:/Bird Species Classification/birdImages/validation'

# number of epochs to train top model

epochs = 50

# batch size used by flow\_from\_directory and predict\_generator

batch\_size = 16

# Here, we create our VGG16 model - without the final fully-connected

# layers (by specifying include\_top=False) - and load the ImageNet weights

model = applications.VGG16(include\_top=False, weights='imagenet')

# Creating the data generator for training images,

# and run them on the VGG16 model to save the bottleneck features for training.

train\_datagen = ImageDataGenerator(rescale=1. / 255)

train\_generator = train\_datagen.flow\_from\_directory(

train\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode=None,

shuffle=False)

nb\_train\_samples = len(train\_generator.filenames)

num\_classes = len(train\_generator.class\_indices)

predict\_size\_train = int(math.ceil(nb\_train\_samples / batch\_size))

bottleneck\_features\_train = model.predict\_generator(

train\_generator, predict\_size\_train)

np.save('D:/Bird Species Classification/bottleneck\_features\_train.npy',

bottleneck\_features\_train)

# generator.filenames - contains all the filenames of the training set.

# By getting its length, we can get the size of the training set

# generator.class\_indices - is the map/dictionary for the class-names

# and their indexes. Getting its length gives us the number of classes

# repeating the same steps for validation data

valid\_datagen = ImageDataGenerator(rescale=1. / 255)

valid\_generator = valid\_datagen.flow\_from\_directory(

validation\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode=None,

shuffle=False)

nb\_validation\_samples = len(valid\_generator.filenames)

predict\_size\_validation = int(math.ceil(nb\_validation\_samples / batch\_size))

bottleneck\_features\_validation = model.predict\_generator(

valid\_generator, predict\_size\_validation)

np.save('D:/Bird Species Classification/bottleneck\_features\_validation.npy',

bottleneck\_features\_validation)

# with the bottleneck features saved, now we're ready to train our top model.

# In order to train the top model, we need the class labels for each of the

# training/validation samples. We use a data generator for that also.

# We also need to convert the labels to categorical vectors.

train\_datagen\_top = ImageDataGenerator(rescale=1./255)

train\_generator\_top = train\_datagen\_top.flow\_from\_directory(

train\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode='categorical',

shuffle=False)

nb\_train\_samples = len(train\_generator\_top.filenames)

num\_classes = len(train\_generator\_top.class\_indices)

# load the bottleneck features saved earlier

train\_data = np.load('D:/Bird Species Classification/bottleneck\_features\_train.npy')

# get the class lebels for the training data, in the original order

train\_labels = train\_generator\_top.classes

# convert the training labels to categorical vectors

train\_labels = to\_categorical(train\_labels, num\_classes=num\_classes)

# Repeating the same steps for validation feature as well

valid\_datagen\_top = ImageDataGenerator(rescale=1./255)

valid\_generator\_top = valid\_datagen\_top.flow\_from\_directory(

validation\_data\_dir,

target\_size=(img\_width, img\_height),

batch\_size=batch\_size,

class\_mode=None,

shuffle=False)

nb\_validation\_samples = len(valid\_generator\_top.filenames)

validation\_data = np.load('D:/Bird Species Classification/bottleneck\_features\_validation.npy')

validation\_labels = valid\_generator\_top.classes

validation\_labels = to\_categorical(validation\_labels, num\_classes=num\_classes)

# create and train a small fully-connected network - the top model - using

# the bottleneck features as input, with our classes as the classifier output.

model = Sequential()

model.add(Flatten(input\_shape=train\_data.shape[1:]))

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(optimizer='rmsprop',

loss='categorical\_crossentropy',

metrics=['accuracy'])

model.summary()

# model.summary() - gives us summary of our model. (Number of layers used)

# To know how our model gets trained, we save the log file and plot it using line Plot

from keras.callbacks import EarlyStopping, ModelCheckpoint

from keras.callbacks import CSVLogger

filepath="D:/Bird Species Classification/bottleneck\_fc\_model.h5"

checkpoint = ModelCheckpoint(filepath,

monitor='val\_loss',

verbose=1,

save\_best\_only=True,

save\_weights\_only=False,

mode='min')

training\_log = CSVLogger("D:/Bird Species Classification/training\_log.log",

separator = ',',

append=False)

# patience is the number of epochs to wait before early stop

# if no progress on the validation set.

early\_stop\_ft = EarlyStopping(monitor='val\_loss',

mode='min',

verbose=1,

patience=5)

callbacks\_list = [early\_stop\_ft, training\_log]

# Training our model

model.fit(train\_data,

train\_labels,

epochs=epochs,

batch\_size=batch\_size,

validation\_data=(validation\_data, validation\_labels),

callbacks=callbacks\_list)

model.save\_weights(top\_model\_weights\_path)

#model.load\_weights(top\_model\_weights\_path)

(eval\_loss, eval\_accuracy) = model.evaluate(

validation\_data,

validation\_labels,

batch\_size=batch\_size,

verbose=1)

print("[INFO] accuracy: {:.2f}%".format(eval\_accuracy \* 100))

print("[INFO] Loss: {}".format(eval\_loss))

'''

Results:

[INFO] accuracy: 76.00%

[INFO] Loss: 1.1039417777508498

'''

###############################

# Plotting Line Plot for train data

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

history = pd.read\_csv("D:/Bird Species Classification/training\_log.log",

sep=',',

engine='python')

epochs = range(len(history['accuracy']))

fig, (ax1, ax2) = plt.subplots(nrows = 1, ncols = 2)

sns.lineplot(x = epochs, y=history['accuracy'], ax=ax1, label = 'accuracy')

sns.lineplot(x = epochs, y=history['val\_accuracy'], ax=ax1, label = 'val\_accuracy')

ax1.set\_ylim(0, 1.05);

ax1.set\_title('model accuracy');

sns.lineplot(x = epochs, y=history['loss'], ax=ax2, label = 'loss')

sns.lineplot(x = epochs, y=history['val\_loss'], ax=ax2, label = 'val\_loss')

ax2.set\_title('model loss');

fig.suptitle('Model Training Metrics', fontsize = 16);

max\_val\_acc = max(history['val\_accuracy'])

print(f'This model achieved a max validation accuracy of {max\_val\_acc}')

'''

Result:

This model achieved a max validation accuracy of 0.8000000119209291

'''

**###############################**

1. **Results**