IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p-ISSN: 2278-8727, Volume 27, Issue 3, Ser. 4 (May. – June. 2025), PP 41-49
www.iosrjournals.org

Bird Species Classification Using MobileNetV2 and CNN: A Comparative Study with Web Deployment

Dipayan Chatterjee and Sudesh Pundir
Department of Statistics, Pondicherry University

Abstract
Bird species classification plays a crucial role in ecological research, biodiversity monitoring, and wildlife conservation. In this study, we propose a machine learning-based image classification system for identifying bird species using a custom dataset of 20 bird species. We explore and compare the performance of a Convolutional Neural Network (CNN) and MobileNetV2 architecture for classification. With a total of 3408 images divided into training, validation, and test sets, we fine-tuned both models and found that MobileNetV2 significantly outperformed the custom CNN model in terms of accuracy and generalization. The trained MobileNetV2 model was then converted to TensorFlow Lite (.tflite) format and integrated into a Flask-based web application. The system predicts the species name with associated confidence or informs the user if the species is outside the known dataset. This research not only highlights the potential of deep learning in species classification but also demonstrates real-world application deployment.

Date of Submission: 12-06-2025 Date of Acceptance: 25-06-2025

I. Introduction

Bird classification is essential in ornithology, enabling efficient data collection, conservation planning, and ecological monitoring. Manual classification, however, is time-consuming and error-prone. With the advancement of computer vision and deep learning, automated species identification has become feasible.

Recent developments in convolutional neural networks (CNNs) and pre-trained models like MobileNetV2 have significantly improved image classification tasks. MobileNetV2, designed for mobile and embedded vision applications, offers a balance between computational efficiency and accuracy.

This research compares a custom CNN architecture and MobileNetV2 for bird species classification, followed by deployment of the better-performing model in a web application using Flask. The system was trained on a well-organized dataset and provides users with predictions and confidence scores.

II. Methodology

2.1 Dataset Description

The dataset consists of images of 20 different bird species, divided into:

- Training set: 3208 images
- Validation set: 100 images (5 per class)
- Test set: 100 images (5 per class)

Each class is stored in a separate folder under respective directories.

The 20 species of birds which consists the dataset are: Abbotts Babbler, Abbotts Booby, Abyssinian Ground Hornbill, African Crowned Crane, African Emerald Cuckoo, African Firefinch, African Oyster Catcher, African Pied Hornbill, African Pygmy Goose,

Albatross, Alberts Towhee, Alexandrine Parakeet, Alpine Chough, Altamira Yellowthroat, American Avocet, American Bittern, American Coot, American Flamingo, American Goldfinch, American Kestrel.

2.2 Data Preprocessing and Augmentation

To improve generalization and avoid overfitting, data augmentation techniques were applied using TensorFlow's ImageDataGenerator.

TensorFlow is an open-source machine learning framework developed by Google Brain Team. It is designed to facilitate the development and deployment of machine learning and deep learning models across a variety of platforms, including desktops, servers, and mobile devices.

- Tensors: Multidimensional arrays that form the basic data structure in TensorFlow. Every computation involves operations on tensors.
- Graphs: TensorFlow builds a computation graph, where nodes represent operations and edges represent the data (tensors) that flow between operations.

- Eager Execution: Allows for immediate execution of operations, which is useful for debugging and development.
- Model Building APIs: Includes both low-level APIs and high-level APIs like Keras for rapid prototyping and training of models.
- Deployment Support: TensorFlow supports deployment via TensorFlow Lite (for mobile and embedded devices), TensorFlow Serving (for production environments), and TensorFlow.js (for browser-based ML).

Why TensorFlow in used in this project:
- Integration with Keras for quick model prototyping.
- Compatibility with pre-trained models (e.g., MobileNetV2).
- Tools for model optimization and conversion (e.g., .tflite format).
- Support for data pipelines and augmentation using ImageDataGenerator.

The ImageDataGenerator class is part of TensorFlow's Keras API, used to efficiently load, preprocess, and augment images in real-time during model training.

Key Functions:
1. Data Augmentation: Dynamically alters training images to improve generalization and avoid overfitting. Common transformations include: Rotation, Zoom, Shearing, Flipping, Shifting.
2. Normalization: Scales pixel values (e.g., rescale=1./255) to bring them to a uniform range (0 to 1), which speeds up training and improves convergence.
3. Batch Generation: Generates batches of image-label pairs from directories, which allows memory-efficient training even on large datasets.

2.3 CNN Model Architecture

A baseline CNN was created with convolutional layers, max pooling, dropout, flattening, and dense layers.
Convolutional Neural Networks (CNNs) are a class of deep learning models specifically designed for analyzing visual data such as images. CNNs have revolutionized computer vision tasks like image classification, object detection, and facial recognition.

Key Layers in a CNN:
1. Convolutional Layer
   - The core building block of a CNN.
   - Applies filters (kernels) to the input image to extract spatial features (edges, textures, patterns).
   - Mathematically, it performs a dot product between the filter and sections of the input image.
   - Output: A feature map that highlights relevant features.

2. Activation Function (ReLU)
   - Applied after each convolution to introduce non-linearity.
   - ReLU (Rectified Linear Unit) replaces negative values with zero, helping the model learn complex patterns.
3. Pooling Layer (MaxPooling)
   - Reduces the spatial dimensions (width and height) of the feature maps.
   - MaxPooling selects the maximum value from a patch, preserving dominant features while reducing computation and overfitting.
4. Dropout Layer
   - Randomly turns off a fraction of neurons during training.
   - Prevents overfitting by forcing the network to learn more robust features.
5. Flatten Layer
   - Converts the 2D feature maps into a 1D vector before feeding it into fully connected layers.
6. Fully Connected Layer (Dense Layer)
   - Traditional neural network layers where each neuron is connected to all neurons in the previous layer.
   - Performs classification based on the extracted features.
7. Output Layer
   - Uses Softmax Activation to assign probabilities to each class.
   - The class with the highest probability is chosen as the prediction.

Why Use CNNs in This Project:
- Effective feature extraction from images without manual engineering.
- Parameter sharing (same filter used across the image) reduces memory usage.
- Translation invariance, meaning CNNs can recognize objects regardless of position in the image.

- Ideal for image classification tasks like bird species identification.

2.4 MobileNetV2 Transfer Learning

MobileNetV2 pre-trained on ImageNet was fine-tuned with additional pooling and dense layers.

MobileNetV2 is a lightweight deep convolutional neural network designed for efficient computation, particularly on mobile and embedded devices. Developed by Google in 2018, it builds upon the success of MobileNetV1 with improved accuracy and reduced computational complexity.

Key Innovations in MobileNetV2:

1. Depth-wise Separable Convolutions
   - Traditional convolutions are replaced by depth-wise separable convolutions which factorize a standard convolution into:
     - Depth-wise Convolution: Applies a single filter to each input channel.
     - Pointwise Convolution (1x1): Combines the outputs of the depth-wise convolution.
   - This drastically reduces the number of parameters and computation.
2. Inverted Residuals
   - Instead of expanding and then reducing dimensions (as in ResNets), MobileNetV2 uses inverted residuals, which:
     - Start with a low-dimensional input.
     - Expand it to a high-dimensional space.
     - Apply depth-wise convolution.
     - Project it back to a low-dimensional output.
   - The input and output of this block are connected by a shortcut connection (residual connection), allowing gradients to flow efficiently during training.
3. Linear Bottlenecks
   - Unlike other networks that apply non-linearities (ReLU) after every layer, MobileNetV2 uses linear activation at the bottleneck layers to avoid losing valuable information in low-dimensional space.

2.5 Model Training

Both models were trained with Adam optimizer, early stopping, and appropriate callbacks.

CNN Model Training:

The custom CNN model was constructed using:

- Multiple convolutional layers (for feature extraction),
- MaxPooling layers (for down-sampling and reducing computation),
- Dropout layers (to prevent overfitting),
- Dense layers (for decision making),
- And a softmax output layer for multi-class classification.

The training involved feeding batches of augmented images using ImageDataGenerator for better generalization. The model was trained over multiple epochs using the categorical cross-entropy loss function and the Adam optimizer, with accuracy as the evaluation metric.

Transfer Learning with MobileNetV2:

To improve accuracy and reduce training time, MobileNetV2—a pre-trained model trained on ImageNet—was fine-tuned for the bird classification task. Transfer learning offers the following benefits:

- Leverages previously learned visual features from a large dataset (ImageNet),
- Requires less data and training time compared to building a model from scratch,
- Produces better generalization on small datasets.

In this work:

- The base MobileNetV2 model was loaded without its top (classification) layer.
- A new classification head was added to match the 20 output classes.
- The model was trained using fine-tuning, where the base model layers were either frozen (initially) or unfrozen partially for deeper training.

Training Pipeline:

1. Preprocessing: Images were resized to 256x256 pixels and normalized.
2. Augmentation: Real-time augmentations like flipping, rotation, and zoom were applied using Image Data Generator.
3. Model Training:
   - Both models were trained using model.fit() over a fixed number of epochs.
   - Validation data was used to monitor overfitting.
   - Model checkpoints and early stopping were optionally used.

4. Evaluation:
   - Accuracy and loss were plotted for both training and validation sets.
   - Performance was evaluated on the test dataset with unseen images.

2.6 Model Conversion and Web Deployment

The MobileNetV2 model was converted to .tflite and deployed via Flask, where the user uploads an image and receives the prediction with confidence.

III. Analysis

3.1 Some random images from the dataset

3.2 Class Imbalance Analysis:

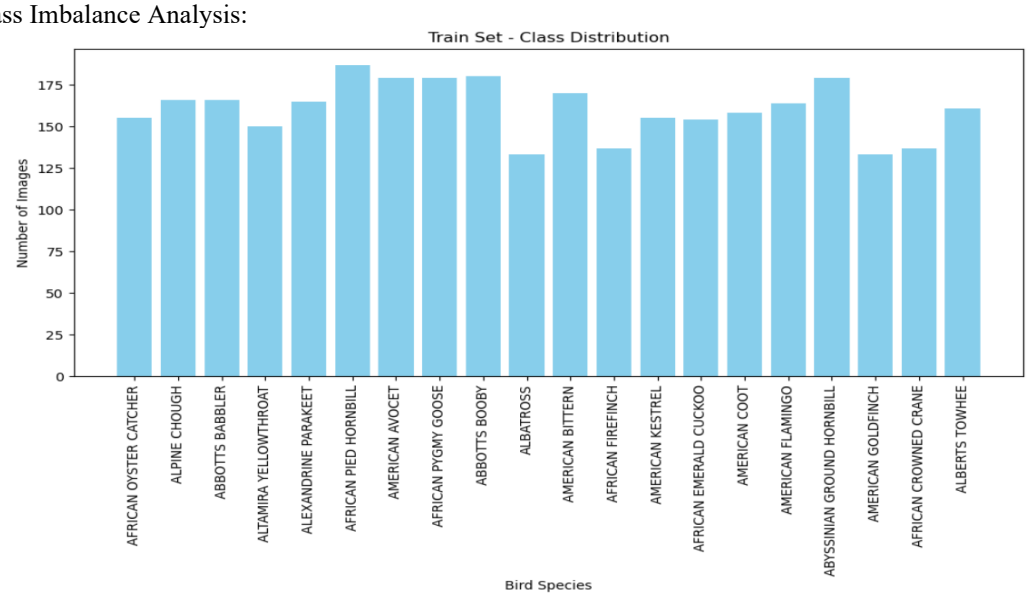

Fig 1

Fig 1 shows that there is no such significant class imbalance among the 20 classes in the training data.

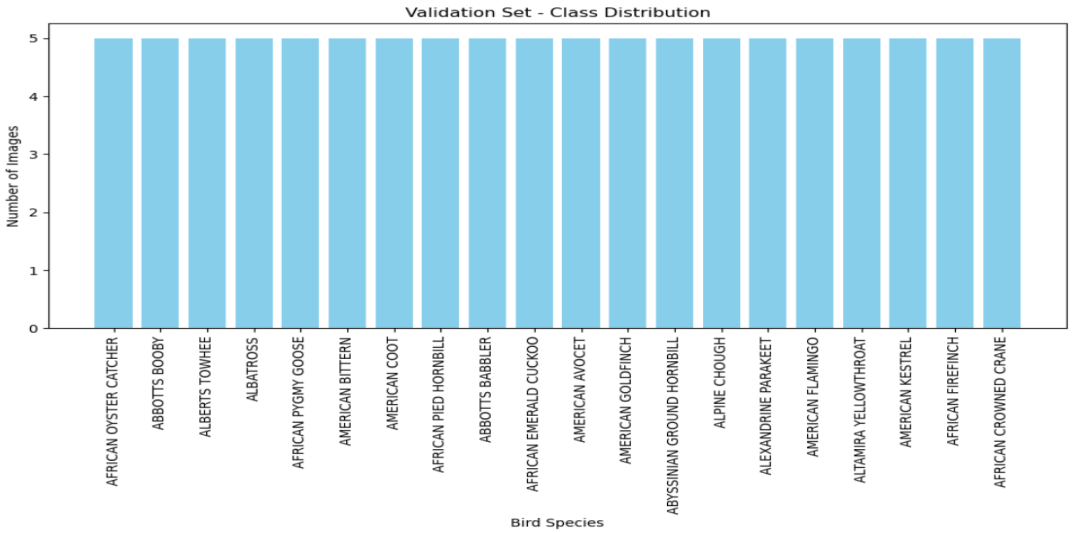

Fig 2

Fig 2 shows that all the 20 classes of the validation data are equally balanced.

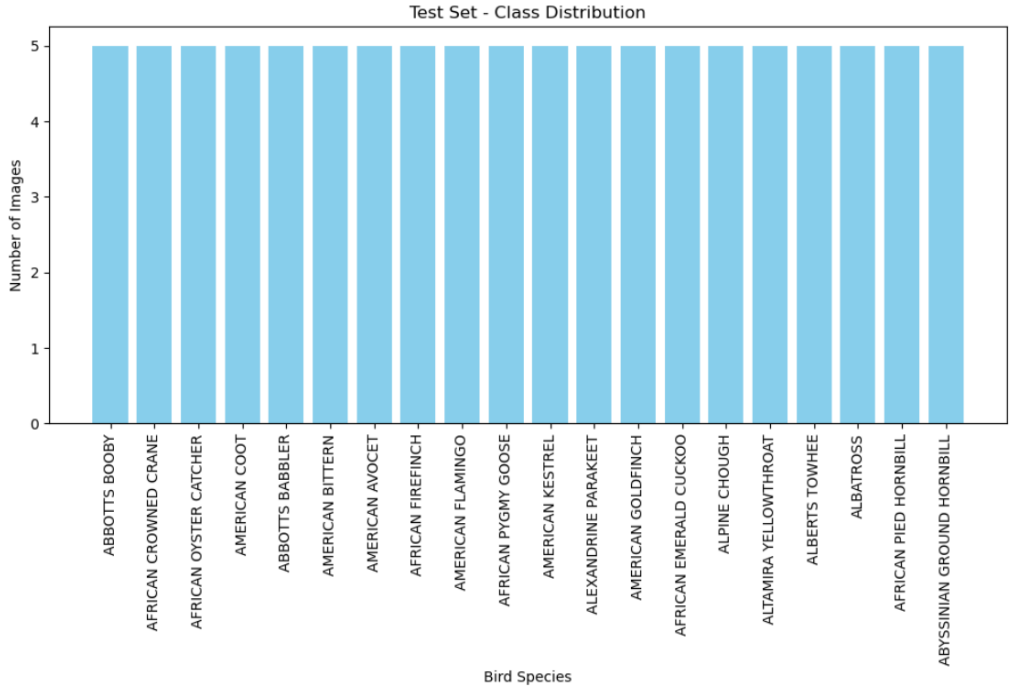

Fig 3

Fig 3 shows that all the 20 classes of the testing data are equally balanced.

3.3 Some random augmented images

Fig 4

3.4 Color Distribution Analysis

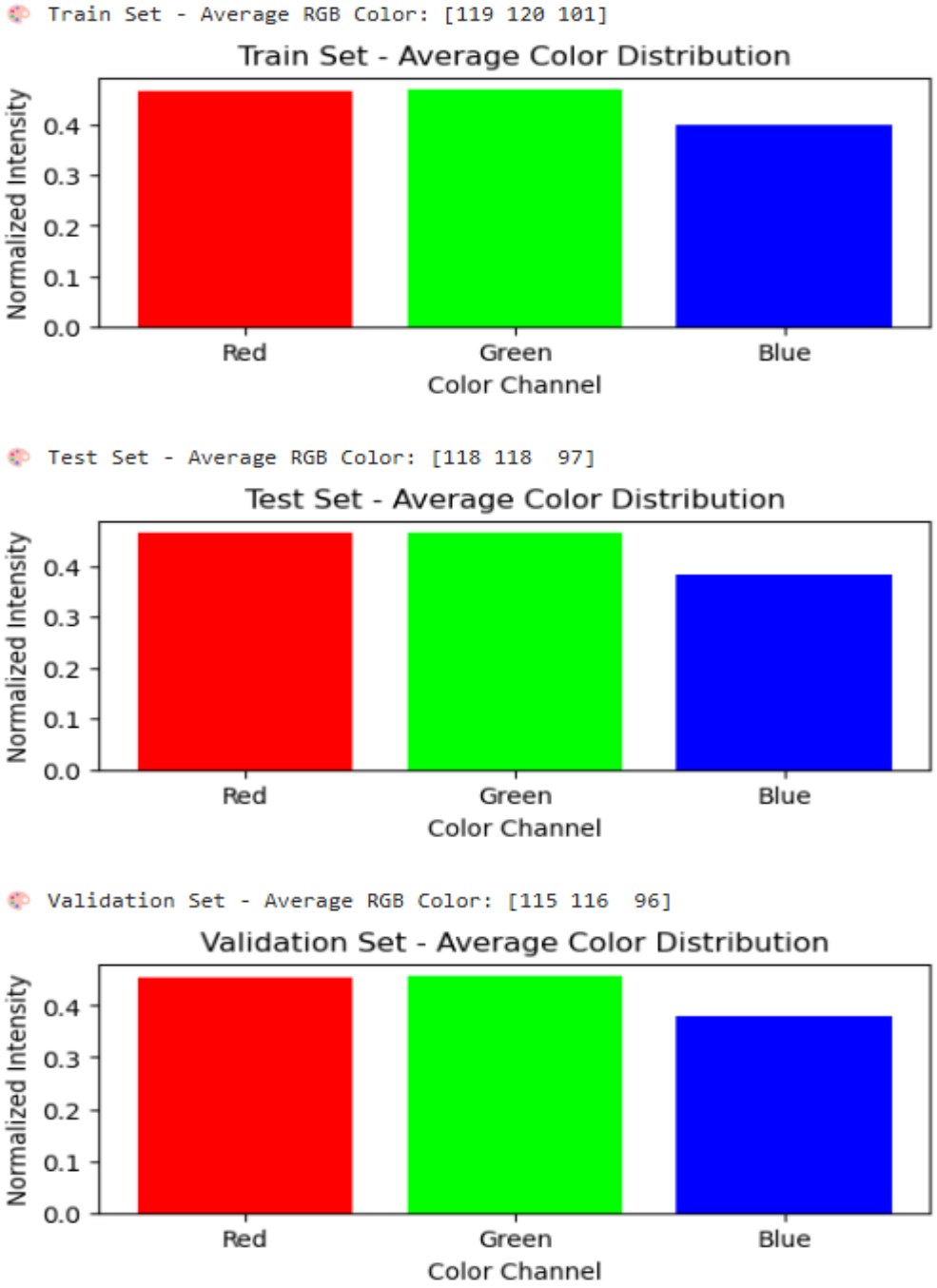

Fig 5

Fig 5 shows the distribution of colors of the images present in each of Training, Validation and Testing dataset.

3.5 Model Analysis

1) MobileNetV2 architecture

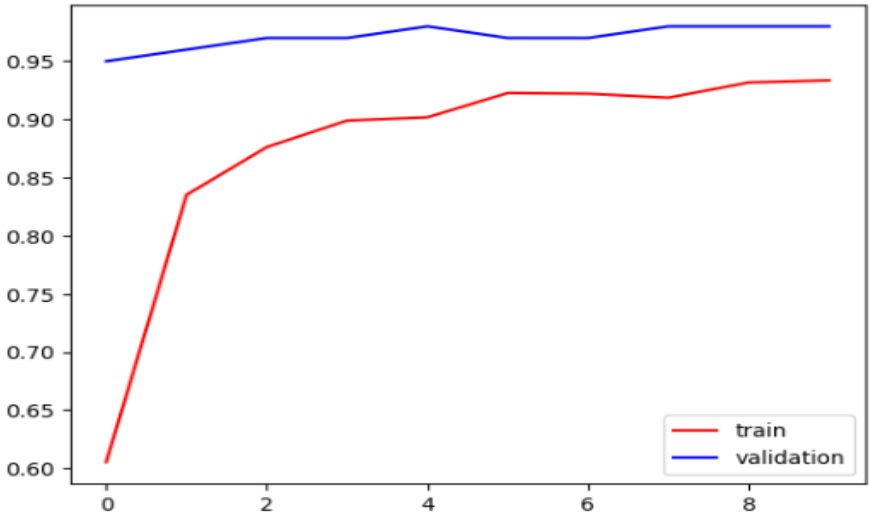

Fig 6

The training accuracy is about 93% and the validation accuracy is about 98%

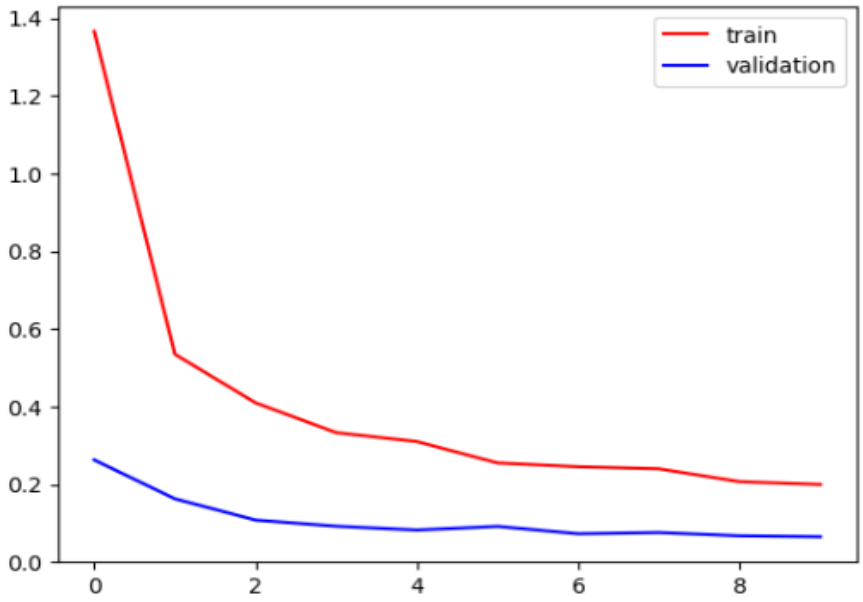

Fig 7

The training loss is about 0.2 and the validation loss is about 0.4

2) CNN architecture

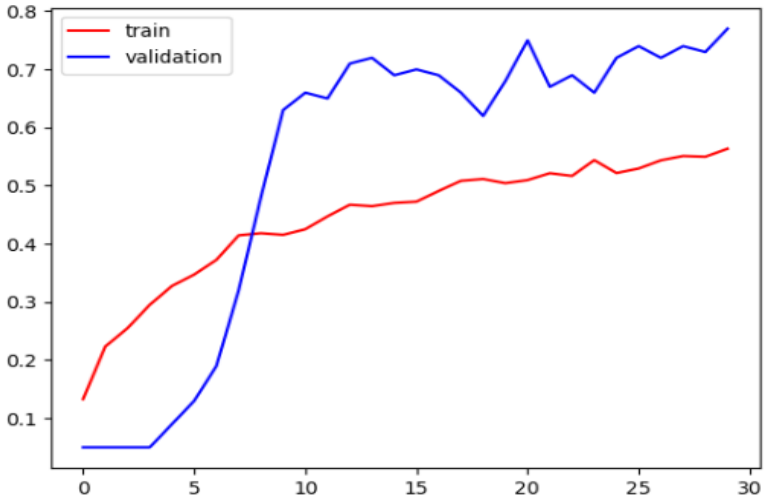

Fig 8

The training accuracy is about 55% and the validation accuracy is about 78%

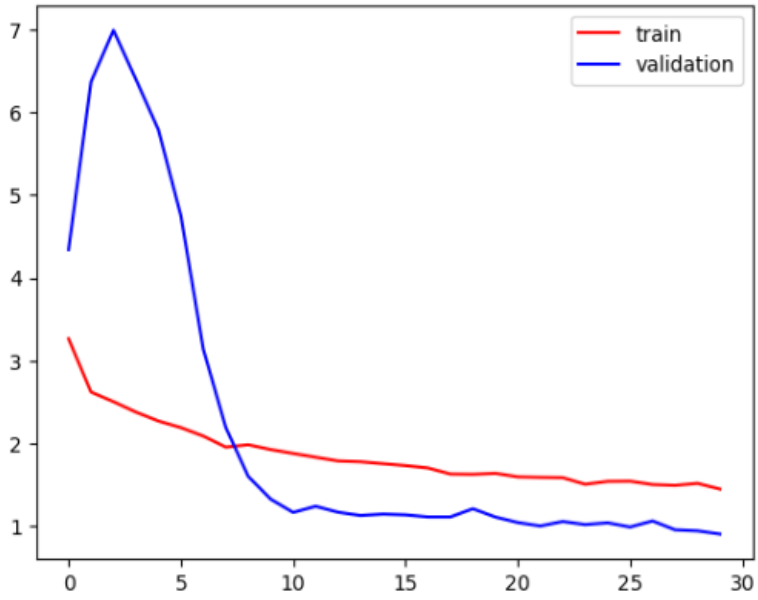

Fig 9

The training loss is about 1.7 and the validation loss is about 1

The accuracy on testing data for MobileNetV2 model is 99% while for CNN model the accuracy is 82%.

IV. Discussion

The comparative study reveals the benefits of using a pre-trained model like MobileNetV2 over a custom CNN for bird classification:

- Accuracy: MobileNetV2 showed better generalization on unseen data.
- Efficiency: Smaller size and faster inference after TFLite conversion.
- Ease of Deployment: Easily integrated into web apps and mobile-compatible.

Why MobileNetV2 Outperforms CNN:

1. Transfer Learning on a Large Dataset (ImageNet)

- MobileNetV2 is pre-trained on ImageNet, which contains over 1.2 million images across 1000 classes.
- It has already learned to detect general features like edges, colors, textures, shapes, and even more abstract features from various objects—including birds.
- This head-start in feature learning gives MobileNetV2 a huge advantage when fine-tuned on your relatively small dataset.

In contrast, your CNN model starts from scratch, so it needs more data and training to learn these same features from the ground up.

2. Better Feature Representation

- MobileNetV2 uses inverted residual blocks and depthwise separable convolutions, which help:
  - Preserve important spatial features,
  - Reduce computation,
  - And allow deeper yet more efficient learning.
- These architectural improvements help it capture subtle differences between bird species, which are often visually similar.

A standard CNN may fail to capture high-level, fine-grained patterns, especially with limited data.

3. Regularization and Generalization

- Pre-trained models like MobileNetV2 are better regularized by design.
- Your CNN might overfit to the training data unless carefully tuned with techniques like dropout, data augmentation, or weight regularization.

MobileNetV2 generalizes better, especially when the training data is limited and the problem involves high intra-class similarity (like birds).

4. Efficient Use of Limited Data

- MobileNetV2 requires fewer parameters and is optimized for small datasets and mobile environments.
- It converges faster, needs fewer epochs, and provides better results without a massive dataset.

In comparison, training a deep CNN from scratch often requires a large, diverse dataset, longer training time, and more computational power.

A critical addition is the "unknown species" detection, where the app avoids false positives by analyzing the softmax confidence. This improves usability in real-world scenarios.

V. Resources

1) Kaggle Dataset Link: https://www.kaggle.com/datasets/umairshahpirzada/birds-20-species-image-classification

2) GitHub Repository link (Click on this link to view all the codes and files used to develop this project and deploy): https://github.com/Dip1502/Bird-Classification

3) The live URL for the web application deployed on Render Cloud platform:
https://bird-classification-3h56.onrender.com
(The page may take 1-2 minutes time to load due to Render traffic.)

VI. Conclusion

This study demonstrates that MobileNetV2, combined with proper data augmentation and fine-tuning, provides robust and accurate bird species classification. Deploying the model using Flask further showcases the feasibility of real-world applications. The dataset is taken from Kaggle which is an open source. Hence the dataset is relatively small covering only 20 different species of birds. But if the variation of species and the number of images can be increased in the dataset then this idea and approach can be extended into a large scale and can have a huge impact in the field of Ecology and Environmental Science. For much larger datasets, especially those with high variability and complexity, it is often more effective to use a powerful and deeper CNN architecture (such as ResNet, Inception, or a custom deep CNN) than lightweight models like MobileNetV2. These complex models can capture more intricate patterns and benefit more from abundant data, whereas MobileNetV2 is optimized for speed and efficiency on smaller or resource-constrained settings.

References

[1]. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. CVPR.
[2]. Chollet, F. (2017). Xception: Deep Learning with Depthwise Separable Convolutions. CVPR.
[3]. TensorFlow Documentation. https://www.tensorflow.org
[4]. Brownlee, J. (2019). Deep Learning for Computer Vision: Image Classification Projects. Machine Learning Mastery.
[5]. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. CVPR.